DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
REPORT A-2013-3

A Model-Driven Approach to Service Ecosystem Engineering

Toni Ruokolainen

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XIV, University of Helsinki Main Building, on 22 February 2013 at noon.

UNIVERSITY OF HELSINKI
FINLAND

Supervisor
Lea Kutvonen, University of Helsinki, Finland

Pre-examiners
João Paulo A. Almeida, Federal University of Espírito Santo, Brazil
Antonio Vallecillo, Universidad de Málaga, Spain

Opponent
Marten van Sinderen, University of Twente, Netherlands

Custos
Sasu Tarkoma, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.Helsinki.fi/
Telephone: +358 9 1911, telefax: +358 9 191 51120

ISSN 1238-8645
ISBN 978-952-10-8620-5 (paperback)
ISBN 978-952-10-8621-2 (PDF)
Computing Reviews (1998) Classification: D.2.1, D.2.11, D.2.12, I.6.5
Helsinki 2013
Unigrafia

A Model-Driven Approach to Service Ecosystem Engineering

Toni Ruokolainen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
Toni.Ruokolainen@cs.Helsinki.FI
http://www.cs.helsinki.fi/Toni.Ruokolainen

PhD Thesis, Series of Publications A, Report A-2013-3
Helsinki, February 2013, 232 pages
ISSN 1238-8645
ISBN 978-952-10-8620-5 (paperback)
ISBN 978-952-10-8621-2 (PDF)

Abstract

The passage from monolithic, product-driven business models to networked and service-based business has given rise to the emergence of service ecosystems. A service ecosystem is a socio-technical complex system that enables service-based collaborations between entities such as enterprises, institutions or individuals. Service ecosystems may emerge spontaneously due to a common interest or demand, or as a result of long-term strategic planning. Examples of service ecosystems include software ecosystems, electronic business networks, cloud computing platforms and social networking platforms. The emergence of service ecosystems has been driven especially by technological developments and innovations. Advancement in areas such as service-oriented computing and cloud computing has provided foundations for implementation and operation of contemporary service ecosystems.

Contemporary service ecosystems are now challenged especially on two fronts. On one hand, networked business and networked business models require support especially for efficient business decision making, opportunistic and flexible business networking and service ecosystem governance. Such activities require explicit knowledge about the capabilities and other characteristics of service ecosystems. On the other hand, the visions of Internet of Things and Internet of Services necessitate means for efficient, on-demand establishment of domain-specific service ecosystems, as well as loosely coupled connectivity and interoperability both in and between distinct service ecosystems. Due to the implicit nature of contemporary service ecosystem architectures, and technology-driven engineer-

ing processes the current service ecosystems are at risk to become yet new technological and conceptual silos.

The current technology-driven approaches for establishing service ecosystems are infeasible to meet these challenges. From an engineering perspective, there is an evident lack of a service ecosystem engineering discipline. The prevailing technology-driven approaches stress the implementation and operation of service ecosystem infrastructure while neglecting analysis and design of service ecosystems as holistic systems. Consequently, service ecosystems are established based on implicit and technology-dependent architectures and design principles. Such *ad hoc* service ecosystems suffer from reduced longevity due to technology-dependent designs, as well as interoperability problems and inflexible governance practices due to ambiguous ecosystem architectures. There is a need for a systems engineering discipline which provides means for analysis, design, instrumentation and operation of service ecosystems.

From a business perspective, the current approaches for service ecosystem establishment do not provide sufficient support for ecosystem sustainability. Sustainability is defined as a quality of a service ecosystem to support continued viability. The viability of a service ecosystem can only be established if the concerns of stakeholders, e.g. service providers and service engineers, are feasibly addressed during the service ecosystem engineering life cycles. For example, service providers must be provided with means for service contracting and bundling while service engineers must be provided with facilities that enable well-advised and efficient service production. Guaranteeing continuation of ecosystem viability necessitates e.g. preservation of member autonomy and dynamic business service collaboration capabilities for allowing efficient utilization of core competencies and exploitation of new business opportunities.

This thesis proposes service ecosystem engineering as a novel systems engineering discipline for enabling a transition from contemporary *ad hoc* service ecosystems to sustainable ones. Such a transition is required for supporting modern networked business practices and the “services everywhere” vision. The service ecosystem engineering discipline is realized with a model-driven approach. The approach facilitates establishment of sustainable service ecosystems where service ecosystem viability is achieved by addressing the characteristic concerns of ecosystem stakeholders.

Service ecosystem engineering is defined as a systems engineering life cycle which comprises phases of ecosystem analysis, design, instrumentation and operation. The model-driven approach for service ecosystem engineering is based on a holistic conceptual model of service ecosystems, which in itself supports service ecosystem analysis. The conceptual model is formalized as a meta-model which enables further design and utilization of explicit service ecosystem architecture models. The conceptual model and the meta-model provide foundations for the Service Ecosystem Architecture Framework (SEAF), which enables service ecosystem architecture analysis and design. SEAF is implemented over a commercial UML-modelling tool. Finally, a model-driven methodology for service ecosystem instrumentation is defined. Service ecosystem instrumentation denotes delivery of the infrastructure necessary for enabling the operation of the ecosystem. Service ecosystem instrumentation may involve for example implementation of infrastructure services and engineering tools, and configuration of middleware platforms. The methodology utilizes the architecture models and model-driven engineering practices for instrumentation of domain-specific service ecosystems. In addition, service ecosystem architecture models can be applied for service ecosystem governance approaches, as well as for establishing interoperability and predictability of ecosystem operation.

The impact of this work is two-fold. First of all, the results of this work facilitate rigorous engineering of sustainable service ecosystems. Enabling analysis and design of service ecosystems as holistic systems decreases the risks associated for example with technology dependency and migration. Explicit service ecosystem architectures enable efficient analysis techniques, e.g. assessment of service ecosystem maturity with respect to local criteria, to support business decision making when joining and operating in service ecosystems. Moreover, service ecosystem architecture models are utilizable for model-based interoperability approaches, as well as for supporting service ecosystem governance.

Secondly, added value and utility is provided for service ecosystem stakeholders. Ecosystem members are delivered with means for efficient utilization of core competencies, opportunistic and flexible business networking, supporting progressive business environments, and efficient business decision making. As demonstrated in the evaluation part of this thesis, these activities are addressed by the approach in service ecosystem engineering life cycles.

Computing Reviews (1998) Categories and Subject Descriptors:
D.2.1 Requirements / Specifications: Methodologies
D.2.11 Software Architectures: Domain-specific Architectures
D.2.12 Interoperability
I.6.5 Model Development

General Terms:
service ecosystems, service ecosystem engineering, model-driven engineering, service ecosystem sustainability, open service ecosystems, service-oriented computing

Additional Key Words and Phrases:
architecture framework, meta-modelling, model-driven methodology, systems engineering, service ecosystem meta-model, foundational ontology

Acknowledgements

This book was written within a rich and complex socio-technical ecosystem. Now that the work is done I want to thank all the people involved in this co-creation experience.

First of all, I would like to thank my supervisor Lea Kutvonen for being patient and visionary enough to let me find my own path as a researcher. I especially appreciate the discussions that have happened at the "meta-level", providing me guidance how to proceed both methodologically and as an expert on our field that is a strange mixture of computer science, information systems, mathematics, economics and sometimes even philosophy.

I would like to thank all the former and current CINCO members for all the support and discussions. I feel especially lucky that I have had the opportunity to have really inspiring talks with my colleague Tuomas Nurmela. His industrial expertise and knowledge of economics and service business has certainly had an effect on my thinking along the way. I hope that our collaboration will carry on.

I have had the pleasure of working at the best computer science department in Finland. I am grateful for all the support I have had from the Department of Computer Science. While I have not been the most talkative person in the coffee room I have really enjoyed the off-topic discussions while enjoying a break from deep thinking or sometimes even agonizing verbalization of thoughts.

I thank my pre-examiners, João Paulo A. Almeida and Antonio Vallecillo, for their valuable feedback and encouraging comments. Their expertise shows in the book especially in enhanced comprehensibility and transparency of the contributions. I am honored to have the opportunity to defend my thesis in front of my assigned opponent, Marten van Sinderen, whose long research career especially in the area of specification and modelling of distributed systems shows also in this book.

I am blessed to have an extended family whose support has enabled me to proceed on my research career. They have taken especially good care of my daughter Aada who has taught me more that I can ever teach her back. She has taught me especially about sincerity, presence and joy of life. I am also grateful to all the other teachers I have had, and still have, in my life. You have shown me valuable

things about myself and the world around us. And of course, I am thankful to all my friends who I have had the honour to grow up and have fun with.

Finally, I would like to express my eternal gratitude to my late mother who brought me up to appreciate the simplicity of life.

Raasepori, 27 January 2013
Toni Ruokolainen

Contents

Chapter 1

Introduction

The passage from monolithic, product-driven business models to networked and service-based business has given rise to the emergence of service ecosystems. A service ecosystem is a socio-technical complex system that enables service-based collaborations between entities such as enterprises, institutions or individuals. Service ecosystems may emerge spontaneously due to a common interest or demand, or as a result of long-term strategic planning. Examples of service ecosystems include software ecosystems, electronic business networks, cloud computing platforms and social networking platforms.

This academic dissertation provides model-driven facilities for service ecosystem engineering. The contributions of this thesis support a transition from contemporary, technology-driven *ad hoc* service ecosystems to sustainable service ecosystems. Such transition is a pre-requisite for reaching the Internet of Services envisioned by the European Commission. A sustainable service ecosystem is a well-defined, governed system with an explicit architecture that addresses the fundamental concerns of ecosystem stakeholders, and provides means for maintaining viability of ecosystem operation.

This Chapter introduces the context, motivation and contents of the dissertation. First in Section 1.1 the notion of service ecosystems is introduced. After that, in Section 1.2, an application scenario is introduced for clarifying the demand for the results of this thesis. Section 1.3 identifies research research challenges which need to be addressed. Motivation and objectives for this work are introduced in Section 1.4 which formulates the concrete research questions for this dissertation. The approach for reaching the objectives is described in Section 1.5. Related research fields are identified in Section 1.6. The structure of the dissertation is described in Section 1.7. Finally, the research history that lead to this academic dissertation is described in Section 1.8.

1.1 Emergence of service ecosystems

The current business landscape is characterized by commoditization and shortened product life spans [30]. In an effort towards increasing their revenues in this landscape, many enterprises are moving to service-centric business models and especially the utilization of electronic services. Also public organizations operating in domains like health-care (and government) provide and utilize electronic services for increasing their coverage, efficiency and customer satisfaction. Even individuals consume electronic services provided by governmental bodies, commercial actors, and communities of interest. These services include different forms of information sharing and social networking services.

The passage from monolithic, product-driven business models to networked and service-based business has given rise to the emergence of service ecosystems. A service ecosystem is a socio-technical complex system that enables service-based collaborations between entities such as enterprises, institutions or individuals. A service is here considered as an autonomous, self-descriptive software component with a well-defined interface [151] that delivers a set of capabilities to fulfill the business needs of a client. Service ecosystems may emerge spontaneously due to a common interest or demand, or as a result of long-term strategic planning.

Examples of service ecosystems include software ecosystems, service-based collaborative networks, and web-application platforms. Software ecosystems have become popular as a means for producing software applications more efficiently for heterogeneous clientele with varying requirements. A software ecosystem is typically based on a software platform provided by an organization. The platform is then used by internal and external developers for implementation of applications [22]. Infrastructure services are provided by the platform owner for application provisioning and distribution. Software ecosystem strategy is utilized by companies such as Apple[1] or Nokia[2] for establishing communities of developers and clientele over their own corresponding platforms.

Organizations participate in service-based collaborative networks which are established to share risks, development costs or intellectual capital, or to complement each others' skills [157, 27]. A common form of collaborative network in a business context is a supply chain coordinated and managed by a dominant player of the domain [157]. In other kinds of scenarios, such as formal joint ventures, professional virtual communities, or collaborative engineering environments [101], collaborations need to be formed in a more loosely coupled and democratic manner. Such collaborative networks can be realized as different kinds

[1] http://www.apple.com
[2] Nokia Ovi: http://www.ovi.com/

of virtual organizations [28], for example.

The *Web 2.0* (see e.g. [105]) emphasizes provision of Web-based applications and their *mash-ups* (see e.g. [176]) over the Internet. Typical Web 2.0 applications can be classified into social networking and knowledge sharing services, tools and platforms for end-user content creation, and on-line collaboration tools for specific tasks [146]. Especially social networking platforms have become popular, since they typically allow light-weight and easy methods for sharing knowledge within different communities of interest. Social networking platforms, such as YouTube[3], Facebook[4] or LinkedIn[5], provide infrastructure services for sharing knowledge, such as video feeds, user profiles, personal information, or competencies of individuals. Additional services, such as discussion forums for users, are typically provided by social networking platforms.

The emergence of service ecosystems has been driven especially by technological developments and innovations, such as the Web [173] and Web Services [169], and generalization of communication technology. Advancement in the areas of service-oriented computing and cloud computing has provided foundations for implementation and operation of contemporary service ecosystems. Service-oriented computing is a paradigm for designing and implementing complex distributed systems [122, 151] that is based on the concept of services. The service-oriented computing paradigm comprises four conceptual elements: services, service descriptions, service composition, and service-oriented architectures (SOA). Services are advertised by publishing their descriptions in service brokers. Service descriptions are produced by service providers and they characterize the properties and capabilities of corresponding services. Service consumers use service discovery mechanisms provided by a service broker infrastructure to locate appropriate services. Other kinds of infrastructure services, such as repositories for domain vocabularies, can also be available in service ecosystems. Finally, SOA is an architectural style, i.e. a collection of design and implementation principles, for design and development of service-based systems. The design principles of SOA seek to attain loose coupling and late binding between software services.

Cloud computing is an approach where computing, storage and software application resources are delivered to customers as services [166]. Cloud computing is based on abstraction of computing infrastructure providing an elastic capacity and illusion of infinite computing resources, self-service interfaces, and a pay-per-use model of resource utilization [166]. The cloud computing approach includes three service models based on the abstraction level of capabilities and services

[3] YouTube: http://www.youtube.com
[4] Facebook: http://www.facebook.com
[5] LinkedIn: http://www.linkedin.com

delivered to customers. Infrastructure as a Service (IaaS) refers to a cloud computing service model where the primary capabilities delivered are associated with computing infrastructure, such as storage and networking. The capabilities are managed and accessed through a virtual infrastructure manager [166]. The Amazon EC2[6] is an example of an IaaS-platform. In the Platform as a Service (PaaS) model the delivered capabilities include programming languages and frameworks, and structured data. The PaaS services are accessed and managed through a cloud development environment which provides a platform for service developers to implement and deploy software services. Google AppEngine[7] can be considered a PaaS-platform. The Software as a Service (SaaS) model is characterized by web-browser accessible consumer services, such as social networks, office suites, or video processing applications [166].

Contemporary service ecosystems can be characterized as closed systems. That is, either the set of available services is restricted to specific kinds of services that are pre-determined before ecosystem operation, or then utilization and composition of services is based on manual service discovery and static, development-time binding of service interfaces. Interoperability between services is established manually and pre-determined before actual operation; tightly coupled technology-level integration is used for establishing interoperable service compositions. Current service ecosystems typically do not support evolution (i.e. introduction of new kinds of services and collaboration forms), dynamic service discovery and composition, or run-time collaboration establishment, all of which are considered crucial elements in the vision of service ecosystems, service-oriented computing [122, 123] and Internet of Services [146, 92, 42].

In future, organizations and individuals collaborate within open service ecosystems for enabling service-oriented networked business. An open service ecosystem is characterized especially by the autonomy of its entities, its evolution with respect to available services and collaboration types, and dynamic establishment of collaborations. In an open service ecosystem the service providers and clients are not bound to a shared development platform. Instead, each ecosystem member may utilize methods and technologies that suit their own needs best. A set of global infrastructure services are then used for service publication and discovery, as well as dynamic establishment of service-based collaboration networks [87]. An open service ecosystem is based on the SOA architectural style with service brokering and dynamic binding facilities, but requires in addition more sophisticated infrastructure services for enabling interoperable service collaboration.

In open service ecosystems means for sharing and maintaining *interoperabil-*

[6] Amazon EC2: http://aws.amazon.com/ec2/

[7] https://developers.google.com/appengine/

ity knowledge need to be provided. We define *knowledge* as explicit and formalized information that is shared within a community of entities having a common understanding of the information and its purpose. Knowledge is based on the conceptual framework of the community and is specific for the corresponding domain of interest. Interoperability knowledge is utilized in open service ecosystems for guaranteeing interoperable operation of service-based collaborations. Interoperability knowledge includes information about service compatibility with other services and their applicability with respect to different models of collaboration, for example. Interoperability knowledge is typically public or shared within a community of ecosystem members, and it is dependable in the sense that it comes from trusted sources and is verified with respect to some domain-specific formal criteria.

Services, service-based systems and other artefacts instrumenting service-based collaborations are envisioned to be produced in open service ecosystems by globalised software engineering processes. From such a service-oriented software engineering perspective this dissertation provides means for sharing *engineering knowledge* and for instrumenting service-oriented software engineering frameworks. Engineering knowledge is characterized by its design and development time usage and its context dependence: a knowledge fragment can be meaningful only within a specific phase of an engineering process. Complete, independent knowledge artefacts are composed from such incomplete knowledge fragments within knowledge management and software engineering processes. Engineering knowledge, as opposed to interoperability knowledge, is not necessarily verified all the time due to its possible incompleteness.

1.2 Engineering a home-automation service ecosystem

The demand for results produced in this thesis are clarified by a running example. The application scenario involves home-automation for supporting so-called active ageing. Active ageing is *"is the process of optimizing opportunities for health, participation and security in order to enhance quality of life as people age. It applies to both individuals and population groups"* [177]. Home-automation is one of the applications that especially can utilize the Internet of Things [43] for establishing connectivity between home-appliances, end-user terminals and interfaces, and services available in the Internet.

In western countries, such as Finland, ageing of the society is nowadays a great, multi-faceted challenge to be solved. Let us assume that a governmental body decides that ageing should be addressed in the nation by supporting active ageing, thus enhancing the quality of living for elderly people. The governmental body has acknowledged the national strategy for utilizing service-orientation and

government-supported service platform to increase efficiency and competitiveness of national companies [8].

The governmental body decides that they should support the emergence of an open service ecosystem for home-automation. Thus, a working group is created for analysing the requirements for such a strategy. The working group consists of selected experts in the domains of service-based business, health-care, and government. Their first task is to consider what kinds of end-user services are critical in the context of home-automation for active ageing.

The working group creates a vision of the service ecosystem by analysing for example the business ecosystem surrounding home-automation and the social system around aging people. The working group identifies different actors in the service ecosystem vision, such as "Doctor", "Relative" or "Practical nurse" and appliances such as "Health monitor", "Tablet computer" or "Tracking device". After that, the working group analyses what kind of service infrastructure is needed for supporting the delivery of such services, what kinds of actors produce and consume those services, and are there any third-parties needed for delivering critical capabilities to the ecosystem. Examples of different engineering activities, contextual information and stakeholders are illustrated at the top of Figure 1.1.

For the activity of service ecosystem analysis, the working group needs a consistent and complete conceptual framework. Without such conceptual framework for service ecosystems, the domain-experts may fail to find and address critical components of the service ecosystem. Especially, dependencies between different service ecosystem elements, such as actors, services or life cycles, might get unnoticed or ambiguous. This thesis defines a conceptual framework for service ecosystems that is utilizable for service ecosystem engineering; the conceptual framework is discussed in Chapter 2. With the activity of service ecosystem engineering the service ecosystem vision is turned into a more concrete and unambiguous service ecosystem conceptualization by the working group, as illustrated in Figure 1.1.

After conducting the analysis of the service ecosystem the findings of the working group are documented as a service ecosystem conceptualization, or initial design. The working group now addresses a larger community of stakeholders by issuing a request for comments and proposals. The extended stakeholder community includes for example additional industry members (potential service providers), and both governmental and non-governmental organizations. The stakeholders include for example potential service ecosystem infrastructure providers, such as telecommunication operators. Based on the feedback on requests a consortium is founded for creating a more formal design of the home-automation

[8] Such an strategy has been suggested as of 2013 in Finland. The strategy was prepared under a workgroup of the Ministry of Employment and Economy: http://tem.fi (in Finnish)

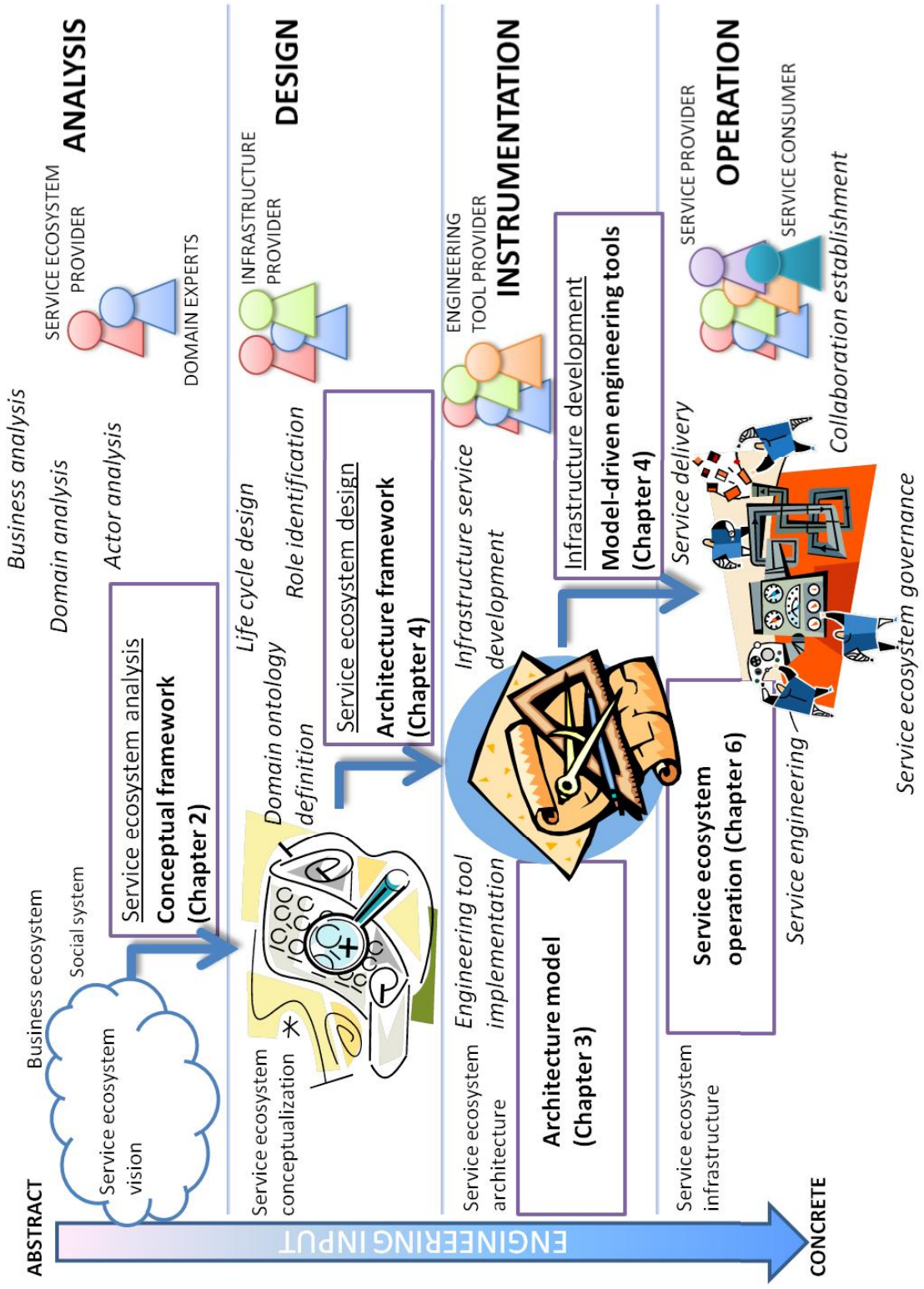

Figure 1.1: Illustrating service ecosystem engineering.

service ecosystem for active ageing.

The first task of the consortium is to refine and formalize the service ecosystem conceptualization. As the service ecosystem design is now collaboratively worked upon in a large community of stakeholders, the effectiveness of design work and unambiguity of concepts as well as design representation become important issues. Design work should be conducted with an appropriate modelling language that supports experts in different domains to share and develop the service ecosystem concepts and elements. Without such a design environment, the work of the consortium become inefficient due to lack of a shared vocabulary and its representation, and design practices.

To address the above challenges, the consortium decides to use a well-defined architecture framework for the design work. An architecture framework is an engineering tool which establishes common practise for creating, interpreting, analyzing and using architecture descriptions within a specific domain or community of stakeholders [72]. The consortium utilizes an architecture framework for service ecosystems to describe the concepts needed in the home-automation ecosystem (especially what kinds of service concepts are required), defining the ecosystem life-cycles and processes, and identification of the fundamental roles the different ecosystem actors may take (e.g. an elder may take the role of "Patient" or "Resident" in life cycles associated with hospitals or residences, correspondingly).

As illustrated in Figure 1.1, the design activities take the service ecosystem conceptualization as an input and produce a service ecosystem architecture description as an output. The architecture framework defined in Chapter 4 is utilizable for supporting such service ecosystem design work. It provides the means for the creation of service ecosystem designs, as well as utilization of shared modelling practices. The service ecosystem architecture description is made explicit in a form of an architecture model. The architecture model enables formalization and sharing of the service ecosystem design among the stakeholders. The foundations for such architecture description models are provided in form a meta-model which is defined in Chapter 3.

After the consortium is finished with the design work, the resulting service ecosystem architecture description is used as an engineering input for developing the infrastructure required for service ecosystem operation. The service ecosystem infrastructure especially includes infrastructure services. Infrastructure services, such as registries maintaining the collection of offerings for home-automation services, provide ecosystem members with means for delivery of the end-user services. The consortia decides that for establishing trust among the end-users, the elderly people, the infrastructure services should be provided by a government-lead organization. The governmental organization becomes thus an

ecosystem provider.

However, the ecosystem provider organization does not have the necessary skills to develop the infrastructure services. Thus it contacts various service engineers and provides them with the service ecosystem architecture model. The service engineers use the architecture model to implement the required software components for the infrastructure services. Engineering tools are utilized by service engineers and service providers to design and implement services that conform with the requirements and characteristics of the home-automation service ecosystem. Ecosystem specific engineering tools are now needed for enabling efficient production of services for the home-automation service ecosystem. The consortium contacts potential engineering tool providers and gives them access to the service ecosystem design (or makes the design public). Domain-specific service engineering methods that meet the regulations and quality requirements of the home-automation service ecosystem are part of the design; they are used by both service engineers and engineering tool providers in their activities.

Efficient means for instrumentation of a service ecosystem is a deciding factor for the success of a service ecosystem. Service ecosystem instrumentation denotes delivery of the infrastructure necessary for enabling the operation of the ecosystem. Service ecosystem instrumentation may involve for example implementation of infrastructure services and engineering tools, and configuration of middleware platforms with model-driven approaches such as [4]. A model-driven methodology enabling efficient service ecosystem instrumentation is defined in Chapter 4.

Finally, when the home-automation service ecosystem for active aging is instrumented with the necessary infrastructure, the service ecosystem can become operational. During the service ecosystem operation different stakeholders require knowledge about the state and characteristics of the service ecosystem. Knowledge is needed for example to discover appropriate business services, coordination of collaboration between services, and governance of the service ecosystem as a whole. Without any means for describing and maintaining such knowledge, service delivery in the service ecosystem becomes problematic, thus decreasing its sustainability and value to stakeholders (especially service consumers).

The model-driven methodology for service ecosystem engineering that is implemented as part of this thesis provides means for establishing a knowledge management infrastructure for service ecosystems. Moreover, the explicit service ecosystem architecture models can be utilized during service ecosystem operation for coordinating the collaboration between services, as well as establishing interoperability with so-called model-driven interoperability approaches (see e.g. [19]). Finally, the service ecosystem architecture descriptions can be used for the governance of the service ecosystem [143].

The above example illustrates the different kinds of activities associated with establishment of service ecosystems. In this thesis a methodology is defined which supports all the different activities discussed above. The methodology is model-driven: service ecosystem architecture models developed during the analysis and design activities are utilized for generation of various software engineering artefacts during service ecosystem instrumentation. Service ecosystem operation is not discussed explicitly in this thesis; however, the application of service ecosystem models is discussed briefly in Chapter 6.

1.3 Identifying the challenges

Contemporary service ecosystems are now challenged especially on two fronts. On one hand, networked business and networked business models require support especially for efficient business decision making, opportunistic and flexible business networking and service ecosystem governance. Such activities require explicit knowledge about the capabilities and other characteristics of service ecosystems. On the other hand, the visions of Internet of Things [43] and Internet of Services [42] necessitate means for efficient, on-demand establishment of domain-specific service ecosystems, as well as loosely coupled connectivity and interoperability both in and between distinct service ecosystems.

The current technology-driven approaches for establishing service ecosystems are infeasible to address the preceding advancements. Due to the implicit nature of contemporary service ecosystem architectures, and technology-driven engineering processes the current service ecosystems are at risk to becoming yet new technological and conceptual silos. The current situation calls for novel solutions for establishing and managing service ecosystems. For creating such a solution, three primary research challenges must be addressed. The challenges are:

1. establishment of a service ecosystem engineering discipline;
2. support for service ecosystem sustainability; and
3. knowledge management in open service ecosystems.

First, from an engineering perspective there is an evident lack of a service ecosystem engineering discipline. The prevailing technology-driven approaches stress the implementation and operation of service ecosystem infrastructure while neglecting analysis and design of service ecosystems as holistic systems. Consequently, service ecosystems are established based on implicit and technology-dependent architectures and design principles. Such *ad hoc* service ecosystems suffer from reduced longevity due to technology-dependent designs, as well as interoperability problems and inflexible governance practices due to ambiguous

ecosystem architectures. There is a need for a service ecosystem engineering discipline with an appropriate systems engineering life cycle. Thus, a rigorous service ecosystem engineering discipline must be provided which enables analysis, design, instrumentation and operation of service ecosystems. In this context rigorous engineering discipline means that the involved stakeholders are provided with means for formalizing architectural designs and knowledge, assessing architectural designs, as well as tracing design decisions and their implications. Such a rigour is enabled by explicit and formal service ecosystem architecture models.

Secondly, from a business perspective the current approaches for service ecosystem establishment do not provide sufficient support for ecosystem sustainability. Sustainability is defined as a quality of a service ecosystem to support continued viability [143]. Viability of a service ecosystem can only be established if concerns of stakeholders, e.g. service providers and service engineers, are feasibly addressed during the service ecosystem engineering life cycles. For example, service providers must be provided with means for service contracting and bundling while service engineers must be provided with facilities that enable well-advised and efficient service production. Guaranteeing continuation of ecosystem viability necessitates e.g. preservation of member autonomy and dynamic business service collaboration capabilities for allowing efficient utilization of core competencies and exploitation of new business opportunities.

Thirdly, contemporary service ecosystems lack means for establishing the knowledge management practices required for open service ecosystems. Management of service ecosystem knowledge comprises the standard activities of *1)* knowledge creation, *2)* knowledge integration and *3)* knowledge dissemination [46]. Regarding creation of ecosystem knowledge a lot of relevant research has been conducted. Different kinds of modelling languages have been specified for definition of services, service compositions, and service-oriented architectures (e.g. [126, 12, 14]). Methods for creating service engineering artefacts have been provided in the form of design and development principles [124], reusable service development patterns [180], and methodological frameworks [94], for example. Tools have been created for service development purposes which utilize different approaches, such as the semantic web services approach [78], or model-driven engineering with UML profiling mechanisms [179].

However, none of the current solutions for creating service-related knowledge consider their usage in service ecosystems but typically address only the engineering side of service-oriented computing. This academic work provides a unified framework which enables knowledge creation in open service ecosystems. The framework provides a methodology and tools for specification of service ecosystem architecture models that specify the features of a service ecosystem as a whole. The models facilitate knowledge creation and management in the eco-

system. The service ecosystem architecture models define ecosystem knowledge artefacts (e.g. service descriptions), infrastructure services (e.g. service discovery agents) and life cycles. Especially, the models declare explicitly how all these elements, which are typically addressed in the current state-of-the-art only individually, are related with each other; such holistic frameworks for service ecosystems do not exist yet.

Interpretation and utilization of knowledge artefacts is context-dependent in service ecosystems: for example service descriptions are utilized as implementation templates during the development phase, and as conformance criteria during operation. Especially in open service ecosystems it becomes important for knowledge integration purposes to make explicit the relationships between knowledge artefacts, ecosystem life cycles and their phases, and engineering methods utilized by service providers. Moreover, in open service ecosystems knowledge elements need to be mapped against varying domain-specific languages and engineering processes used in different organizations. For enabling collaborative service engineering between autonomous partners, knowledge integration activities are emphasized in the knowledge repository designs developed in this dissertation.

Representational formalization means putting information in a form such that computational mechanisms can access and interpret it [46]. Formalization of concepts related to service ecosystems has been addressed by research communities in service-oriented computing, model-driven engineering and service-oriented software engineering. However, the research efforts have typically taken a rather unilateral view on ecosystem knowledge and emphasized either knowledge as an engineering asset or knowledge as means for providing interoperable service delivery and collaboration.

On the contrary, the formalization given in this dissertation emphasizes equally the engineering and interoperability viewpoints over knowledge management in service ecosystems. These two roles of knowledge artefacts are unified by involving at the very core of the framework the principles of ontological and linguistic meta-modelling, similarly to [125]. Ontological meta-modelling addresses construction of ontologies whereas linguistic meta-modelling is about construction of so-called system models.

An ontology is a *descriptive model* used for characterizing the existing world, the environment and the domain of the system [5]. An ontology is inherently associated with with an open-world assumption: anything that is not explicitly stated remains unknown. Especially, two different systems (models) may satisfy an ontology if they differ in areas that are not explicitly mentioned in the ontology [5]. This is in contrast with the system models that completely specify a system within the limits of the corresponding point of view and the abstraction level used.

System model is a *prescriptive model* that is used to specify and control the

system under study [5]. A system model gives a specification of the system that must be conformed to by the corresponding implementations. Model-driven software engineering typically emphasizes the use of prescriptive models for facilitating development processes where abstract models of the system are first designed and then refined during the process to more concrete models and development artifacts. A modelling framework for open service ecosystems must provide mechanisms for construction of both ontologies and system models to cater for the dynamic knowledge landscape of open service ecosystems and the prescriptive nature of knowledge in engineering activities associated with service ecosystems.

Knowledge dissemination includes activities for making knowledge explicitly available for the knowledge workers to help their problem solving [46]. Knowledge dissemination mechanisms in service ecosystems have to support knowledge *a)* accessibility, *b)* conformance and *c)* coherency. For enabling accessibility of knowledge dissemination mechanisms in service ecosystems, global repositories that can be used by ecosystem members for knowledge publication and retrieval should be provided.

Mechanisms utilizable for knowledge dissemination in service ecosystems have been addressed especially in research related to semantic web services architectures [128, 165] and global model management frameworks [20, 21, 104, 24]. Knowledge repositories found in semantic web service architectures and global model management infrastructures are however quite generic, since they are designed for managing any kinds of ontologies and software models. In service ecosystems the knowledge repositories need to be more specific to enhance accessibility, consistency and coherency of knowledge.

From the accessibility perspective especially pragmatic support for shared engineering processes are missing from ontology repositories found in semantic web services architectures. Global model management repositories, such as the ModelBus infrastructure [20], support shared processes for knowledge creation and sharing [1]. However, the model management repositories currently concentrate on providing generic support for software development tool integration and dissemination of engineering knowledge between different tools [20, 60]. Consequently, providing a federated service ecosystem knowledge base becomes problematic, since mechanisms for maintaining inter-repository relationships induced by domain-specific coherency rules and support for repository interoperation in service ecosystem life cycles are not addressed.

1.4 Research questions

The goal for this academic dissertation is to enable a transition from contemporary *ad hoc* service ecosystems to sustainable service ecosystems required by modern business practices and the vision of open service ecosystems. A sustainable service ecosystem is a well-defined, governed system with an explicit architecture that addresses the fundamental concerns of ecosystem stakeholders, and provides means for maintaining viability of ecosystem operation. For supporting this transition, a service ecosystem engineering discipline is needed. Moreover, facilities for supporting service ecosystem sustainability are required. In addition, support for establishing open service ecosystems must be provided.

The above goal can be elaborated to more specific research questions. The fundamental research questions contemplated in this thesis are:

1. What are the essential concepts of service ecosystems?

2. How to facilitate rigorous service ecosystem engineering?

3. How to address the requirements stemming from sustainability and the vision of open service ecosystems in service ecosystem engineering life cycles?

To answer these questions first a conceptual model for service ecosystems is defined and formalized. Rigorous service ecosystem engineering is facilitated by a model-driven approach. The requirements stemming from the definition of service ecosystem sustainability and vision of open service ecosystems are addressed in different ways in the artefacts designed and developed in this thesis. Especially, the requirements are used as the criteria for evaluating the applicability and utility of the artefacts.

1.5 Research approach and contributions

This thesis proposes service ecosystem engineering as a novel systems engineering discipline. The discipline is realized with a model-driven approach. There are three main contributions in this thesis. First, the means for a rigorous service ecosystem engineering is provided. Service ecosystem engineering is defined as a systems engineering life cycle which comprises phases of ecosystem analysis, design, instrumentation and operation. Rigour for all these phases is provided with creation and utilization of explicit service ecosystem architecture models. Secondly, instruments for supporting service ecosystem sustainability are delivered. Service ecosystem sustainability is supported with a model-driven approach that

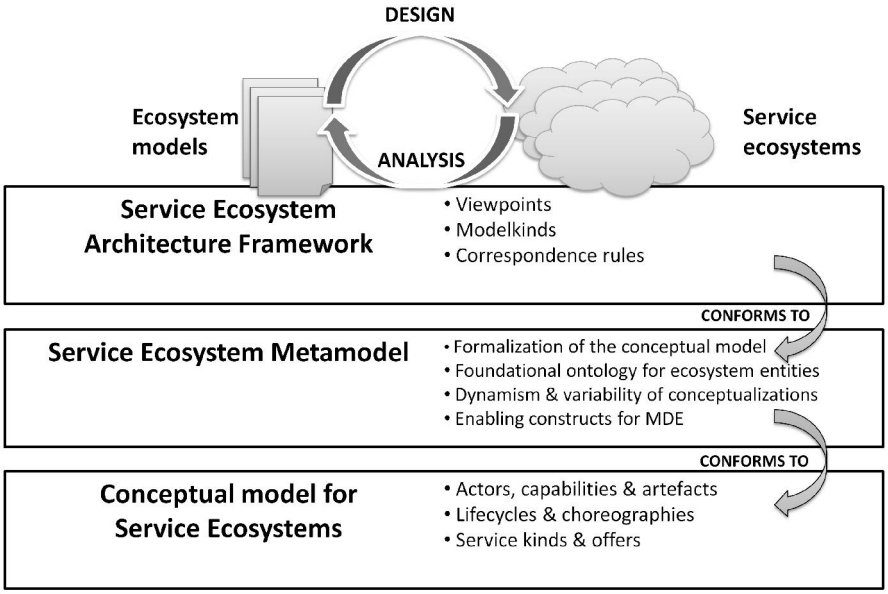

Figure 1.2: Relationships between the service ecosystem conceptual model, meta-model and the architecture framework.

addresses different stakeholder concerns in the different phases of service ecosystem engineering. Thirdly, means for managing interoperability and engineering knowledge in open service ecosystems are provided. These contributions in concert enable establishment of sustainable and open service ecosystems.

The approach taken for realizing rigorous service ecosystem engineering is model-driven. Means for analysing, designing and applying explicit service ecosystem architecture models are defined in this thesis. The approach is founded on three components: *1)* a conceptual model for open service ecosystems, *2)* a service ecosystem meta-model, and *3)* the Service Ecosystem Architecture Framework. These artefacts and their mutual dependencies are illustrated in Figure 1.2.

The approach is based on a holistic conceptual model which addresses several aspects of service ecosystems related to contracting, binding and service engineering, for example. Concepts defined in the conceptual model enable expression of service categories, service offers, service ecosystem behaviour, and actors, capabilities and artefacts, for example. Primary elements of the conceptual model are characterized in Chapter 2. The conceptual model is inspired by the Pilarcos framework [87, 85] whose concepts were used as a starting point for creating a more extensive vocabulary required for describing varying kinds of service ecosystems and their features. The applicability and completeness of the conceptual model is evaluated by a case study in Chapter 5.

The conceptual model for service ecosystems is elaborated and formalized by the service ecosystem meta-model defined in Chapter 3. The meta-model validates the feasibility of the conceptual model by providing means for service ecosystem architecture modelling. Also, the unified meta-model validates the consistency (e.g. contains no dangling references or orphan concepts) of the conceptual model that comprises various viewpoints on service ecosystems. Especially, the meta-model includes elements which enable application of model-driven engineering principles [144] over service ecosystem architecture descriptions. The meta-model also addresses ecosystem dynamism and evolution by defining foundations for expressing dynamism of ecosystem concepts, and specification of generalization and instantiation relationships. Moreover, the meta-model provides constructs enabling service ecosystem architecture specialization through domain-specific variations and extensions of the meta-model. Elements of the service ecosystem meta-model have been previously described for example in [135, 141]. The service ecosystem meta-model defined in Chapter 3 extends this work by providing refinements of the concepts and re-factoring of the meta-model to better support the needs of sustainable service ecosystems.

The third artefact developed in this thesis is the Service Ecosystem Architecture Framework (SEAF) which is defined in Chapter 4. SEAF is an architecture framework in the sense of ISO 42010 [72] which is based on the identification of service ecosystem stakeholders and their concerns. SEAF provides means for analysis and design of service ecosystem architectures. SEAF is implemented over a commercial UML-modelling tool. The applicability of SEAF is evaluated by the case study described in Chapter 5.

Finally, a model-driven methodology for service ecosystem engineering is defined in Chapter 4. The methodology utilizes architecture models produced with SEAF and model-driven engineering practices for engineering of domain-specific service ecosystems. The proof of concept methodology validates the feasibility of the model-driven approach for service ecosystem engineering, as well as serves as an example for efficient utilization of service ecosystem architecture models.

This academic dissertation does not explicitly address the operation phase of service ecosystem engineering life cycles. However, the the architecture models produced with SEAF can be utilized for model-based interoperability (see e.g. [13, 89]) and governance [143] of service ecosystem operation. The applicability of the architecture models for management and governance of service ecosystem operation is discussed in Chapter 5 and Chapter 6.

The research methodology utilized follows design science [62]. Design science is a research paradigm for artificial, man-made constructs where a set of artefacts are evaluated with respect to the utility they provide for stakeholders in a selected domain of interest. In this academic work, a set of artefacts are designed

which address the challenges of engineering sustainable service ecosystems and the vision of open service ecosystems. The utility of the artefacts is then evaluated with respect to the needs of service ecosystem stakeholders and requirements stemming from the definition of service ecosystem sustainability. The set of proof of concept artefacts comprise:

- a conceptual model for service ecosystems;
- a service ecosystem meta-model;
- SEAF; and
- a model-driven methodology for service ecosystem engineering.

All the preceding artefacts are assessed in Chapter 5 with respect to appropriate criteria, that is, conceptual completeness, feasibility, applicability and utility. Conceptual completeness of the conceptual model is evaluated using a case study where the Pilarcos service ecosystem is modelled with SEAF. The feasibility of the approach is validated by assessing the technical implementability of the individual artefacts. The applicability of the approach is validated with respect to the practical usability of the artefacts in service ecosystem engineering life cycles. Finally, the utility of the approach is evaluated with respect to the needs of service ecosystem stakeholders, and requirements stemming from the definition of service ecosystem sustainability defined in Chapter 2.

The impact of this work is two-fold. First of all, the results of this work facilitate rigorous engineering of sustainable service ecosystems. Enabling analysis and design of service ecosystems as holistic systems decreases the risks associated for example with technology dependency and migration. Explicit service ecosystem architectures enable efficient analysis techniques, e.g. assessment of service ecosystem maturity with respect to local criteria, to support business decision making when joining and operating in service ecosystems. As demonstrated by the model-driven methodology, service ecosystem architecture models are utilizable for instrumenting service ecosystems with necessary infrastructure services. Moreover, service ecosystem architecture models are utilizable for model-based interoperability approaches, as well as for supporting service ecosystem governance.

Secondly, added value and utility is provided for service ecosystem stakeholders. Ecosystem members are delivered the means for efficient utilization of core competencies, opportunistic and flexible business networking, supporting progressive business environments, and efficient business decision making. As demonstrated in the evaluation part of this thesis, these activities are addressed by the approach in service ecosystem engineering life cycles.

Added value and utility are provided for the different service ecosystem stakeholders. Initiating a service ecosystem becomes feasible when a vocabulary and a methodology for defining such abstract environments is provided. New service ecosystems can be initiated effectively to serve the purpose of a particular business domain, enterprise architecture, or social network, for example. Ecosystem designers working towards establishing new service ecosystems can utilize the meta-models for defining conceptual models and infrastructure services for service ecosystems.

Implementation and maintenance of the infrastructure services required for instrumenting a service ecosystem can be a business of its own, given an appropriate business model. Operation of such an infrastructure service provider can be motivated by an access to service usage patterns, or other ecosystem-wide business intelligence, that can breed new innovations or even be information of direct monetary value for some enterprises. The model-driven approach for service ecosystem engineering enables efficient construction of knowledge repositories for serving the purposes of the infrastructure service providers. Model transformations can be exploited for generating the required knowledge repository implementations. This academic work enables such efficient implementation approaches for infrastructure services.

Service ecosystem models are used by the ecosystem members for joining the ecosystem in an interoperable manner. The corresponding knowledge base is utilized by service consumers for finding appropriate services using the service discovery mechanisms. Dually, service providers use infrastructure services to feed the knowledge base with service offers that advertise the available services. The efficiency and dependability of service discovery are one of the most fundamental quality aspects of service ecosystems, as this function serves as an interface between service consumers and providers, as well as facilitates collaboration establishment processes. From this perspective, the service ecosystem models provide means for categorization of services; this is definitely needed in an open service ecosystem to enable efficient service delivery.

Service developers can utilize the service ecosystem models as a unification mechanism for integrating their development tools, domain-specific languages and engineering processes to serve the purposes of a specific ecosystem. Moreover, for enabling collaborative software engineering activities, engineering processes and engineering artefacts need to be integrated and shared among the participants of the domain; the meta-models and knowledge repository designs defined in this academic work are crafted for serving such purposes.

For managing interoperability knowledge in open service ecosystems this dissertation defines domain ontology model for service ecosystems. The domain ontology model defines concepts for description of service ecosystems. The domain ontology model can be specialized for the purposes of specific service eco-

systems. Especially, the domain ontology model suggests a two-level hierarchy of concepts divided into ontological types and instances. This division enables dynamic extension of the ontology during the operation of the ecosystem and moreover, declares a generic typing relationship between fundamental concepts existing in service ecosystems.

Regarding management of engineering knowledge, this dissertation provides a comprehensive definition of the elements, entities and features of service ecosystems. A unifying, holistic framework enabling management of engineering knowledge is provided. Relationships between functional and non-functional elements in service ecosystems can be made explicit in this framework, for example. Corresponding knowledge repositories can be utilized by developers for enabling global software engineering practices. For enabling collaborative software engineering activities, engineering processes and engineering artefacts need to be integrated and shared among the participants of the domain; the conceptual framework and its formalization presented in this dissertation are crafted for serving such purposes.

This academic dissertation induces several other research tracks and questions to be considered later. Service-oriented software engineering methodologies provide a whole other discipline which should be studied for providing a complete service ecosystem, for example. Such work involves development of software-engineering processes suitable for service-engineering and service-based system engineering. From a global software engineering point of view, the engineering processes should be provided with explicit models. The software engineering processes are envisioned to be composed of modelling workflows that utilize the knowledge repositories and different stake-holders involved in the engineering process. Consequently, the knowledge repositories must be provided with facilities to support such engineering workflows with transaction and notification support, for example. The meta-models used for defining service ecosystems are designed in such a way that these future research topics can be weaved into the the conceptual models easily. More over, the author envisions that the approach provided can be utilized for efficient production of ecosystem-specific engineering tools and a coherent families of domain-specific languages.

In service ecosystems the same knowledge repositories and models representing services, or other relevant artefacts, are needed equally by service-oriented software engineering tools and infrastructure facilities for service-oriented computing. To fully utilize the potential of this trend, complete tool-chains with consistent concepts and semantics for service-oriented software engineering are still missing. This dissertation provides a contribution towards this objective in form of a conceptual model and knowledge repository designs providing facilities for both design and run-time use of service ecosystem knowledge.

1.6 Related research areas

This academic dissertation contributes to several research areas. First of all, contributions towards the area of *service-oriented computing* [122, 151, 123] are given by elaborating and formalizing concepts of service ecosystems, and thus services and service-based collaborations in general. The contributions are concretized by the conceptualization and the corresponding meta-models. In addition, the knowledge repositories developed in this work are utilizable for service discovery and selection when realizing mature service-oriented architectures.

Secondly, this dissertation contributes to the research in *service-oriented software engineering* (SOSE) (see e.g. [156, 160, 97, 124]) by providing means for supporting engineering knowledge management in globalized environments and production of ecosystem-specific service development tools. Knowledge repositories enable engineering knowledge propagation in distributed software engineering processes, while models defining service ecosystems can be utilized for generating domain-specific languages and corresponding tools semi-automatically. SOSE is a software engineering approach which utilises constructs and concepts conforming with the service-oriented computing paradigm for designing, modelling and developing service-based systems. Corresponding methods used for production of service-oriented solutions utilize the infrastructure services of the ecosystem for enabling global software engineering [61] practices.

Finally, this work contributes in the research area of *model-driven engineering* (MDE) [144] by providing feasible methods for linguistic and ontological meta-modelling [8, 54] of models (ontologies) with both static and dynamically extensible concepts. The modelling method is based on deliberate use of meta-model extension [11] and partial instantiation of meta-models. Model-driven engineering is a software engineering discipline which considers models as first-class entities and primary objects of engineering [144]. The main challenges that the MDE approach pursues to respond to are induced by the complexity and evolution of computing platforms as well as complexity of system integration and configuration [144]. Domain-specific languages, model transformations, and code generation are utilised to bridge the semantic gap between problem domains and technology, and to efficiently produce software artefacts and systems that are "correct-by-construction" [144].

The approach presented in this academic dissertation shares similarities with *enterprise architectures* research. An enterprise architecture can be defined as *"a coherent whole of principles, methods, and models that are used in the design and realization of an enterprise's organizational structure, business processes, information systems, and infrastructure"* [88]. There are several enterprise architecture frameworks developed in the industry and academia, such as The Open Group Architecture Framework (TOGAF) [118], the US Department of Defense

Architecture Framework (DoDAF) [161], or ArchiMate [119].

Enterprise architectures are applied for managing the inherent structural complexity of enterprises, enabling enterprise engineering [65] and directing enterprise transformations [58]. In comparison, the approach developed in this thesis is directed towards enabling engineering and governance of sustainable service ecosystems. Both enterprise architectures and the service ecosystem architectures discussed in this thesis can be applied for enabling business decision making in organizations. Enterprise architectures do not typically address collaboration between autonomous entities, but concentrate on operations within a single enterprise; the approach developed in this thesis is founded on the idea of enabling such loosely coupled collaborations.

1.7 Structure of the dissertation

The remainder of this dissertation comprises five chapters. The conceptual model for service ecosystems and implications of service ecosystem sustainability are first elaborated in Chapter 2. The conceptual model describes the elementary elements of service ecosystems, as well as identifies stakeholders of open service ecosystems. Each stakeholder is associated with a set of concerns. These stakeholder concerns give the foundations for elaborating the notion of service ecosystem sustainability.

Chapter 3 introduces the meta-model for service ecosystems. The meta-model elaborates and formalizes the conceptual model. First in the chapter the primary design principles for the meta-model are identified. After that, the different constructs defined in the service ecosystem meta-model are discussed. The discussion does not detail all the elements included in the meta-model, since there are over 260 classes included in it. However, this chapter provides a sufficiently complete description of the meta-model elements and their characteristics such that the work can be evaluated.

Tools for model-driven service ecosystem engineering are defined in Chapter 4. The tools include an architecture framework for service ecosystems and a model-driven methodology for service ecosystem engineering. The architecture framework comprises modelling viewpoints and notations for service ecosystem architecture description. First in the chapter the elements of the architecture framework are described, and specifications of the viewpoints and corresponding notations are given. Secondly, a model-driven methodology for service ecosystem engineering is defined. The methodology utilizes model transformations that consume service ecosystem architecture models designed with the architecture framework. The model transformations produce refined architecture models, and technology-specific engineering artefacts required for enabling operation of the

service ecosystem.

Validation and evaluation methods and results are then discussed in Chapter 5. The validation approach and evaluation criteria are first defined. After that, the evaluation case is introduced. Evaluation results are then presented. The artefacts developed in this thesis are assessed with respect to conceptual completeness, feasibility, applicability and utility. The chapter is concluded with a comparison to related work.

Finally, conclusions about the research results and further prospects are discussed in Chapter 6.

1.8 Research history

This academic dissertation is a result of research stemming from the areas of inter-enterprise computing, interoperability management and service-oriented middleware platforms. The research has been conducted as part of the CINCO (Collaborative and Interoperable Computing) research group [84] which works towards enabling and facilitating service-based collaborative computing especially in inter-enterprise contexts.

The author's research was initiated for providing a so-called service typing discipline and corresponding type management infrastructure (type repositories) for the Pilarcos interoperability middleware [86, 85]. Interoperability challenges imminent in dynamic business-to-business environments were discussed in [86] which described the concepts and functionality of the Pilarcos B2B-middleware. Service types and type repositories were identified as foundational for the working of the federated service collaboration model described in the paper.

Service typing is used in the Pilarcos framework for achieving service interoperability. The author of this academic dissertation studied several formal methods for finding a feasible method for formalizing service interoperability. The research in formal methods resulted in construction of the service typing discipline [139, 134] which is based on session typing [162]. Session typing is a formal method that gives rigour to such concepts as business service behaviour, and behavioural compatibility and substitutability.

A brief description of the type management infrastructure needed in collaborative systems was given in [139] which followed the RM-ODP reference model for open distributed computing, consisting of type repositories and name registries. Information about service types and their properties were proposed to be propagated through a service-oriented software engineering process and standardised middleware interfaces to corresponding meta-information repositories [139].

In addition to interoperability between individual services, there are other aspects of interoperability that have been addressed for managing interoperability

in service ecosystems. Interoperability in service-based, inter-enterprise environments was represented with a layered model of interoperability management, consisting of five layers of increasing abstractness from technical level to business level [138]. The interoperability model provided can be regarded as a reflective meta-information system, i.e. changes in lower-layers of the interoperability meta-information "stack" are propagated to upper-layers using well-defined transformations. Moreover, the interplay between autonomy and interoperability in service-based environments were discussed [137], in addition to more generic characterizations of interoperability [142].

From the research efforts described above it became evident that, for achieving interoperability in service collaborations, one must provide a conceptual formalization of the service ecosystems as a whole. Achieving interoperability is a multi-faceted problem. For achieving service collaborations between autonomous partners each entity must have an unambiguous interpretation of the features affecting the form and quality of the collaboration. Moreover, pragmatic support is needed at the conceptual level for enabling production of new kinds of collaboration forms and services enabling their operation.

For achieving collaborative and interoperable computing in service ecosystems, the author developed a set of meta-models and ontologies describing the foundational elements of service ecosystems [140, 135]. Management of interoperability and engineering knowledge in service ecosystems was now taken as the focus of research. The conceptual model of federated service communities that had been developed within the Pilarcos framework [87, 86, 85] provided a basis for further research in the more generic context of service ecosystems.

The concepts of the Pilarcos framework were first generalized and additional concepts for enabling complete ecosystem description were developed. In addition, the role of service-oriented software engineering was emphasized while developing the service ecosystem concepts. The concepts were formalized using a set of meta-models and the practices of model-driven engineering [144] were taken as the primary means for instrumenting knowledge repositories [135]. Especially, the resulting conceptual framework for declaring features of service ecosystems was founded on the core principles of meta-modelling [45, 54] and on separation between linguistic and ontological meta-modelling [5, 125].

This academic dissertation is a continuation of the research process described above. The conceptual framework developed in the author's Licentiate thesis [135] has been further developed. In addition, a methodology for defining service ecosystem domain models has been developed and is described in this dissertation. Especially, the conceptual framework for service ecosystems is now validated within the context of this work.

Chapter 2

Sustainable service ecosystems

The conceptual model described in this chapter enables service ecosystem analysis by domain experts. Such an analysis tool is needed for example to construct a service ecosystem conceptualization from an ecosystem vision, as was illustrated in Figure 1.1. The conceptual model defines the fundamental elements of service ecosystems as well as dependencies between the elements. Thus means for unambiguous interpretation and communication of service ecosystem characteristics is provided.

The conceptual model addresses especially the requirements for establishing sustainability: it identifies the essential elements of service ecosystems, stakeholders and their primary concerns, as well as objectives, principles and implications for continued viability. The conceptual model and the definition of service ecosystem sustainability provide foundations and design principles for the other artefacts developed in this thesis. Moreover, the definition of service ecosystem sustainability and its implications declare evaluation criteria for assessing the applicability and utility of the model-driven approach for service ecosystem engineering. The completeness of the conceptual model is evaluated by a case study in Chapter 5.

The elements of service ecosystems are first elaborated in Section 2.1. Section 2.2 then introduces service ecosystem stakeholders and their concerns. After that, the notion of service ecosystem sustainability is defined in Section 2.3. Service ecosystem sustainability is defined as a quality of a service ecosystem to support continued viability; implications of this definition are discussed in the section.

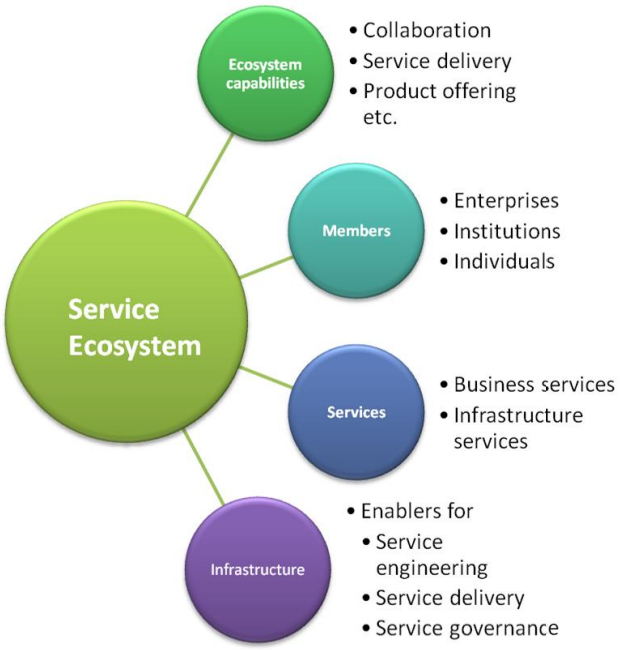

Figure 2.1: Elements of service ecosystems.

2.1 Elements of service ecosystems

Service-centric business takes place in varying kinds of service ecosystems. A service ecosystem is a socio-technical complex system that enables service-based collaboration between autonomous entities such as enterprises, institutions and individuals. The ecosystem provides an environment for creating and managing service-based collaboration networks, such as virtual organizations or service mash-ups, from services provided by a community of autonomous entities.

For handling the inherent complexity of service ecosystems, and for enabling analysis and description of service ecosystems, the fundamental elements and their inter-relationships must be identified and formalized. Based on a survey and an analysis of different service ecosystems and related frameworks [135], four fundamental elements existing in service ecosystems are identified. The ecosystem elements are : *1)* ecosystem capabilities, *2)* members, *3)* services, and *4)* infrastructure. The elements are illustrated in in Figure 2.1.

An *ecosystem capability* is the ability of a service ecosystem to perform actions. An ecosystem capability provides a declaration of core functionality in the service ecosystems. A set of ecosystem capabilities describes the motive for ecosystem existence, or the ecosystem purpose. Ecosystem capabilities include service delivery and collaboration, for example. In product-centric service ecosystems a capability of product provisioning is the primary motive for the ecosystem operation.

Ecosystem members are autonomous entities such as enterprises, institutions or individuals which act in the service ecosystem in specific roles. An *entity* is considered as an object of the ecosystem with an identity and that has its own distinct existence. Entities bound to a role must possess the capability of discharging the commitments a role imposes for them.

A *service* is considered here as a mechanism to access one or more capabilities [154]. A service is realized by an exchange of value objects between a service consumer and a service provider. Following the definitions provided in [9] and [76], a *value object* is created, traded, and consumed by ecosystem entities and is of economic, ideological, political or social value for at least one of the entities involved. From an economic perspective, services are considered as intangible, perishable, inseparable, and variable goods, as opposed to products (see e.g. [175]).

In the conceptual model for open service ecosystems, services are classified into three different categories [143], each category associated with a characterizing kinds of capabilities: *1)* infrastructure services delivering ecosystem capabilities, *2)* business services delivering business transactions, and *3)* component services delivering service protocols.

Infrastructure services are used for delivering ecosystem capabilities. For example, in the Pilarcos open service ecosystem [87] the set of infrastructure services include service offer repositories delivering loose coupling between service providers and consumers, as well as the populator service for dynamic business network establishment. Infrastructure services especially include the knowledge repositories needed for managing the global knowledge base of the service ecosystem. Additional infrastructure services, such as trust or reputation management systems, may be provided for facilitating collaboration establishment processes in service ecosystems.

A *business service* is a service which delivers the business transactions required to fulfill commitments of a business network role. Business services are technological representations of services (i.e. intangible goods) and are provided as software-based components that utilize information and communication technology for service delivery. A *business network* is a prescribed choreography between business roles which describes a business collaboration. A collaboration is a process of shared creation among a group of actors that share information, resources, responsibilities and rewards to achieve a common goal [27].

Business services are provided for clientele with service offers. A *service offer* is a declaration for a set of business services offered as a service bundle, and their capabilities and qualities. A service offer provides a basis for negotiating a service-level agreement between a service provider and a service consumer. Negotiation is *"a process by which a group of agents communicate with each other*

and try to come to a mutually acceptable agreement on some matter" [93]. After a successful negotiation process, the participants are provided with a shared *service-level agreement*, or SLA (see for example [96, 152, 153]), expressing the mutual commitments and expectations about the quality properties of the business services under contract.

Component services are utilized in business services to implement their functionality. Component services represent reusable service engineering assets which deliver functionality defined by service protocols. A service protocol provides a definition of component service operations and behaviour. Exemplary component services may include services for domain-specific business protocol exchanges, e.g. banking protocols, or more technically oriented services such as key exchange protocols for encryption.

Ecosystem infrastructure provides means for realizing service engineering, service delivery and service ecosystem governance activities in the service ecosystem. In addition to infrastructure services, ecosystem infrastructure includes *1)* service engineering tools and methods, *2)* specifications of ecosystem features and feature bindings, and *3)* cooperation facilities enabling dependable service interactions and communication. Service ecosystem governance is enabled with infrastructure services which coordinate cooperation between ecosystem members in accordance to ecosystem life cycles [143].

2.2 Service ecosystem stakeholders

Several stakeholders can be identified in service ecosystems. Typically at least service consumers and service providers are identified in contemporary service ecosystems. However, especially in open service ecosystems there are several additional stakeholders which have their distinctive roles. The roles may or may not overlap with service consumers and providers. The ecosystem stakeholders include service consumers, service providers, service engineers, engineering tool providers, infrastructure providers, and ecosystem providers, as illustrated in Figure 2.2.

Service consumers are legal entities that utilize business services available in the ecosystem to fulfill their business needs. Service consumers are responsible for utilizing the business services as declared in agreements made with service providers about service utilization. Service consumers may confront challenges related to identification and selection of suitable business services, dependability of business services, and monitoring of business service operations.

Service providers are legal entities that offer business services for their clientele. In a business landscape comprising diverse service ecosystems, service provider differentiation and revenue is created by providing added value for the cus-

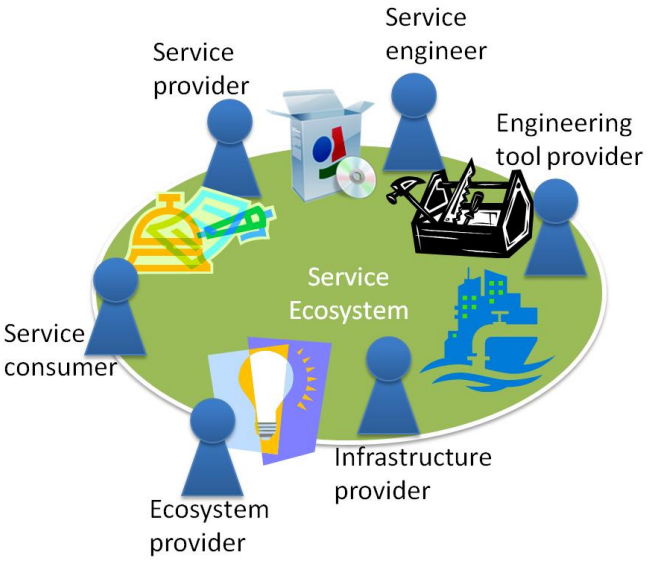

Figure 2.2: Stakeholders in service ecosystems.

tomers. Service providers act in service ecosystems for creating profit by optimizing their business service usage, and supporting their business models. The services consumed by the clientele may directly benefit the business of the service provider. Services can also be provided for creating support and added value for other products, or for establishing and promoting business platforms around the offered services [30]. Business services are provided within service bundles by publishing appropriate service offers.

Service engineers produce so-called component services by implementing technological artefacts (e.g. web services) which are then utilized by service providers to implement their business services. The primary concern of service engineers is that of well-advised service engineering. Well-advised service engineering here refers to both correspondence of service engineering with the business models ("well-advised motivations") and pragmatics ("well-advised practices") of the ecosystem. The notion of well-advised service engineering is elaborated in Section 2.3.1.

Engineering tool providers are responsible for providing service engineers with tools. For increasing the efficiency of service production in a service ecosystem, the tools should be specialized for the specific domain of operation. Consequently, the primary concern of engineering tool providers in service ecosystems is efficient tool production and specialization.

Infrastructure providers deliver infrastructure services required by service ecosystems. Delivering infrastructure services could be business on its own, or it might be affordable due to supporting other aspects of infrastructure provider

business models (e.g. bootstrapping the use of business services in the service ecosystem). Thus, efficiency of infrastructure service production becomes a concern for infrastructure providers.

Ecosystem providers represent entities such as organizations or consortia whose business models require utilization and instrumentation of a service ecosystem. Ecosystem providers initiate establishment of service ecosystems by first analysing and designing required ecosystem capabilities, and identifying means for delivering those capabilities. The primary concerns associated with ecosystem providers are analysis and design of service ecosystems, and enabling and maintaining ecosystem viability by service ecosystem governance.

2.3 Defining ecosystem sustainability

For supporting business in future service ecosystems, a transition from *ad hoc* solutions to sustainable service ecosystems is needed. Sustainability in essence enables feasible business and longevity for service ecosystem members. Feasible business is enabled especially by delivery of mechanisms that support efficient utilization of core competencies and business decision making. Longevity is enabled with means for opportunistic and flexible business networking, as well as by support for progressive business environments. These characteristics are elaborated further in Section 2.3.2.

In this academic dissertation, the definition of sustainability is based on terminology used in the context of software architecture research[1]. The definition for service ecosystem sustainability is given as Definition 2.1.

Definition 2.1 (Service ecosystem sustainability) *Service ecosystem sustainability is a quality of a service ecosystem to support continued viability.*

The implications of the above definition are discussed in the rest of this section. The requirements for service ecosystem viability are elaborated in Section 2.3.1 which identifies service ecosystem stakeholder concerns. After that, the means for establishing continuation of service ecosystem viability are defined in Section 2.3.2.

2.3.1 Requirements for service ecosystem viability

Viability is achieved by supporting the operation of stakeholders and addressing their primary concerns through the phases of the service ecosystem engineering life cycle. Service ecosystem engineering life cycle is a systems engineering life

[1]Sustainability in the context of software architectures is defined as “An architectural property of a program which allows continued viability” in the Open Knowledge Initiative by MIT.

cycle which comprises the phases of service ecosystem *1)* analysis, *2)* design, *3)* instrumentation, and *4)* operation.

Service ecosystem analysis is a systems engineering activity where a needs assessment is performed. Needs assessment is utilized for identifying the capabilities required from the service ecosystem to support the business objectives of associated stakeholders. The capabilities are formalized during the design phase. The service ecosystem design phase provides definitions for the necessary ecosystem infrastructure, life cycles, service ecosystem features, and domain concepts required for realizing the identified capabilities. During the service ecosystem instrumentation phase all the necessary engineering components required for service ecosystem operation are delivered. This includes especially implementations of the infrastructure services, and possibly selected component and business services to bootstrap the operation phase of the service ecosystem. During the operation phase service ecosystem stakeholders exploit the service ecosystem and its infrastructure to support and establish service-based collaborations.

Service ecosystem stakeholders have different concerns associated with service ecosystem engineering life cycles. The primary concerns are identified in Table 2.1 and discussed below.

Ecosystem provider concerns

For service ecosystem providers three primary concerns are identified in Table 2.1. In the service ecosystem analysis phase ecosystem providers need instruments which can support the identification of required service ecosystem capabilities, and formulation of ecosystem elements that support the delivery of those capabilities. Supporting this concern, a conceptual model for service ecosystems is provided by this thesis. The conceptual model is applicable for structuring the complexity of service ecosystems, identifying capabilities fulfilling business needs, and further constructing the required ecosystem elements.

A service ecosystem architecture for supporting delivery of the required ecosystem capabilities is constructed during the design phase. The architecture descriptions are utilized for communicating and further refinement of ecosystem designs between associated stakeholders. Moreover, different architecture analysis techniques, such as ATAM [77], are applicable for enacting the design process. To enable such architectural design activities, the architecture descriptions must be unambiguously defined. This requires formalization of the architecture descriptions. Formalization also provides means for validating the consistency and completeness of the ecosystem designs, as well as enables tool support for their further analysis. Towards these purpose, the conceptual model of service ecosystems is formalized as a service ecosystem meta-model in this thesis; the meta-model enables unambiguous description of service ecosystem architectures

Stakeholder	Engineering phase	Concern
Ecosystem provider	Analysis	Identification of required ecosystem capabilities and elements supporting the delivery of those.
	Design	Formalization of service ecosystem architecture descriptions.
	Operation	Supporting service ecosystem governance.
Infrastructure provider	Design	Identification and design of required infrastructure services.
	Instrumentation	Efficient production of infrastructure services.
	Operation	Infrastructure service interoperability.
Engineering tool provider	Design	Identification of engineering capabilities required for component service production.
	Instrumentation	Efficient tool production and specialization.
	Operation	Tool and language interoperability and integration.
Service engineer	Design	Identification of engineering capabilities required for component service production (shared with engineering tool providers).
	Operation	Well-advised service engineering.
		Efficient component service production.
Service provider	Operation	Service bundling.
		Service contracting.
Service consumer	Operation	Business service identification, location and selection.
		Business service dependability.
		Business service monitoring.

Table 2.1: Stakeholder concerns in service ecosystem engineering life cycles.

with models and modelling languages conforming with the meta-model.

During the operation phase, the primary concern of a service ecosystem provider is to enable and support service ecosystem governance. Service ecosystem governance is defined as a collaborative activity taking place between ecosystem members, which extends service governance (such as described in the Open Group SOA Governance Framework [117], for example) of the individual members in the ecosystem [143]. Service ecosystem governance comprises activities and structures for directing, monitoring and managing the operations enacted in a service ecosystem. For enabling and supporting such collaborative activities, service ecosystem providers define ecosystem life cycles. A service ecosystem life-cycle definition prescribes behaviour and roles expected from ecosystem members in the corresponding life-cycle phases.

The conceptual model, the service ecosystem meta-model, and the architecture framework defined in this thesis provide instruments for prescribing service

ecosystem life cycles and structures. Moreover, the model-driven approach used for designing and instrumenting service ecosystems supports governance activities by delivering a unified service ecosystem model. These service ecosystem models can be used during the operation phase for coordinating the behaviour of participants in ecosystem life cycles, and for realizing model-driven interoperability (see e.g. [19]) approaches.

Infrastructure provider concerns

Infrastructure providers take part in the service ecosystem design phase especially to analyse and design the kinds of infrastructure services required for realizing the ecosystem capabilities. Infrastructure providers require concepts and modelling facilities for expressing the features of infrastructure services and the requirements they induce to other parts of the service ecosystem architecture.

For addressing infrastructure provider concerns, this thesis provides concepts and an architecture framework for designing infrastructure services and formalizing their designs as part of architecture models. During the instrumentation phase, the architecture models can be exploited for enabling efficient production of infrastructure services. Finally, interoperability between infrastructure services must be supported during ecosystem operation. Formal architecture models defining the features, structure and inter-dependencies between infrastructure services can be utilized for achieving interoperability. Especially model-driven approaches become applicable for establishing both conceptual and technological unification.

Engineering tool provider concerns

Engineering tool providers are involved during the service ecosystem design phase by identifying and designing required engineering capabilities. Engineering capabilities comprise tools and methods needed for producing engineering artefacts in the service ecosystem, especially for component service production. During service ecosystem instrumentation, efficiency of engineering tool production and specialization becomes a concern for engineering tool providers. The engineering tools produced are utilized by service engineers during ecosystem operation for developing component services. The efficiency of component service production can be increased, if the tools themselves address domain-specific features and use domain-specific concepts. Thus, efficient specialization of engineering tools becomes a concern for engineering tool providers. Finally, during service ecosystem operation the interoperability between engineering tools and languages, and their integration may become an issue for engineering tool providers.

For addressing the engineering tool provider concerns, this thesis provides means for designing the required engineering capabilities, instrumenting efficient

modelling tool production with model-driven methods, and integrating domain-specific languages through the unified service ecosystem models.

Service engineer concerns

Service engineers are primarily engaged by concerns associated with service ecosystem operation phase. However, service engineers also have their role during the design phase when identifying engineering capabilities for the ecosystem together with engineering tool providers. During the operation phase, service engineers are confronted with two foundational challenges: *1)* establishing well-advised service engineering, and *2)* increasing the efficiency of component service production.

Well-advised service engineering here refers to both correspondence of service engineering with the business models ("well-advised motivations") and pragmatics ("well-advised practices") of the ecosystem. Well-advised motivations means that service engineering activities should align with the requirements stemming from the business models and business networks applied in the service ecosystem. Alignment can be advanced with service engineering methods and tools, which utilize domain-specific concepts for bridging the semantic gap between business and engineering domains. Well-advised practices means that service engineers should be provided with re-usable engineering artefacts that allow encoding and sharing of best practices and regulations associated with the service engineering domain. Well-advised practises increase the efficiency of service production, thus increasing the viability of service engineers.

From service engineers' perspective, added value for customers is delivered by crafting services to fit the requirements of the clientele and the domain of the service ecosystem. In this setting, efficiency and flexibility of service production becomes a critical asset for service providers. Towards this purpose, the tools and facilities utilized by service engineers must support rapid reaction to market changes, emergence of new service ecosystems, and adaptation and reuse of services in different service ecosystems. This can be achieved with engineering facilities that are targeted for specific ecosystem domains.

Service engineer concerns are addressed in this thesis by providing means for formalizing and modelling engineering capabilities (tools and methods) as part of ecosystem architecture descriptions. Well-advised service engineering motivations are instrumented with explicit models of business network models, and formalization of relationships between component services, business services and business networks.

Service provider concerns

Service providers are responsible for delivering the business services as declared in corresponding service-level agreements. Moreover, the service providers are expected to act in accordance with the rules of the service ecosystem. Service providers may encounter several challenges related to provisioning and delivery of services. These challenges include service bundling, contracting issues, business network management, and service governance. In addition to the above challenges, service innovation is a fundamental challenge in service-based business environments. Service innovation is a complex topic which may include economical, organizational and social challenges (see e.g. [30]). However, service innovation is not in the scope of this dissertation.

Service bundling is used by service providers for supplying a selection of services as a single service for consumers. The services used in a bundle can be provided by other service providers (i.e. "sub-contractors"). Reasons for service bundling include cost reduction through utilization of common processes and shared infrastructure, inter-dependency of bundled services (e.g. bundling a set of financial services), product differentiation, increasing revenues, and increasing competitiveness through creation of entry barriers [9]. In service ecosystems service bundles are advertised through service offers. Service bundling can become an issue if the ecosystem does not allow declaration of provider-specific bundles (i.e. the ecosystem does not allow autonomous decisions over bundling), or if the service offering mechanism utilized in the ecosystem does not support service bundling (i.e. issues in expressing bundling).

Contracting is needed in service ecosystems for guaranteeing dependable service delivery between autonomous service providers and consumers. With respect to contracting, service providers are encountered with challenges related to contract formulation and establishment. A service ecosystem should provide mechanisms that can be utilized for managing these challenges.

With respect to contract formulation, a service ecosystem should provide concepts that can be used by service providers to express varying kinds of service contracting relationships. In different kinds of business ecosystems different kinds of business models, business networks and relationships are utilized. In product-driven business ecosystems there is typically a dominating partner, the product owner. In a corresponding service ecosystem, the business networks could be formulated in a way that in every business network established at least some of the services must be provided by the product owner, for example. In more traditional business ecosystems sub-contracting relationships between ecosystem members constrain and direct the formulation of available business networks. Moreover, different in- and out-sourcing scenarios might be dominating factors in business networks allowable in certain service ecosystems. The conceptual framework of

service ecosystems should enable expression of such varying kinds of service contracting and business networking scenarios.

Service consumer concerns

In open service ecosystems service consumers and providers are free to leave and join the ecosystem as they wish. Moreover, service providers decide autonomously about the features and accessibility of their business services. In such an environment, service consumers require means for identifying and locating services they wish to use and selecting the most appropriate ones for their needs. For this purpose, the infrastructure designed for an open service ecosystem must provide mechanisms for efficient service discovery and selection.

Open service ecosystems must provide means for establishing a suitable level of dependability in business service interactions and communication. Service consumers and providers should have the possibility to dynamically select, agree and realize features of business service interactions and communication, based on the operational context and business sensitivity, for example. This requires concepts and instruments for unambiguously describing the nature of service ecosystem features and their effects on service channels used for enabling business service collaboration.

Especially in open service ecosystems where service contracts are established dynamically between autonomous ecosystem members, service monitoring is needed for identifying and reacting to possible contract breaches. Service monitoring includes observation of service interactions (behavioural monitoring), service level (quality monitoring), and service compliance. For enabling behavioural monitoring, the behaviour of services associated with a service contract should be unambiguously described. Service level monitoring requires well-defined descriptions about the mutually agreed service quality values. Finally, service compliance monitoring involves both observation of contractual and regulatory compliance. Contractual compliance here refers to conformance of partners with the properties of the business network, such as sub-contracting relationships, or in- and outsourcing contracts. Regulatory compliance means conformance with legislation, business rules and policy frameworks associated with the business ecosystem and partners as legal entities.

2.3.2 Establishing continuation of viability

Continuation of service ecosystem viability is established by supporting activities that ensure relevance and sensibility of ecosystem operation. Towards this purpose, service ecosystems should provide means for *a)* efficient utilization of core competencies, *b)* opportunistic and flexible business networking, *c)* support-

ing progressive business environments, and *d)* efficient business decision making. These objectives are realized by principles for supporting continuation of viability. The objectives, their realizing principles and implications of following the principles are described in Table 2.2 and discussed below.

Supporting efficient utilization of core competencies

For supporting efficient utilization of core competencies a service ecosystem must allow autonomy of its members. *Autonomy manifests itself as degrees of freedom given to ecosystem members* with respect to design and implementation of services, and decisions concerning entities' willingness to collaborate and local service governance [137]. By allowing autonomy, the ecosystem members can maintain their competitiveness and agility of their core business operations more efficiently. For example, service engineers want to utilize production methods and tools that best suit their engineering practices; service providers want to exploit their business networks and added-value services for increasing the efficiency of service provisioning and provider differentiation.

A viable service ecosystem must not restrict the facilities or business networks stakeholders use for their operations locally, since the well-being (competitiveness, efficiency etc.) of the ecosystem members promotes the viability of the ecosystem itself. Instead, the service ecosystem should provide facilities for the ecosystem members for joining and operating in the ecosystem effortlessly. Such facilities include for example mappings between representations used in the service ecosystems and different technological spaces (e.g. Web Services -architecture [169] or REST-based web application frameworks), explicit specifications of engineering tools and practices utilizable for creating service artefacts in the service ecosystem, and unambiguous definitions for ecosystem vocabulary enabling both conceptual and technological mappings between heterogeneous domains.

Autonomy of ecosystem members induces potential interoperability problems. For example, the freedom of design and implementation of services leads to technological and semantic heterogeneity which induces interoperability problems. *Interoperability* means the capability of systems (e.g. organizations, business services, communication technology) to co-operate in such a fashion that eventually either their mutual goals become fulfilled or their co-operation is dissolved in a controllable manner in case of problems, such as conflict of interests or technical failures [135].

Interoperability can be considered from technical, semantic and pragmatic viewpoints [86]. *Technical interoperability* means that the technological facilities underlying the business services are compatible, such that communication paths can be established, for example. *Semantic interoperability* deals with the meaning

Objective	Principle	Implication
Efficient utilization of core competencies	Ecosystem member autonomy	Members can follow their established practices to operate in the ecosystem.
		Facilities are provided for supporting effortless integration and compatibility of local practices with ecosystem capabilities and policies.
		Interoperability service utilities are provided for identifying and handling interoperability problems between autonomously provided services, and features and policies associated with them.
	Dependability of service collaborations	Ecosystem provides means for assessing, establishing and monitoring dependability of service collaborations. In open service ecosystems this involves especially the aspects of trust, privacy and security.
Opportunistic and flexible business networking	Dynamic business service collaborations	Ecosystem provides means for establishing business service collaborations on demand.
		Mechanisms are provided for loose coupling of business services; this includes especially late binding and late encapsulation of services.
Supporting progressive business environments	Open model of ecosystem participation.	Autonomic entities may join and leave ecosystems as they wish. Ecosystem members may take any role available in the service ecosystem, in accordance with ecosystem policies.
	Extendability of ecosystem knowledge bases.	New kinds of services, cooperation facilities and business networks can be introduced on demand.
Efficient business decision making	Predictability of joining and operating in service ecosystems.	(Potential) ecosystem members are provided with methods for analyzing the maturity and feasibility of service ecosystems, and their capabilities and requirements with respect to local enterprise and service architectures.
	Governability of service ecosystem operation.	Local governability: Ecosystem members are able to govern their local service-oriented architectures in accordance with ecosystem requirements and capabilities.
		Global governability: Ecosystem members are provided with means for collaborative governance of service ecosystem operation.

Table 2.2: Objectives, principles and implications for establishing continued viability.

of exchanged information and information exchange patterns. *Pragmatic interoperability* is achieved if the intentions, business rules, and organizational policies of collaborating parties are compatible with each other.

Autonomy allowed for service ecosystem members must be compensated with mechanisms for guaranteeing interoperability. Three approaches can be identified for achieving interoperability, namely integration, unification and federation [86]. Integration aligns with the traditional model of software system development where interoperability is ensured by pre-development and pre-operational agreements about the properties of collaboration components, and basically handcrafting the corresponding software artefacts to fulfill the prerequisites for interoperability. In unification a shared model describes the functionality and responsibilities of each collaboration participant and provides the knowledge needed for attaining interoperability. In the federated approach interoperability is achieved by utilizing shared interoperability knowledge and Interoperability Service Utilities (ISU, see for example [40]. Interoperability knowledge comprises different kinds of artefacts, such as descriptions of services and their features, domain ontologies and communication channel descriptions, for example. Interoperability Service Utilities denotes a category of infrastructure services which provides capabilities for identifying and overcoming interoperability problems. Operation of interoperability service utilities is typically based on unified models representing service ecosystem architecture and vocabulary, for example.

Supporting opportunistic and flexible business networking

Supporting opportunistic and flexible business networking required means for establishing business service collaborations on demand, and mechanisms for loose coupling of business services. For enabling on-demand business service collaborations, dynamic business network establishment and negotiation processes [87, 85], as well as facilities for contract establishment and enforcement are required.

In a dynamic business network establishment process a business network model is filled with appropriate business services. For this purpose, specialized infrastructure services, such as the Pilarcos populator [85], are utilized. In addition, infrastructure services addressing trust [131] and privacy management [149] issues may be needed for establishing dependability in dynamic business network establishing processes.

Late binding, also known as dynamic or run-time binding, of service interfaces is one of the principal tenets of service-oriented computing. In a collaboration involving late binding between business services a *binding process* is initiated for deciding the features, especially the identity and location of service endpoints, of forthcoming service interactions. Essentially, the binding process establishes a contractual context (a binding) between service interfaces to enable service inter-

actions [69]. The binding process may comprise refinements of communication channel models and negotiations about the features of the communication channels.

In late encapsulation the service channels constructed with binding processes are further refined with qualitative features. Late encapsulation involves selection and negotiation about the features such as security, privacy or non-repudiation, for example. After a successful negotiation process, the participants are provided with a shared service-level agreement, or SLA (see for example [96, 152, 153]), expressing the mutual commitments and expectations about the qualitative features of service interactions.

Facilities for contract establishment include repositories for storing contract templates and contracts, and mechanisms for contract negotiation and validity checking [102], for example. Contract templates provide standard contract forms to facilitate the drafting of collaboration contracts. Repositories storing contracts are needed for keeping signed copies of contracts as evidence for possible dispute settlement processes.

Contract enforcement comprises infrastructure services that provide means for monitoring the compliance between service interactions and a collaboration contract, contract enforcement when a collaboration partner deviates from the behaviour prescribed by the contract, and dissolution of collaborations. The essentials for enabling contract enforcement and dispute settlement in electronic collaborations are typically borrowed from conventional (non-electronic) environments: compensations, insurances, fines and trusted third parties acting as notaries are used for giving the motivation for the partners to follow the collaboration contract. Contract notification, mediation and arbitration mechanism are examples of typical contract enforcement facilities [102] used for informing the parties about (possible) contract deviations, and settlement and resolution of actual contract breaches.

Supporting progressive business environments

Progressive business environments require support for ecosystem evolution. Ecosystem evolution means dynamism and flexibility over the selection of actors available in the ecosystem, and extendability of the ecosystem knowledge especially with respect to available services and collaboration types. New members may join and members may leave the ecosystem during its operation. Moreover, new kinds of roles for ecosystem entities can be introduced for supporting newly emerged business opportunities. For supporting ecosystem evolution, the conceptual model underlying the ecosystem must be extendable such that new concepts can be introduced dynamically and on demand. Support for ecosystem evolution enhances the elasticity and longevity of the service ecosystem.

From the knowledge management perspective, especially the ontology of services must be dynamic and extendable in an open service ecosystem. In such a setting new services can be introduced to the ecosystem and removed from it on demand. Moreover, service categories, that is, the types of services that can be offered and utilized, must be extendable to promote introduction of new kinds of services. Extendability of the ontologies must also be enabled for other kinds of artefacts, such as communication channel definitions or non-functional feature models.

Supporting efficient business decision making

Supporting efficient business decision making requires predictability of joining and operating in service ecosystems, as well as governability of service ecosystem operation. Potential ecosystem members must be provided with support for efficient business computing, and for identifying business opportunities and value proposition in the ecosystems. Such support can be delivered for example with service ecosystem modelling and analysis tools, which enable assessment of capabilities and maturity of service ecosystems. Moreover, tools for analysing the compatibility between local enterprise architecture and business models with the service ecosystem architecture and its business networks would support making business decisions about joining and operating in service ecosystems.

Efficient business decision making requires local governance of enterprise systems and global governance of service ecosystems. Governance can be characterized as a *"process of making correct and appropriate decisions on behalf of the stakeholders of those decisions or choices"* [98]. Local governance activities in a service-oriented organization typically involve utilization of so-called enterprise architectures describing the structure of the organization and its operation, and service portfolios declaring the services used and provided by the organization. Frameworks such as the Open Group SOA Governance Framework [117] can be utilized by organizations as a basis for implementing service governance. For enabling efficient business decision making in organizations, the contemporary governance frameworks should be extended with concepts and methods which enable alignment of local enterprise governance with service ecosystem requirements, capabilities and service ecosystem governance structures.

Service ecosystem governance means the collaborative activity of directing, monitoring and managing service ecosystem operation [143]. Governance activities are enacted by ecosystem members in life cycles which define how, when and by whom a certain activity should be taken. These activities are defined in service ecosystem life cycle declarations as part of the service ecosystem architectures.

Chapter 3

A meta-model for service ecosystems

The meta-model defined in this chapter enables construction of service ecosystem architecture description models. The architecture description models are utilized by different stakeholders during service ecosystem engineering, as was illustrated in Figure 1.1. Moreover, the meta-model enables construction of the model-driven methodology for service ecosystem engineering that will be developed in Chapter 4. The meta-model supports especially service ecosystem design and instrumentation activities: it acts as a foundation for the engineering tools required for the engineering activities included in the methodology.

The meta-model elaborates and formalizes the conceptual model and enables instrumentation of service ecosystem sustainability. Especially, the meta-model provides support for continuation of viability and enables model-based management and governance of service ecosystem operation. Such management and governance facilities are required especially in open service ecosystems for guaranteeing correctness of ecosystem operation and establishing interoperability between autonomous members. Moreover, formal ecosystem architecture models support unification and integration of separate (enterprise) systems and even different service ecosystems. This decreases especially the risk of building conceptual and technological stove-pipe systems, since the designs of service ecosystems are explicit and, as such, mappable to other formal system designs (other service ecosystem designs). The applicability of the meta-model for facilitating service ecosystem sustainability is assessed in Chapter 5.

The meta-model design is affected by the requirements and principles stemming from the definition of service ecosystem sustainability and stakeholder concerns. These design principles are discussed in Section 3.1. Before presentation of the service ecosystem meta-model, the modelling practices and conventions used for designing and illustrating the meta-model are described in Section 3.2. The meta-model is founded on a few core concepts that are defined and formalized in Section 3.3. The meta-model elements for service ecosystem architecture

descriptions are defined in Section 3.4. The architecture descriptions include specifications for service ecosystem structure, domain ontologies, intensional definitions, engineering models, and knowledge management models. The corresponding meta-model definitions are presented in Sections 3.5 – 3.9.

3.1 Design principles for the meta-model

There are two foundational requirements that affect the design of the service ecosystem meta-model. First of all, the conceptual model for service ecosystems must be formalized by the meta-model for establishing consistency of service ecosystem architecture descriptions, as well as for enabling rigorous service ecosystem design and analysis. Secondly, design principles for the meta-model need to reflect the requirements implied by service ecosystem sustainability and the vision of open service ecosystems.

The main elements of the conceptual model for service ecosystems were described in Chapter 2. The conceptual model declared service ecosystems as complex systems comprising capabilities, members, services and infrastructure. These service ecosystem elements are elaborated and formalized in the meta-model as a domain ontology model for service ecosystems. The domain ontology model formalizes ecosystem capabilities and relates them with life cycles and infrastructure services. Ecosystem members are represented in the domain ontology model by concepts of actors, roles and legal entities. The domain ontology model formalizes also the service categorization of the conceptual model. Elements of the service ecosystem infrastructure (e.g. infrastructure services and service engineering capabilities) are provided with formal definitions in the meta-model. Moreover, additional constructs are specified in the meta-model when needed for establishing consistent representations and relationships between the service ecosystem concepts.

For establishing viability of service ecosystems, the meta-model must take into account the stakeholders described in Section 2.2 and their concerns identified in Section 2.3.1. Towards this end, the service ecosystem meta-model includes constructs for *1)* enabling efficient engineering techniques and practices, *2)* supporting diverse business and service delivery models, and *3)* a rich feature management model supporting declaration, management and evolution of abstract service platforms.

Efficient service engineering techniques and practices are needed for addressing the concerns and supporting the activities of service engineers, infrastructure providers, and engineering tool providers. For enabling efficient engineering techniques and practices in service ecosystems, the meta-model includes principal concepts underlying model-driven engineering [144] as well as constructs

supporting situational method engineering [103]. The principal MDE concepts included in the meta-model formalize the notions of models and their dependencies [44, 17], enable multi-level meta-modelling [7], as well as provide means for unification of linguistic and ontological modelling practices [8, 54]. Consequently, the service ecosystem meta-model subsumes a *mega-model* [18, 16] (i.e. a model that represents or refers to models) for service-oriented software engineering.

Linguistic meta-modelling is used for defining modelling languages and their primitives on the meta-model level [8, 54] and so-called *linguistic instantiation* is used for instantiating model elements from the types defined within a corresponding meta-model. That is, linguistic instantiation crosses modelling levels and forms the basis for linguistic meta-levels [8] (e.g. levels comprised of meta-meta-models, meta-models and models).

Domain concepts are designed using ontological meta-modelling where so-called *ontological instantiation* is used for creating domain specific artifacts using the concepts defined at the upper-level ontology, or an ontology meta-model [8, 54]. Ontological instantiation takes place within a linguistic modelling level [8]. Ontological instantiation provides support for facilitating dynamic user extensions to modelling concepts, modelling notation and the models created from them [8]. This is especially invaluable for facilitating dynamism of knowledge in open service ecosystems.

The principles of linguistic and ontological meta-modelling are illustrated in Figure 3.1. In the figure, so-called linguistic instantiation takes place between the elements residing at linguistic modelling levels L1 and L2 that are separated vertically: *Breed* is a linguistic instance of *MetaClass*, and *Collie* is a linguistic instance of *Class*, for example. In Figure 3.1 the linguistic modelling level L1 includes three ontological modelling levels O2, O1 and O0. So-called linguistic instantiation takes places between the concepts of *Breed*, *Collie* and *Fido* correspondingly. It should be noted that ontological instantiation is not a transitive relationships: while in the example illustrated in Figure 3.1 *Fido is-a Collie is-a Breed* does hold, it does not make sense to say that *Fido is-a Breed*.

There is an essential difference between linguistic and ontological instantiation, in addition to the fact that the former is an inter-modelling level relationships while the latter is an intra-modelling level relationship. This difference is about the *intensional* and *extensional* meaning [82, 54] of concepts defined in the models. While in the example case illustrated in Figure 3.1 *Fido* is conformant with the characteristics of a *Collie*, as specified in the intensional part of the *Collie* concept, it also belongs to the set of all *Collie*s. That is, in ontological meta-modelling the instantiation relationship happens between two concepts if and only if one concept is conformant to the intensional part of the other and is *an element of* the

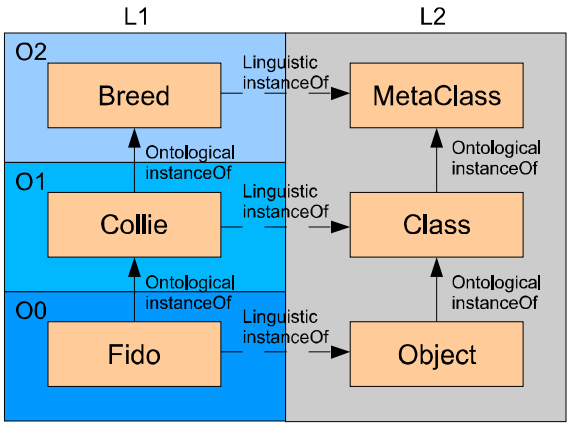

Figure 3.1: An example of ontological and linguistic instantiation relationships [8]

extension of the other concept [54].

Situational method engineering is a software engineering discipline, which focuses on project-specific method construction [127]. In the context of service ecosystems, situational method engineering can be utilized for sharing and re-use of engineering knowledge, and for establishing distributed software engineering practices. For supporting situational method engineering, the service ecosystem meta-model includes constructs for specification of re-usable engineering knowledge, where methods are composed of autonomous method chunks [103].

Support for diverse business and service delivery models is needed for addressing especially the concerns of service providers and service ecosystem providers. This support is realized in the service ecosystem meta-model by several constructs. The meta-model includes constructs for technology agnostic and flexible specification of service composition and bundling. The concept of business services in the meta-model corresponds to composite services delivered for fulfilling a service role. The meta-model provides means for exploiting possible sub-contracting relationships, supply chains, or other forms of business networks in business services. The concepts enabling such rich expression of business networks are based on the notion of commitment operations [150] and loose coupling between service roles, business services and their features. The meta-model includes constructs for specification of service bundles. The constructs representing service bundles enable expression of additional functionality and features for delivering added value and differentiation in service markets. Finally, constructs for representing product models are included in the service ecosystem meta-model.

Ecosystem providers, infrastructure providers as well as service engineers need to be able to evolve and manage the service ecosystem infrastructure to better respond to ever-changing business and customer needs. The service ecosys-

tem meta-model includes a rich feature management model for managing features of service ecosystems and their infrastructures. The feature management model comprises feature categories and binding models. The feature categorization enables control over description and management of service ecosystem features and their inter-dependencies. The binding models allow definition and control over feature usage. The dynamism of features, and the binding targets (e.g. service channels) and features bindable to them can be specified with the constructs provided by the meta-model. This allows for explicit declarations about such characteristics as late binding or late encapsulation in service ecosystems.

The service ecosystem meta-model provides the foundations for supporting continuation of viability during service ecosystem operation, as well as for enabling open service ecosystems. The foundations are delivered by addressing *a)* semantic interoperability, *b)* dynamism required for opportunistic and flexible business networking, *c)* evolution capability supporting progressive business environments, and *d)* ecosystem architecture specialization explicitly in the meta-model definition.

Semantic interoperability is addressed in the service ecosystem meta-model by the domain ontology model. The domain ontology model serves as a means for enabling interoperability through unification (during service ecosystem operation) of concepts. The meta-model enables control over service ecosystem dynamism by demarcation of static and dynamic parts of ecosystem concepts. Ecosystem evolution and specialization is addressed with modelling structures that support specification of generalization and instantiation hierarchies, as well as extension and management of variability of ecosystem concepts and features.

In general, models can be considered either as prescriptive systems models or descriptive domain models [5]. A system model is a description or specification of a system where a domain model describes its environment. The service ecosystem meta-model plays both of these roles. The domain model of service ecosystems is defined by the domain ontology model described in Section 3.6 (used for defining domain concepts and their relationships) and the intension model (used for defining the features that are used for defining the intensions of the concepts) defined in Section 3.7. Other parts of the meta-model define the characteristics of the service ecosystem as a system model.

There is an important implication stemming from the distinction of the system model and the domain model roles of the service ecosystem meta-model that should be noted: ontological instantiation (distinction between types and their instances) takes place only within the domain model part and is addressed by the concepts residing in the domain ontology model and the intension model. For example the model element named *Event* is defined at the system model part of the ecosystem meta-model; it is thus not subject to ontological instantiation and

should not be considered as an ontological type or instance. On the contrary, the model element named *LegalEntityKind* is defined at the domain model part of the ecosystem (more specifically, within the domain ontology model) and plays the role of an ontological type for legal entities. The ontological instances of legal entities are represented with the concept of *LegalEntity* defined within the domain ontology model.

3.2 Modelling practices and conventions

In the following sections the meta-model for service ecosystems is described. There are several options for modelling of conceptual frameworks, ranging from mind-maps [1] to formal ontology definition languages, such as OWL DL [2]. The various modelling approaches differ from each other for example on the basis of their notation, semantics and applicability.

The modelling approach selected for this thesis is based on the Meta Object Facility (MOF) [112]. MOF is an industry-standard developed by the Object Management Group (OMG) [3]. It is is especially applicable for definition, development and management of modelling languages and models created by those languages. In essence, the MOF provides a definition for a so-called meta-meta-model, that is, a meta-model for declaration of meta-models. The foundational elements of MOF include for example definitions for such concepts as *"Class"*, *"Property"* or *"Package"* applicable for description of classes, their features and collections of classes, correspondingly. MOF is utilized for example by the UML-standard [113] for definition of the UML abstract syntax (i.e. the UML meta-model).

The meta-model for service ecosystems has been designed with a commercial UML-modelling tool. UML class diagram notation [111] was utilized for designing and illustrating the meta-model elements. The UML-model was then imported to the Eclipse framework [4] for enabling development of the tools required by the model-driven service ecosystem engineering methodology. During the import the UML-model was converted to Ecore model, which is the meta-meta-model of the Eclipse Modeling Framework [39]. The Ecore meta-meta-model is closely aligned with the OMG MOF.

The service ecosystem meta-model is a single, unified model where every model element is connected to at least one other element of the model by generalization or association. The model exists both as an UML model and an Ecore model [136]. As the meta-model comprises currently over 260 classes it is pre-

[1] http://en.wikipedia.org/wiki/Mind_map
[2] http://www.w3.org/TR/owl-guide/
[3] http://omg.org
[4] http://eclipse.org

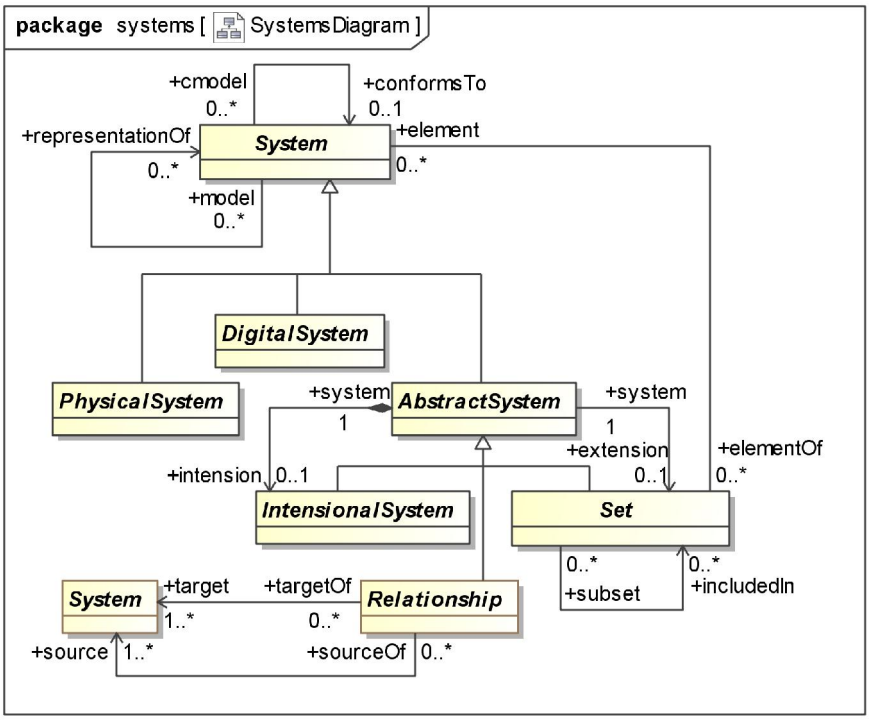

Figure 3.2: Modelling conventions: associations, generalizations and uniqueness of naming.

sented piece by piece in the following sections. Not every class of the meta-model is presented in this thesis for clarity of presentation. Especially, some generalization hierarchies are not presented but only the most generic classes are discussed. Some of the meta-model elements not discussed thoroughly in this thesis have been elaborated in previous publications [135, 141, 140]. However, the presentation in this thesis describes the most fundamental elements of the meta-model such that the feasibility, applicability and utility of the model-driven approach for service ecosystem engineering can be evaluated.

The modelling conventions followed for illustrating the service ecosystem meta-model are exemplified in Figure 3.2 and Figure 3.3. As discussed above, the UML class diagram notation [111] is used for illustrating the meta-model elements. Figure 3.2 describes a fragment of the meta-model that defines seven (7) elements: *System* and its three specializations (*PhysicalSystem*, *DigitalSystem* and *AbstractSystem*), and *Set*, *IntensionalSystem* and *Relationship* as specializations of the *AbstractSystem* element.

Class named *System* occurs twice in Figure 3.2 while class named *AbstractSystem* occurs both in Figure 3.2 Figure 3.3. All occurrences of classes with the same name in the diagrams represent the same meta-model element; correspond-

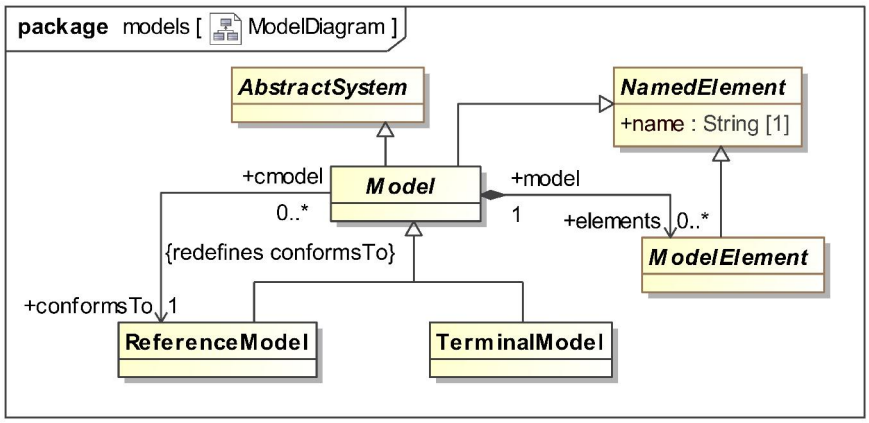

Figure 3.3: Modelling conventions: attributes and redefinitions.

ingly, the meta-model elements are uniquely named. When same class occurs more than once in a class diagram the out-going associations are only described in one place.

Naming of association ends is used in the meta-model illustrations for identifying relationships between classes instead of association names [111]. This is due to the fact that association names are not used in MOF-based meta-modelling but instead navigable association ends are regarded as properties of classes. All association ends are named in the meta-model and they are considered as public properties of their owning classes. All class properties, including association ends, are attached with a multiplicity conforming the the UML-notation [111].

In the class diagrams illustrating the service ecosystem meta-model both one-way and two-way associations are utilized. One-way associations are navigable only to the direction of the arrow-head. For example, *System* owns the association end named *conformsTo* as illustrated in Figure 3.2. The corresponding association can be read as *"System conformsTo System"*. Two-way associations are represented with lines without any arrowheads. A two-way association is defined in Figure 3.2 between the elements named *System* and *Set* with roles names of *elementOf* and *element*. The the two-way association in the example can be read as: *"System (is an) elementOf Set, and Set (has) element(s) (of kind) System"*. Navigable association ends are owned by the classes located at the opposite side of the association line. In the example, *System* owns the *conformsTo* association end, i.e. the class named *System* has a similarly named property which may refer to another *System*.

Two kinds of associations are used in the service ecosystem meta-model: regular associations representing named references between classes and composite associations representing whole-part relationships. Composite associations

are identified by a filled diamond shape. The diamond shape is located at the composite-end (the owning end) of the association. In Figure 3.3 a *Model* is a composite class which includes zero or more *ModelElement*s. In the meta-class this composition is used to denote conceptual "ownership" or inclusion of elements: a *Model* owns the included model elements. Each class may be included in at most one composite. The multiplicity of the association end at the composite-side of the association is always exactly one (i.e. a *ModelElement* is necessarily included in exactly one *Model*).

In addition to association ends classes can include attributes. In Figure 3.3 the class named *NamedElement* includes an obligatory (multiplicity is exactly one) attribute named *name* which is of type *String*. Primitive types, such as *String* or *Integer*, are part of the MOF industry standard and usable as such in modelling elements.

Redefinition of properties is utilized in the meta-model for expressing specialization of inherited properties, especially association ends. Redefinition is supported by the UML-notation with a *redefines* tag. In Figure 3.3 model element named *Model* redefines association end named *conformsTo* which is inherited from class *System* through a generalization hierarchy. This redefinition means that a *Model* can only *conformTo* other *Models*, but not *System*s which was the original definition of the association. In addition to redefinitions, property sub-setting (declared by *subsets* tag) [113] is utilized in some of the meta-model diagrams. MOF standard [112] does not actually support property redefinition or subsetting; these constraints described in the UML representation of the meta-model are however encoded in the model transformations and other modelling tools developed in this thesis.

3.3 Foundations for service ecosystem modelling

For establishing the design principles defined in Section 3.1 and to enable technology-agnostic declaration of ecosystem behaviour, the service ecosystem meta-model is constructed over a small core of foundational concepts. Support for efficient engineering practices in service ecosystems is provided by constructs formalizing the foundations of model-driven engineering [44, 45, 54]. For describing behaviour in service ecosystems, the meta-model defines a classification of events and behaviour. These constructs are described in the following.

The service ecosystem meta-model formalizes the foundations of model-driven engineering using a systemic approach, where models are considered as systems representing other systems. Following the literature in the theory of model-driven engineering [44, 45, 54], systems are classified into physical, digital and abstract systems. A system can be a representation of another system or conform to an-

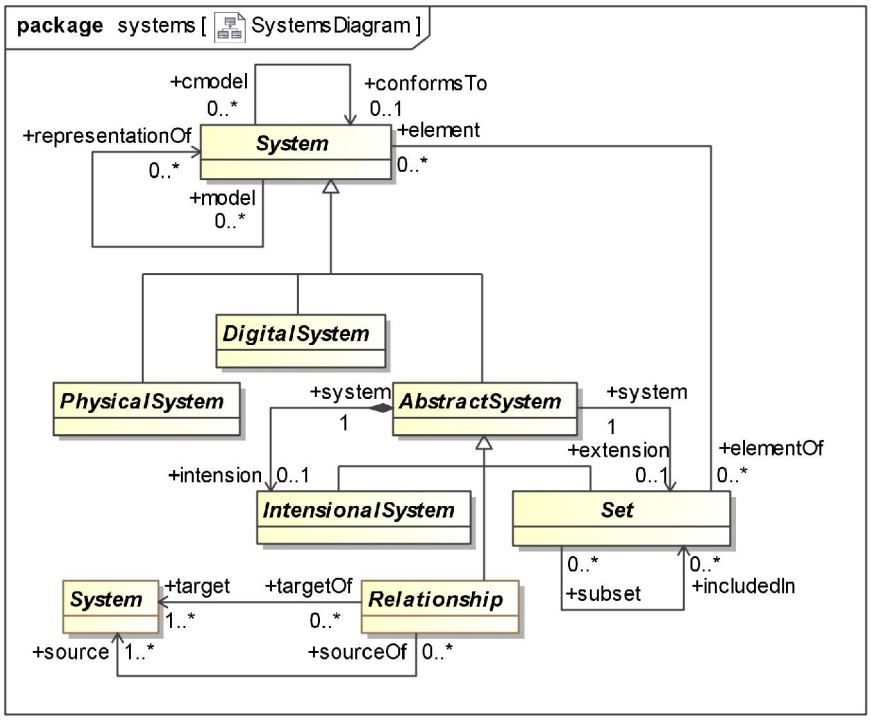

Figure 3.4: Hierarchy of systems.

other system. In addition, a system can be part of a set of systems, where a set is a specific kind of an abstract system. An abstract system can have both intensional and extensional part; this allows linguistic and ontological modelling of abstract systems, as discussed in [8]. The intensional part of the abstract system defines its characteristic features while the extensional part defines the collection of instances of the corresponding abstract system.

A system can play the role of *model* with respect to another system [44]. That is, when a system is considered as a *representation of* another system, we say that the first one is a model of the second one, the *system under study*. A model represents the system under study in a way that satisfies Liskov's principle of substitutability [11]. Liskov's principle [91] states that two entities are substitutable with each other if and only if every property that can be proved about an entity can also be proved about the other. The preceding classification and relationships between systems are formalized in the meta-model by constructs that are illustrated in Figure 3.4.

Systems can be related by domain-specific relationships. For this purpose, a concept of *Relationship* is provided in the meta-model as illustrated in Figure 3.4. A *Relationship* is considered as a kind of abstract system which refers to other systems known as the source and the target of the relationship. A relationship

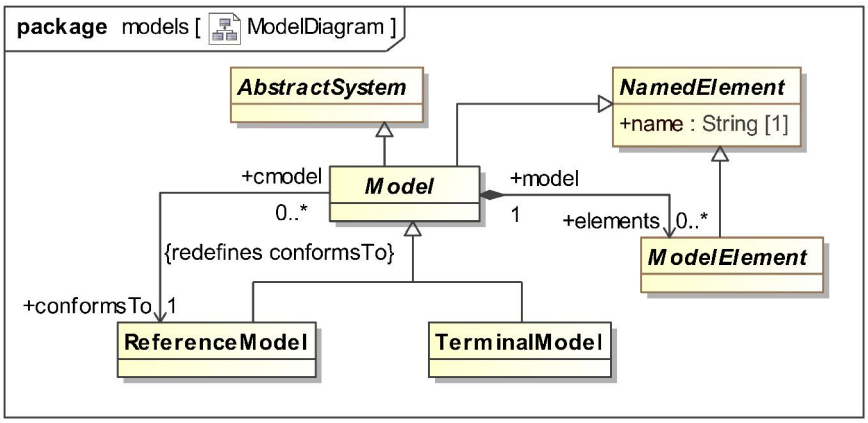

Figure 3.5: Models as abstract systems.

may have multiple sources and targets.

The service ecosystem meta-model introduces an explicit concept for representation of models. As illustrated in Figure 3.5, a model is considered as an abstract system and a named element, which includes a collection of model elements. There are two different kinds of models that are used for practising model-driven engineering. So-called *reference models* [75, 158] are used for defining modelling languages. Reference model defines the typing rules for models, that is, the kinds of model elements and the way they can be arranged, related, and constrained [17]. A reference model thus specifies the rules and the language for describing corresponding kinds of models. At the instance level, models are constructed in conformance to a reference model. A model M is said to conform to its reference model RM if and only if each model element in M has its corresponding meta-element in RM [17]. A model that does not act as a reference model to any other model, is typically called a *terminal model*. The relationships between a model and its reference model is illustrated in Figure 3.5.

Service ecosystems involve different kinds of behaviour associated with service life cycles, services and service endpoints, for example. Behavioural modelling is founded on the concepts of *Event* and *Behaviour* as illustrated in Figure 3.6. An *Event* in the meta-model is considered as a specific point or duration of time where a certain behaviour is enacted. An event is enabled and can be enacted if and only if all its successor events have already happened. Events are classified into phases and actions, making a distinction between events associated with complex and simple behaviour: a *Phase* is associated with a process, while an *Action* is a sort of event which is associated with an (atomic) activity.

A *Process* is considered as a behavioural unit which composes a set of actions. For declaring alternate enactment paths in processes, the notion of *Choice* is used.

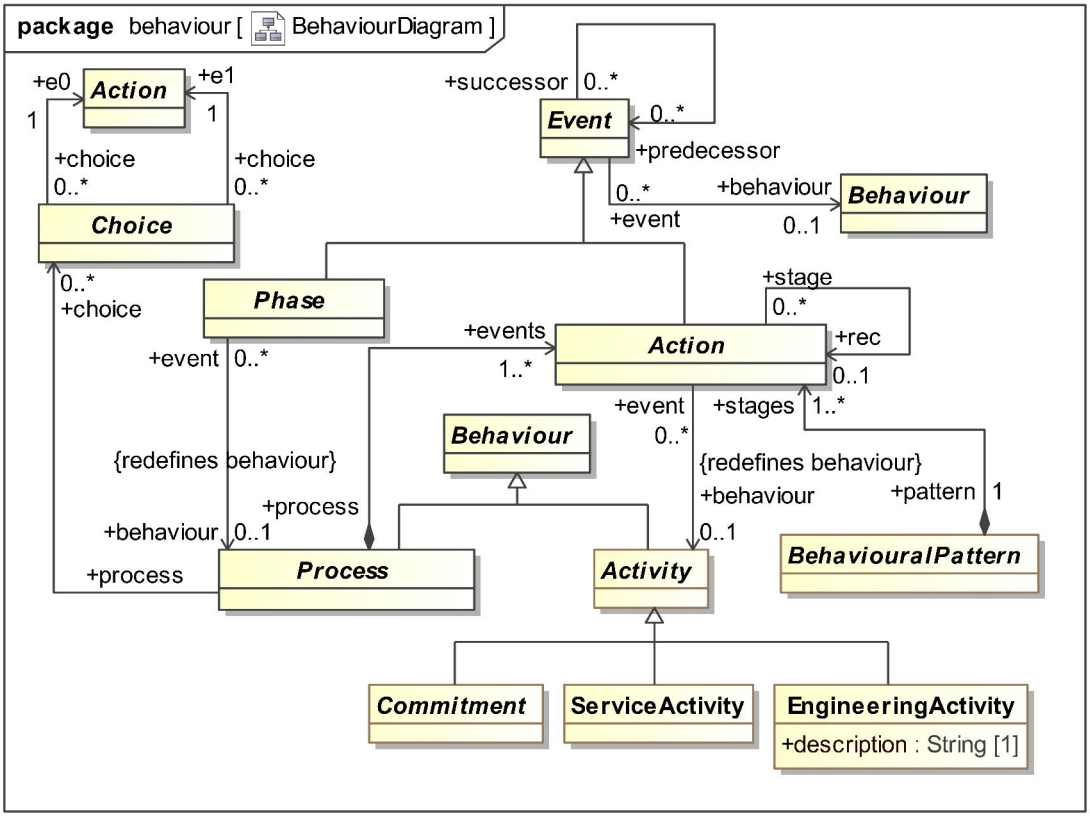

Figure 3.6: Concepts for modelling behaviour in service ecosystems.

A choice prescribes a mutual exclusion between two actions. A set of events defined by a process, the transitive closure of the successor relationships, and a set of choices makes a *Process* a behavioural description that corresponds to the notion *labelled prime event structures* [174]. Event structures provide an elegant and generic formalization of behaviour that is particularly well suited for expressing the externally observable, reactive behaviour of services, and service-oriented business processes and choreographies in varying kinds of service ecosystems.

The basic unit of behaviour is described by the concept of *Activity*, which represents *"any activity that is considered as a conceptual entity at the given level of abstraction"* [163]. An *Activity* is considered a physical or mental act of performing something that changes the state of the cooperative environment. In the case of communication behaviour, the set of activities would include send and receive activities mediating the different kinds of information entities involved between cooperation participants, for example.

Activities are further classified into three distinct categories: *1)* commitments, *2)* service activities, and *3)* engineering activities, as illustrated in Figure 3.6. Commitments are kinds of activities that declare interaction between named roles defined in choreographies. Service activities are declared in interaction schemes of service endpoint kind definitions. An interaction scheme may for example declared service activities for *create*, *read*, *update* and *delete* operations in service ecosystems that follow CRUD-like service architectures. Finally, engineering activities are defined by engineering tool types that are part of the service ecosystem engineering space declarations. They prescribe activities that can be taken in engineering methods for delivery of engineering artefacts.

Choreographies are used in service ecosystems to describe and coordinate multi-lateral, interactive behaviour taking place between a group of actors. The actors take a certain role declared in the choreography specification. The service ecosystem meta-model formalizes choreographies as structures comprising events, conditions, roles and commitments.

As illustrated in Figure 3.7, a choreography in the service ecosystem meta-model is considered as a kind of a *Process*. A *Choreography* is a composition of a collection of *ChoreographyRole*s and a collection of *ChoreographyStage*s. Each choreography role is a named element that is assigned with a set of commitments. A commitment is a kind of activity which denotes a binary, asymmetric relationship between two choreography roles: one role is considered as the *creditor* and the other role is considered as the *debtor* of the commitment. A debtor has the obligation to fulfill the commitment towards the creditor. Typically commitments are fulfilled in service ecosystems with communication activities originating from the debtor and having a creditor as the destination. The real-world effects of discharging the commitments, such as associated with co-creation of value during

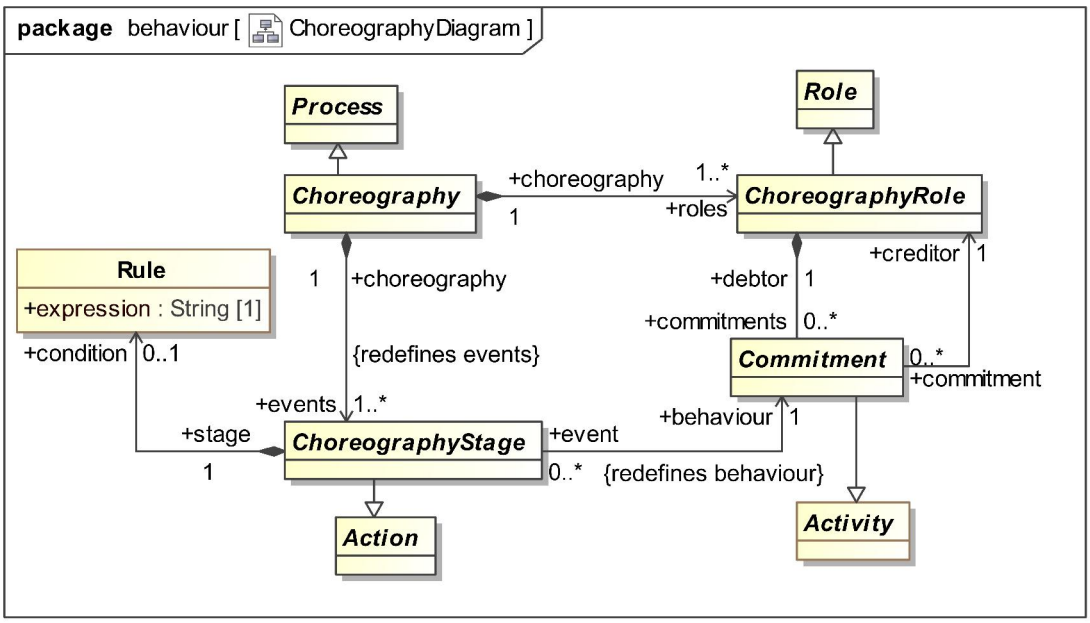

Figure 3.7: Choreography diagram.

the service delivery, are not considered by the meta-model. However, such associations with the real-world effects can be specified in domain-specific ecosystem models as extensions of the *ChoreographyStage* element, for example.

The behaviour of a choreography is declared as a partial ordering between a set of *ChoreographyStage*s. A choreography stage is an action which is labelled by a commitment, and is associated with an optional rule expression. The rule expression defines a condition under which the corresponding action represented by the choreography stage can be enacted. The set of choreography stages can be considered as a collection of event-condition-action rules (ECA) (see for example [100]) where *ChoreographyStage*s correspond to events, *Rule*s express the condition part, and *Commitment*s correspond to actions.

3.4 Service ecosystem architecture descriptions

A service ecosystem architecture description includes representations of several ecosystem aspects, such as behaviour, concepts and structure. The meta-model for architecture descriptions is illustrated in Figure 3.8. A service ecosystem architecture description is a model which comprises five kinds of sub-models: *1)* ecosystem model, *2)* domain ontology model, *3)* intension model, *4)* engineering model, and *5)* knowledge management model. These sub-models are briefly characterized below and elaborated in subsequent sections.

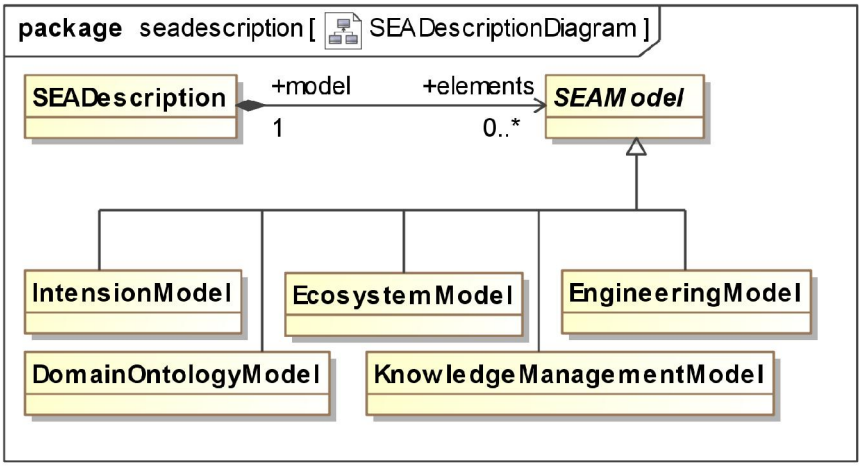

Figure 3.8: Service ecosystem architecture description diagram.

The constructs defined in the ecosystem model facilitate especially service ecosystem governance. The ecosystem model provides modelling constructs for specifying capabilities, structure, behaviour, artefacts and actors of service ecosystems. Ecosystem models are specified primarily by ecosystem providers and infrastructure providers for defining the purpose, scope and structure of service ecosystems.

A domain ontology provides an explicit and formal description of the corresponding portion of reality using selected level of granularity and viewpoint [59]. The service ecosystem concepts are made explicit by the domain ontology model. An ontology in this context is considered as a shared, descriptive, structural model that represents reality by a set of concepts, their interrelations, and constraints under the open-world assumption [5].

The domain ontology model provides means for establishing semantic interoperability by specification of service ecosystem concepts. A service ecosystem architect defines a domain ontology by extending the concepts of the domain ontology model. The domain ontology model includes definitions for different service ecosystem entities (e.g. services and legal entities) and their interrelationships. More over, the domain ontology model includes elements that are used for specification of domain concept hierarchies and relationships.

Interoperability knowledge in service ecosystems is specified with constructs defined in the domain ontology model and the intension model. While domain ontology model specifications address hierarchies and relationships between domain concepts, the intension model provides definitions for structural properties of the concepts. The relationship between the domain ontology model and the intension model makes explicit the inter-dependencies between ontological and linguistic

meta-modelling discussed in Section 3.1. Ontological meta-modelling facilitates dynamic user extensions to modelling concepts, modelling notations and the models created from them, while linguistic meta-modelling is used for defining modelling languages and their primitives on the meta-model level [8]. The unification of the two meta-modelling dimensions in the service ecosystem meta-model enables specification and management of extendable, domain-specific vocabularies and modelling languages.

The engineering model constructs are used for specification of the engineering knowledge and for enabling efficient engineering practices in service ecosystems. Efficient service engineering is supported by prescribing reusable engineering assets. Engineering assets include definitions for product models and binding models, as well as engineering methods. Product models prescribe artefacts and their compositions appropriate to be utilized in service engineering methods. Typically the decompositions of artefacts would follow the structure of knowledge items, but other kinds of structuring allowing more efficient engineering efforts can also be utilized. Binding models declare the rules for late encapsulation of services with bindable features: binding specifications prescribe what kinds of features can be bound to service roles, business transactions or service connections. A situational method engineering approach is applied for specification of engineering knowledge. In this approach, methods are composed of autonomous method chunks [103].

The knowledge management model constructs provide means for establishing interoperability and engineering knowledge management infrastructures required in open service ecosystems. The knowledge management model provides means for specification of knowledge repositories, and knowledge items and relationships managed in the repositories. A knowledge repository represents a coherent knowledge base comprising a collection of artefacts which conform to a common intensional definition. In the service ecosystem meta-model, a single domain concept may have several roles in ecosystem life cycles: concepts can be declared as representations of multiple life-cycle artefacts. Such one-to-many relationship between concepts and life-cycle artefacts is needed for encoding different roles of knowledge in service ecosystem life cycles. This enables formalization of context and situation specific interpretations for concepts, and variable consistency criteria for their intensions.

3.5 Ecosystem model

The service ecosystem meta-model provides constructs for defining the scope, structure and behaviour of service ecosystems; these constructs are defined under the ecosystem model. The scope of service ecosystems is defined with ecosystem

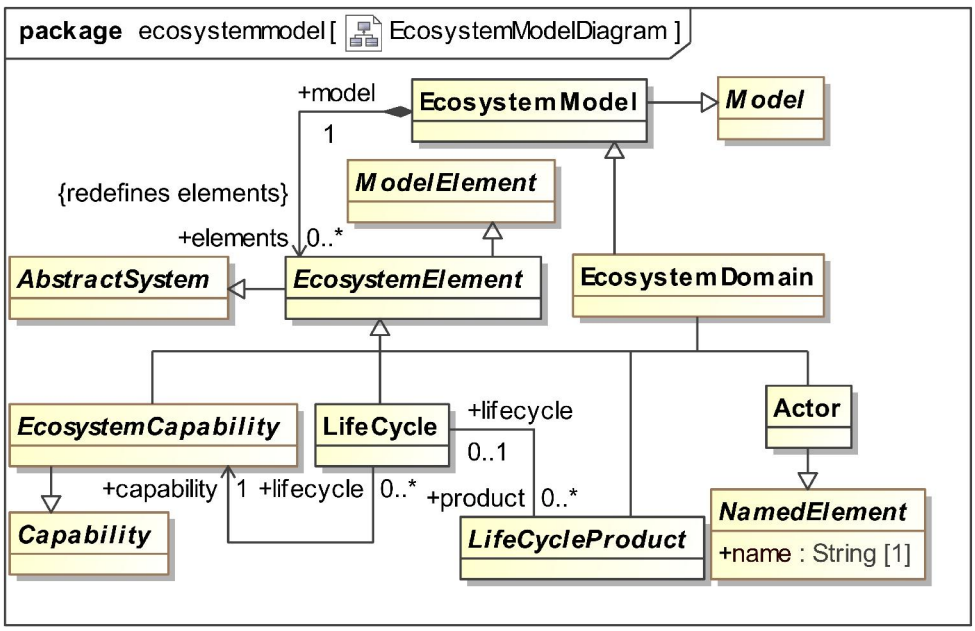

Figure 3.9: Ecosystem model diagram.

capabilities, structure with ecosystem domains, and behaviour with ecosystem life cycles. The ecosystem model identifies the actors and infrastructure services in service ecosystems. The ecosystem model addresses especially service ecosystem governance and knowledge management issues. The ecosystem model comprises a collection of ecosystem elements, as illustrated in Figure 3.9. The ecosystem elements include ecosystem capabilities, domains, life cycles and life-cycle products, and actors; these individual elements are elaborated below.

3.5.1 Ecosystem capabilities

Ecosystem capability specifications are used for representing the purpose of the ecosystem. An ecosystem capability denotes an ability of a service ecosystem to perform actions and to deliver qualities to its members. Ecosystem capabilities are represented in the meta-model by the concept of *EcosystemCapability*, as illustrated in Figure 3.10. An ecosystem capability is a specialization of the *Capability* concept. A capability can exploit one or more other capabilities for delivering the corresponding ability.

Ecosystem capabilities are further categorized into functional and qualitative capabilities, and knowledge bases, as illustrated in Figure 3.10. A *Functional-Capability* represents a service ecosystem's ability to deliver some function to its members. Functional capabilities are supported by qualitative capabilities, repre-

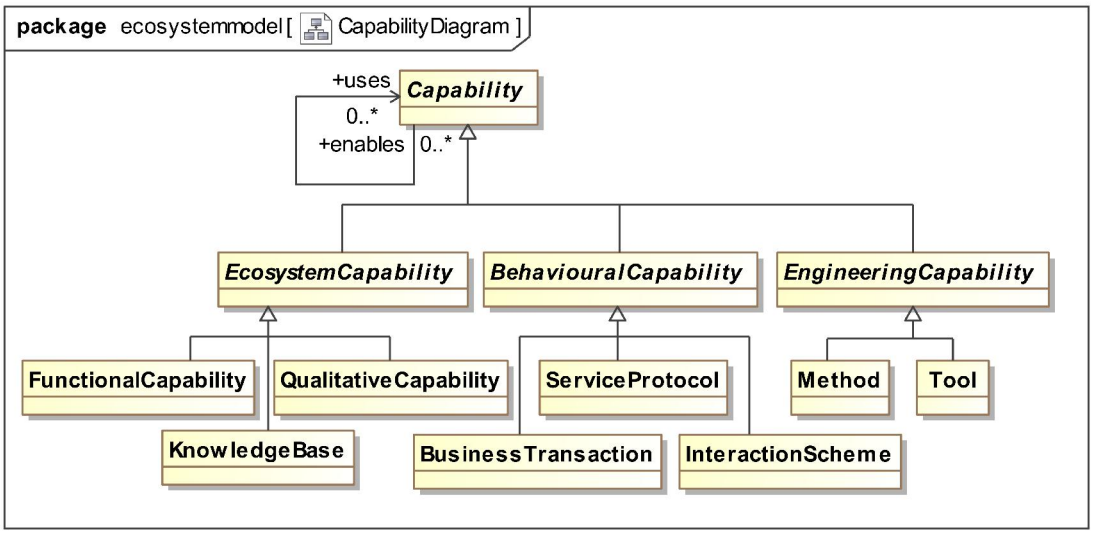

Figure 3.10: Capability diagram.

sented with the concept of *QualitativeCapability* in the meta-model. A qualitative capability specifies how ecosystem functions must be delivered. An example of functional capability is dynamic collaboration establishment, which is supported by a qualitative capability of dependability (e.g. trusted and secure interactions). A *KnowledgeBase* represents a capability of the service ecosystem to maintain a coherent collection of artefacts utilized in life cycles; the notion of a knowledge base will be elaborated below.

In addition to ecosystem capabilities, the service ecosystem meta-model includes categories for behavioural and engineering capabilities. A *BehaviouralCapability* represents a demarcated unit of behaviour exposed by a service ecosystem component. The behavioural capabilities are further categorized into business transactions, service protocols and interaction schemes which are exposed by business services, component services, and service endpoints, correspondingly. The behavioural capabilities are discussed in more detail later.

Engineering capabilities express means for enabling service engineering practices in ecosystem domains. In the meta-model, engineering capabilities are represented by the concept of *EngineeringCapability* illustrated in Figure 3.10. There are two concrete engineering capabilities: methods and tool types. A method provides a process and guidelines for production of a part of an engineering product. A tool type provides a specification of an engineering tool for delivery of engineering activities producing required product elements. These engineering capabilities are further elaborated in Section 3.8.

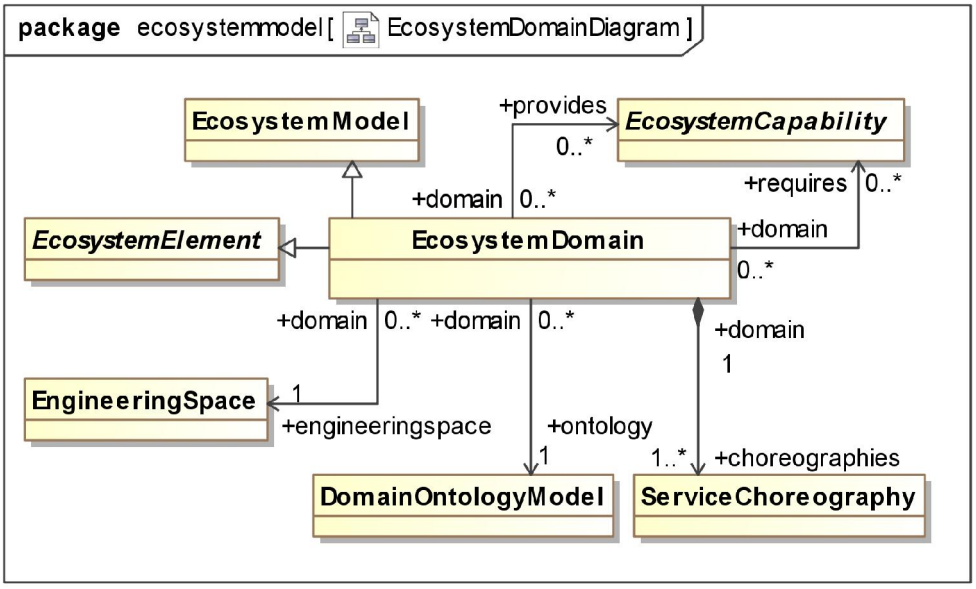

Figure 3.11: Ecosystem domain diagram.

3.5.2 Ecosystem domains

Ecosystem domains are the main construct provided by the meta-model for structuring service ecosystem architectures. For example in cloud computing environments (see e.g. [37]) the notion of domains could be used for representing different cloud computing models such as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). An ecosystem domain is considered as an ecosystem model itself, as illustrated in Figure 3.11. An ecosystem domain may thus include any number of ecosystem elements, including other ecosystem domains.

An ecosystem domain may provide a number of capabilities to be utilized by other domains. This facilitates control over managing dependencies between ecosystem domains. In a cloud computing environment, the PaaS-domain might provide service deployment capabilities to be utilized by a SaaS-domain, for example. In the Pilarcos framework, which is discussed in Section 5.2, an ecosystem domain representing a reputation system [133] provides a trust management capability to be utilized in dynamic collaboration establishment processes.

An ecosystem domain includes specifications for a collection of service choreographies, represented by the element *ServiceChoreography* in Figure 3.11. The notion of service choreography is defined in the domain ontology model discussed in Section 3.6. In addition, an ecosystem domain is associated with an engineering space, and a domain ontology model; these constructs are elaborated in Sec-

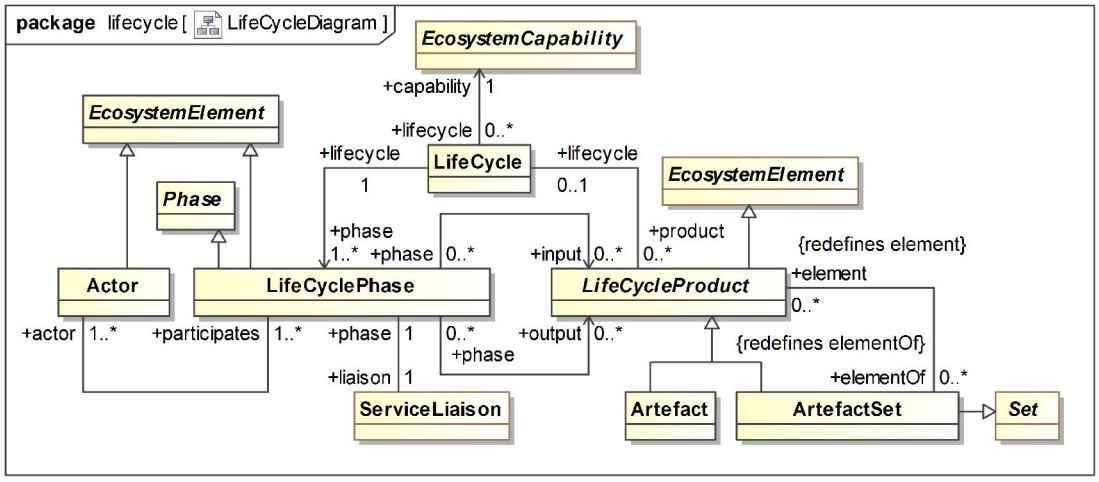

Figure 3.12: Life-cycle diagram.

tion 3.8 and Section 3.6, correspondingly.

3.5.3 Ecosystem life cycles

Within ecosystem domains capabilities are realized with service ecosystem life cycles, infrastructure services and qualitative features. Service ecosystem life cycles are ecosystem elements that compose a set of life-cycle phases and life-cycle products. The formalization of life cycles in the service ecosystem meta-model is illustrated in Figure 3.12. Life-cycle phases are represented by the element named *LifeCyclePhase*. Each life-cycle phase can be associated with one or more life-cycle products acting as inputs and outputs for the corresponding phase. A life-cycle product is represented by the element named *LifeCycleProduct*. Artefacts and artefact sets are considered as realizations of life-cycle products. A life-cycle phase is associated with the set of actors that are eligible for taking roles in the phase. The roles are declared through a service liaison (discussed in Section 3.6), which associates a service choreography with a service life-cycle phase.

3.6 Domain ontology model

The domain ontology model supports establishment of semantic interoperability in service ecosystems. Semantic interoperability is addressed by the domain ontology model by formalizing the conceptual model of service ecosystems. The domain ontology model includes definitions for service contracting concepts, cooperation facilities and service ecosystem entities. The domain ontology model

enables definition of holistic, unifying ecosystem vocabularies which are utilizable for establishing semantic interoperability in service ecosystems.

The domain ontology model comprises a set of domain ontology elements, as illustrated in Figure 3.13. Each domain ontology element is considered as an abstract system in addition to of being a kind of model element. This allows for specification of conformance and representation relationships between the model elements inside domain ontology models, which is required for addressing service ecosystem dynamism through extendability of domain ontologies.

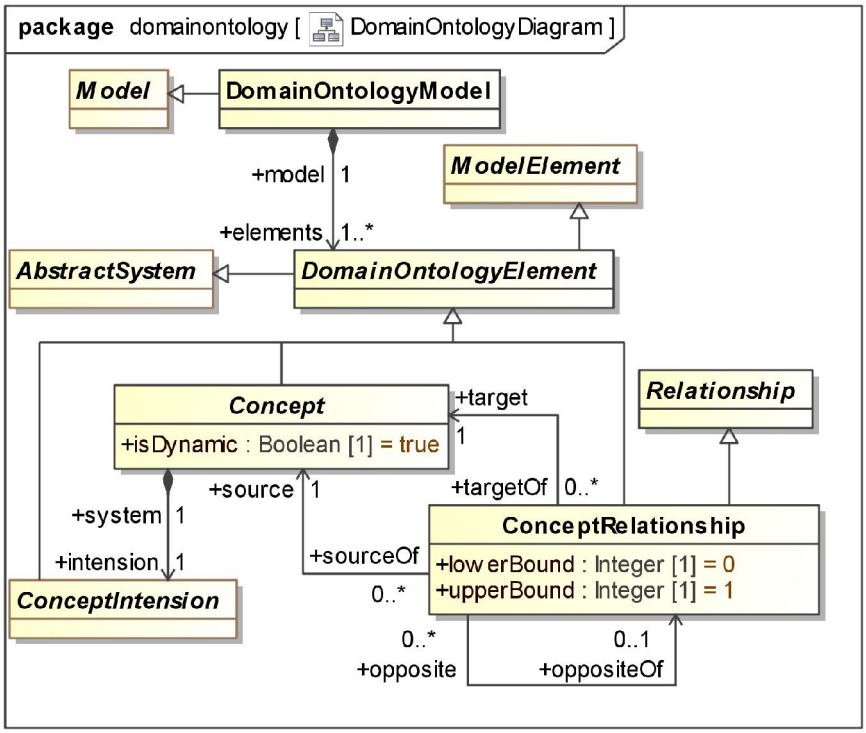

Figure 3.13: Domain ontology diagram.

Domain ontology elements are further classified into concepts and concept relationships. Domain concepts are represented in the meta-model by the notion of *Concept*. Each concept is provided with a declaration of its intension, represented by the element named *ConceptIntension*. As illustrated in Figure 3.13, each concept owns its intension which defines the characteristic features of the concept. The characteristic features are specified with modelling elements specialized from the *InstantiableElement*, especially *Feature* elements which are described in Section 3.7. Consequently, the intension of a concept is described as a collection of selected features.

The composite one-to-one association between concepts and their intensions

represents a design decision that has been made to separate the ontological elements from their intensional features, and their corresponding inter-dependencies. This design decision enables separation of the ontological space and features space of the domain ontology model. The separation between ontological and feature spaces in the service ecosystem meta-model supports reuse of feature definitions when describing concept intensions, and especially, utilization of feature management theories and practices, such as described in [15], for managing service ecosystem features.

In the ontological space, concept specializations and relationships (represented by the *ConceptRelationship* in Figure 3.13) between concepts are used. The intensions of concepts are defined as collections of features that are defined in the feature space, more concretely, using the intensional model described in Section 3.7. Inter-dependencies between features, such as mutual exclusion, can be declared using the modelling construct of *FeatureInteraction* described in Section 3.7.

3.6.1 Top-level concepts

The foundational service ecosystem concepts defined by the service ecosystem meta-model include service liaisons, service contracts, concept sets, service offers, service channels and co-operation facilities. Each concept can provide a representation of a life-cycle product, i.e. an artefact or artefact set used in ecosystem life cycles. The domain ontology model defined in the service ecosystem meta-model is extended in service ecosystem architectures to represent the domain-specific versions of these generic concepts. The top-level concepts of the domain ontology model are illustrated in Figure 3.14.

The element *DomainConcept* is utilized for specifying domain-specific concepts that are not addressed by the service ecosystem meta-model; such concepts might include for example products associated with the ecosystem. The intension of domain concepts comprises a set of user-definable domain features which may have conformance relationships with each other.

The concepts of *EntityKind* and *Entity* declare conceptual typing hierarchies: each entity conforms to a unique entity kind. These categories of entities and entity kinds include concepts for describing services (business and component services), service endpoints and legal entities, for example. The semantics of the conformance relationships between the domain ontology concepts is ecosystem-specific, and are validated with domain-specific infrastructure services and tools. The meta-model does not address the formalization of such domain-specific semantics, however, regulations and constraints describing such semantics can be weaved as comments to the corresponding modelling elements.

An ontological concept is represented by the *Concept* element, which is considered as an abstract system as illustrated in Figure 3.15. Each *Concept* can be

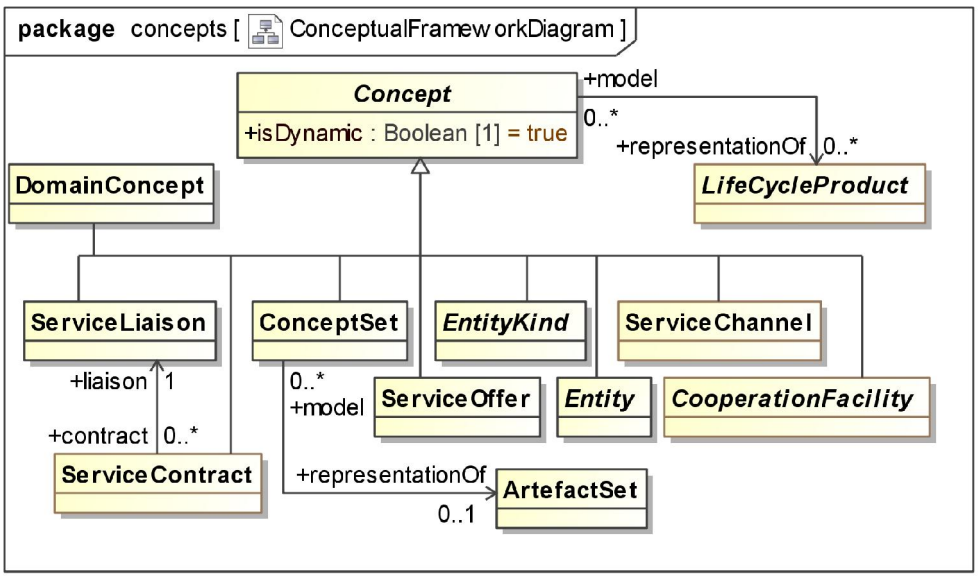

Figure 3.14: Top-level concepts of the domain ontology model.

declared as either static or dynamic. Static concepts can not be instantiated during service ecosystem operation; they can be utilized to represent for example immutable services in Infrastructure-as-a-Service cloud environments. Dynamic concepts are utilizable for expressing extendable service ecosystem knowledge. A *Concept* is a *GenerizableElement* for enabling declaration of abstraction hierarchies between concepts.

The intensional meaning of a concept is provided by the *ConceptIntension* element and extensional meaning by a set of concepts (*ConceptSet* element). The concept intension comprises a set of *InstantiableElement*s which in essence enable deep instantiation [7] hierarchies between intensional elements. Each *InstantiableElement* is associated with a potency which declares if the corresponding element is an instance (potency is zero) or instantiable at a lower semantic modelling level. The service ecosystem meta-model supports natively two-level instantiation hierarchies, which are represented in the domain ontology with *EntityKind* - *Entity* -pairs, and in the intension model with *Feature* - *Property* -pairs. Deeper instantiation hierarchies can be defined by exploiting higher potencies for features and defining conformance relationships between *EntityKind*s and *Feature*s. The concept extension comprises the set of ontological instances of the concept. For example, the intension of a concept named *"Organization"* would define the characteristic features shared by all organizations (such as having a set of sub-organizations and employees) while its extension includes all the organizations available in the service ecosystem.

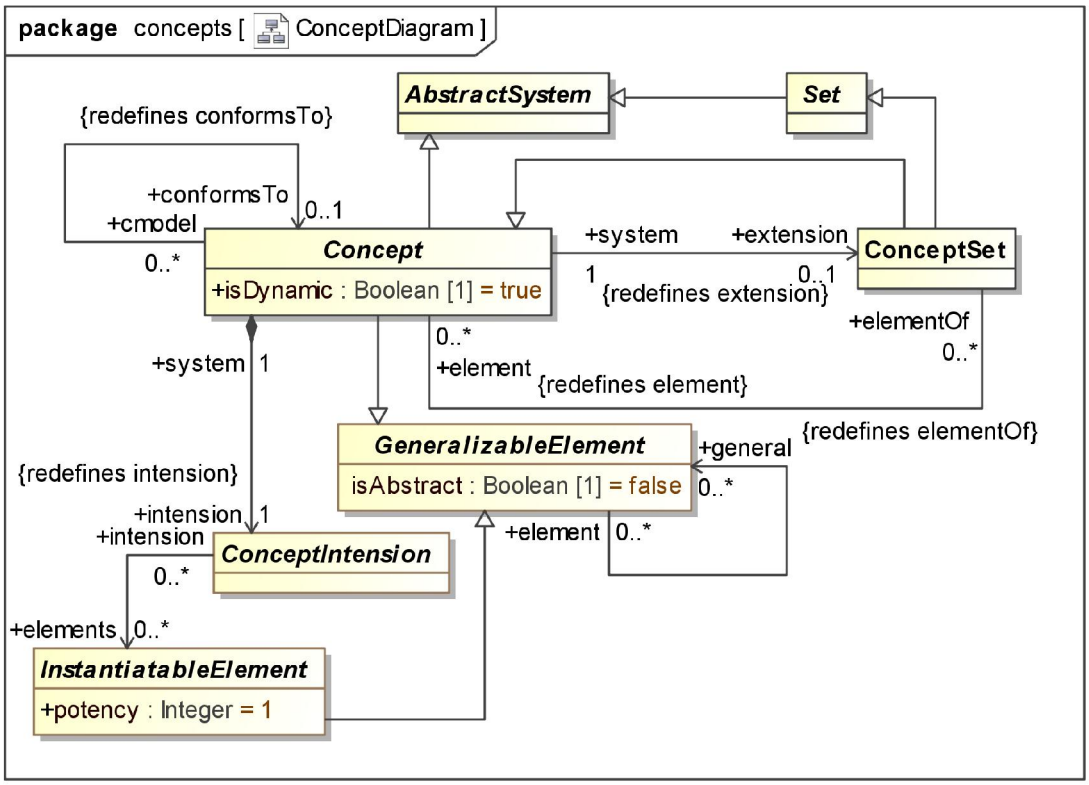

Figure 3.15: Domain ontology concepts and concept sets.

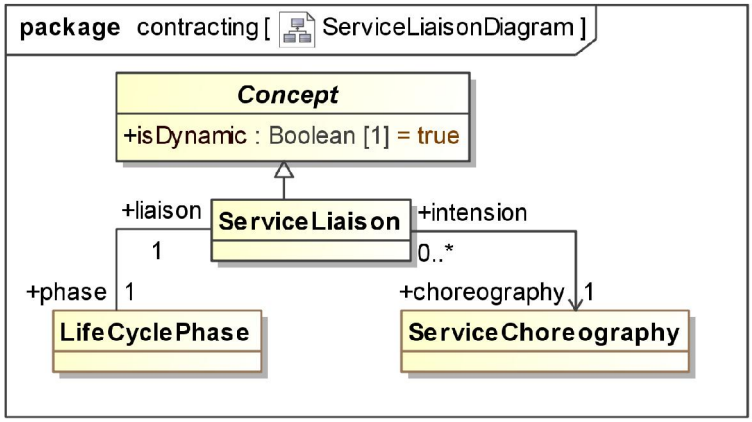

Figure 3.16: Service liaison diagram.

3.6.2 Contracting concepts

The service contracting concepts included in the domain ontology model include service liaisons, service contracts, service channels and service offers.

Service liaisons declare the kinds of service-based collaborations that can be enacted in service life-cycle phases. A service liaison serves as a specification for service contracts in the corresponding life-cycle phase. A service liaison is defined in the service ecosystem meta-model as an association between a service choreography and life-cycle phase, as illustrated in Figure 3.16. The intension of a service liaison comprises a set of feature bindings, which will be discussed in Section 3.7.

A service choreography is defined in the meta-model as a specialization of the choreography introduced in Section 3.3. The formalization of service choreographies is illustrated in Figure 3.17. A *ServiceChoreography* extends the concept of *Choreography* and comprises sets of service connections, business transactions, business actions, and service roles. A *ServiceConnection* makes explicit the negotiable relationship between service roles and acts as a target for binding of extra-functional features. A *BusinessTransaction* is a declaration for a set of business actions that are needed for fulfilling some meaningful business goal. A *BusinessAction* is a specification for an event which realizes a service commitment. A *ServiceCommitment* denotes an exchange of a life-cycle product (i.e. artefact or artefact set) or meaning activity between service roles over a service connection.

Service contracts are utilized in service ecosystems for establishing dependability of interactions between service providers and consumers. Service contracts can involve two or more members. Service contracts can be subject to dynamic

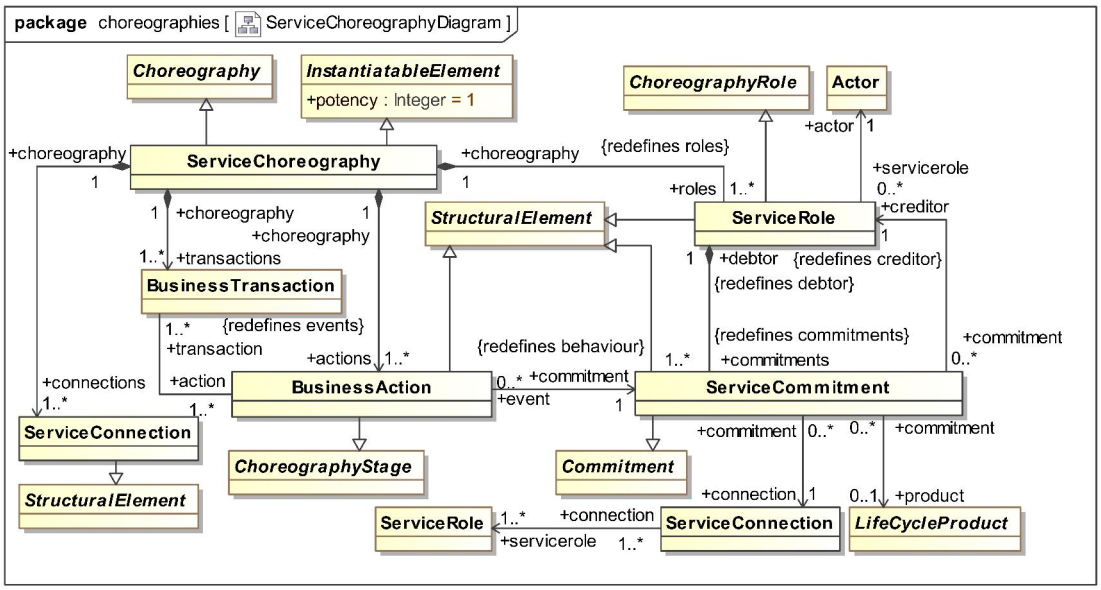

Figure 3.17: Service choreography diagram.

negotiations, especially in open service ecosystems. However, static or implicit service contracts are a feasible option in service ecosystems following a more closed approach to service collaboration establishment.

The service contract concept is formalized with modelling elements illustrated in Figure 3.18. Service contracts are defined in the domain ontology model as concepts that compose a collection of service channels to realize a service liaison. The intension of a service contract comprises a set of property bindings and a set of so-called commitment operations. The property bindings included in service contract intensions are mostly taken for the service offers published by participants of the contract; they are possibly refined and negotiated during contract establishment processes.

Service channels enable provisioning of business services through different interaction media and with varying qualitative properties affecting the service interactions. The concept of service channel defined in the meta-model enables loose coupling between business services and late encapsulation of service features. The *ServiceChannel* concept, illustrated in Figure 3.19, is not related to any other elements at the conceptual level. Instead the intensional definition of a *ServiceChannel* is used for collecting cooperation facility properties, service links and endpoint bindings to allow specification of service interactions with necessary features.

Facility features included in service channel intensions comprise binding ports and channels phases. A binding port represents a logical grouping of service op-

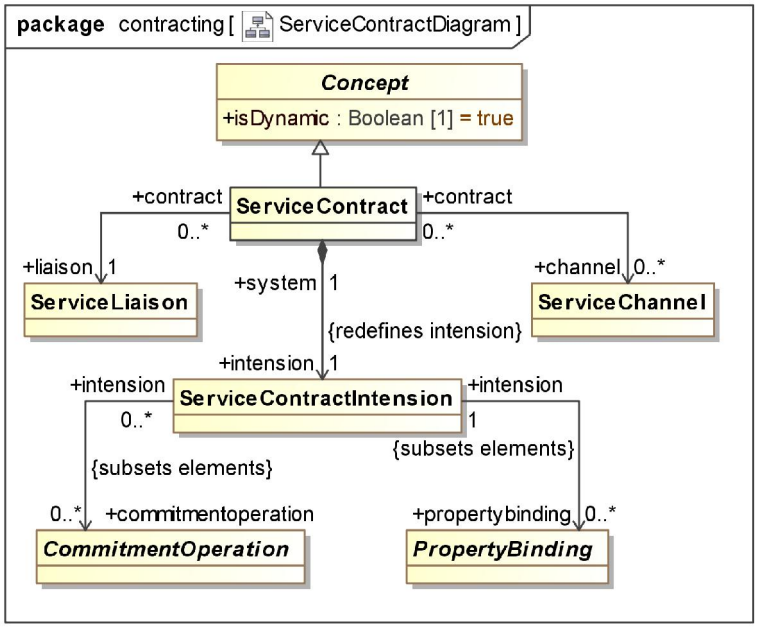

Figure 3.18: Service contract diagram.

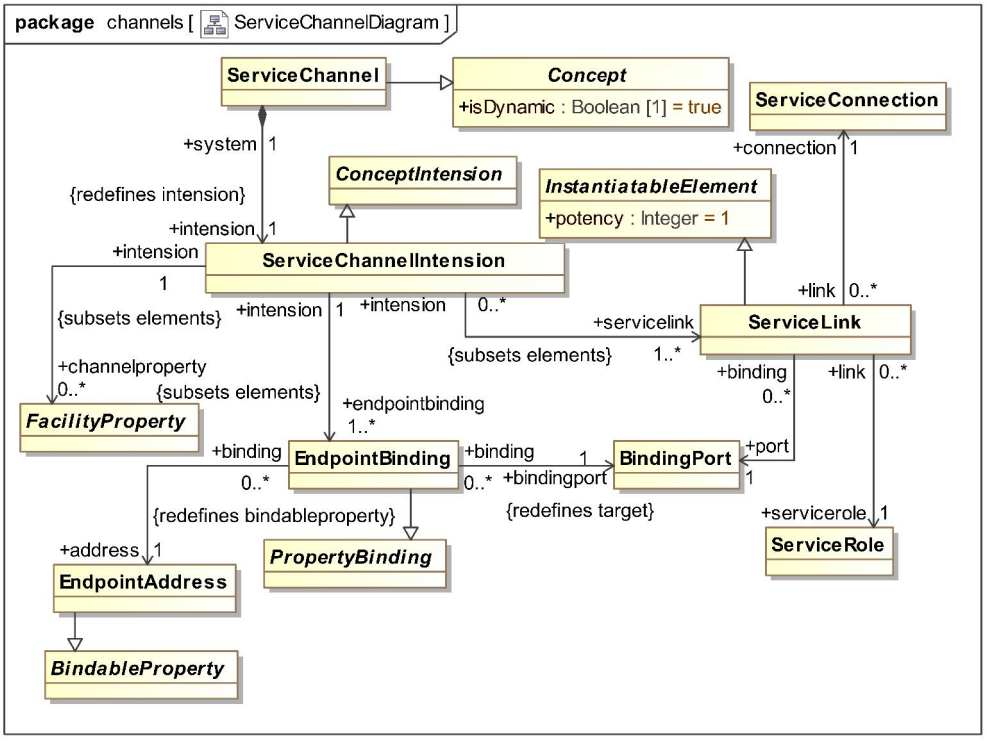

Figure 3.19: Service channel diagram.

erations which can be accessed from the same location (address). Each binding port conforms to a binding port type that is declared in a binding type specification. Binding types are used in the service ecosystem meta-model to represent different interaction mechanisms, such as document-oriented messaging or publish-subscribe paradigm.

Channel phases represent actions that need to be taken to establish the required form and quality of service communication. Each channel phase conforms to a channel feature, which are defined as part of channel type declarations. Channel types are used in the meta-model for representing especially means for realizing qualitative communication features. A channel type realizing secure messaging may include channel features representing payload encryption and decryption, for example.

A service link associates a binding port with a service connection and a service role. Consequently, a service link specifies the binding ports, and thus interaction mechanisms to be used in service connection endpoints. Finally, endpoint bindings are utilized for associating service endpoint addresses with the binding ports. Service endpoint addresses are considered as unique, resolvable identifiers enabling point-to-point communications.

The intensional definition of service contracts comprises a set of commitment operations. The commitment operations are used for discharging the commitments set for roles in the service choreography declared by the service liaison. The set of commitment operations in service contracts represent values mutually agreed on by service contract participants. The set of commitment operations must cover all the commitments declared in the service liaison associated with the contract. The commitment operations follow the ontology for commitments in multi-agent systems [150] and they are defined in the meta-model as illustrated in Figure 3.20.

The concept of *CommitmentOperation* is further classified into five different subclasses, as illustrated in Figure 3.20. The *DischargeCommitment* concept represents the basic commitment operation that is used in the end for discharging a commitment set for an ecosystem member. A commitment is discharged when the desired condition is obtained [150]; in service ecosystems the desired conditions are obtained typically with message exchanges. Commitments are discharged with the service operations provided by component services. Commitments can be cancelled with the *CancelCommitment* concept. A commitment can be canceled when a service provider cannot or will not act in accordance with their commitments. Commitment cancels are subject to cancellation clauses and policies that are in effect [150] in the given service ecosystem and service contract. *ReleaseCommitment* represents a commitment operation which eliminates the commitment and can be performed by the creditor of the given commitment [150]. A release

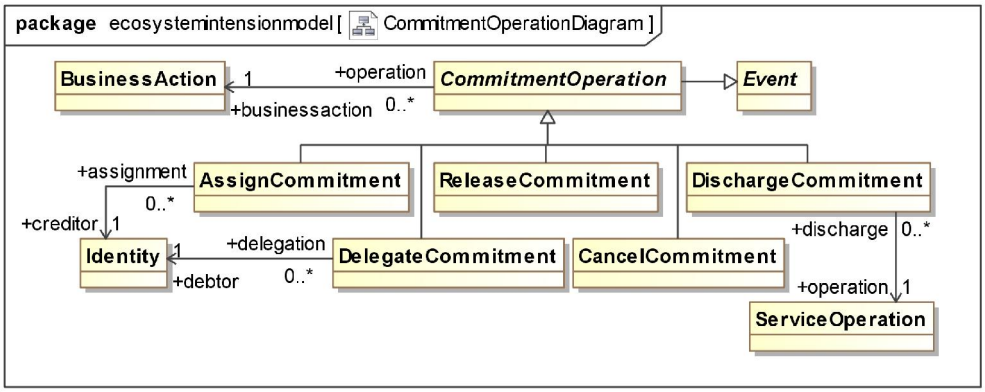

Figure 3.20: Commitment operation diagram.

commitment is distinguished from both discharge and cancel operations, because release does not mean success or failure of the given commitment [150]. The *DelegateCommitment* operation shifts the role of debtor to another identified ecosystem member. Finally, *AssignCommitment* represents a commitment operation which transfers a commitment to another creditor, and can be performed by the present creditor, if authorized by the prevailing policies [150].

Service offers are used by service providers to publish and advertise their business services in a service ecosystem. A service offer enables bundling of services. A service offer is defined in the meta-model as a concept which associates a set of business services with a legal entity. The service provider publishing a service offer defines which commitment operations are offered from the bundled business services, as illustrated in Figure 3.21. Service offer intension is defined as a collection of binding ports, a set of property bindings, and a set of port bindings.

3.6.3 Service ecosystem entities

The notion of entities is used in the domain ontology model to represent elements of service ecosystems that possess unique, resolvable identities and existence. The entities of the domain ontology model follow a two-level abstraction hierarchy where every entity in a service ecosystem conforms to an entity kind. The service ecosystem entity kinds are illustrated in Figure 3.22. Each entity kind is associated with an appropriate identity type which declares the mechanism to be used for identifying the corresponding kinds of entities. Especially, legal entity kinds are associated with a set of service ecosystem actors. The association in effect declares what kinds of roles the corresponding kinds of entities can take in the service ecosystem. The intensional definitions for entity kinds are specified as collections of features, as illustrated in Figure 3.22.

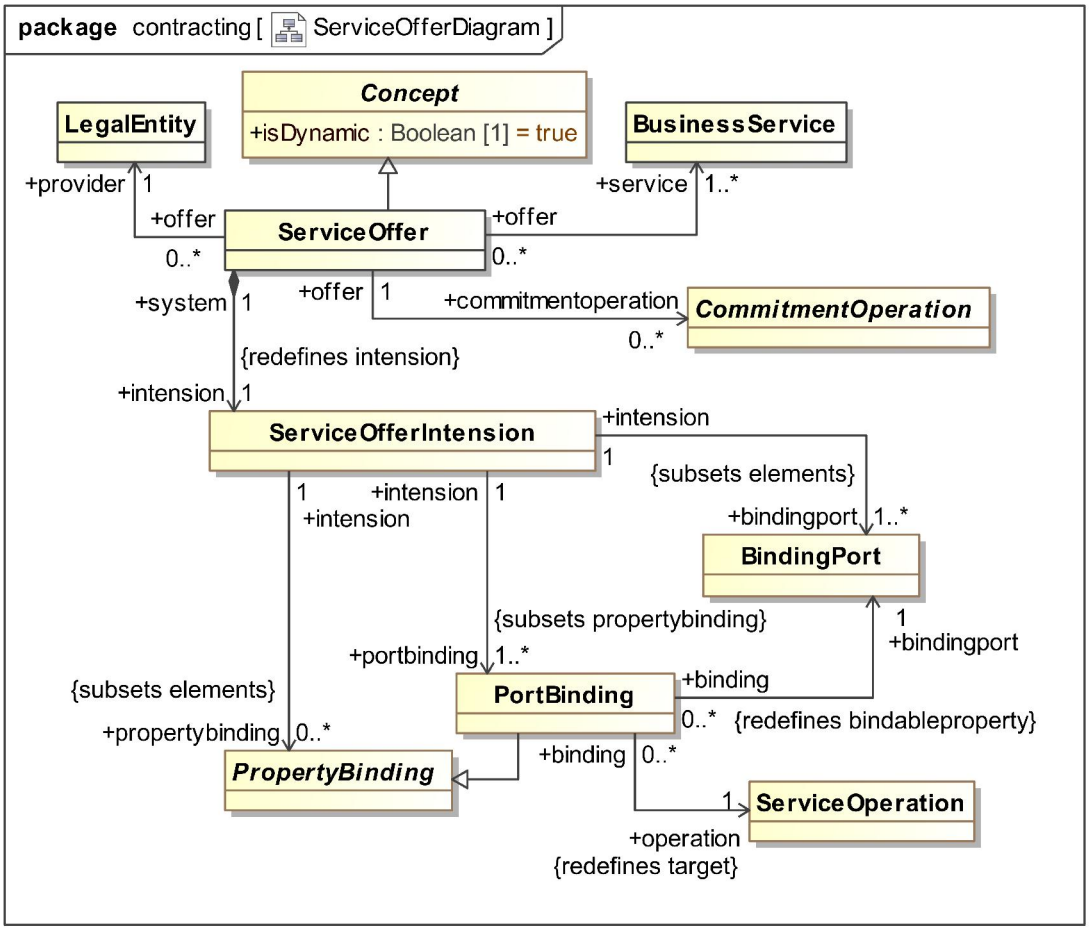

Figure 3.21: Service offer diagram.

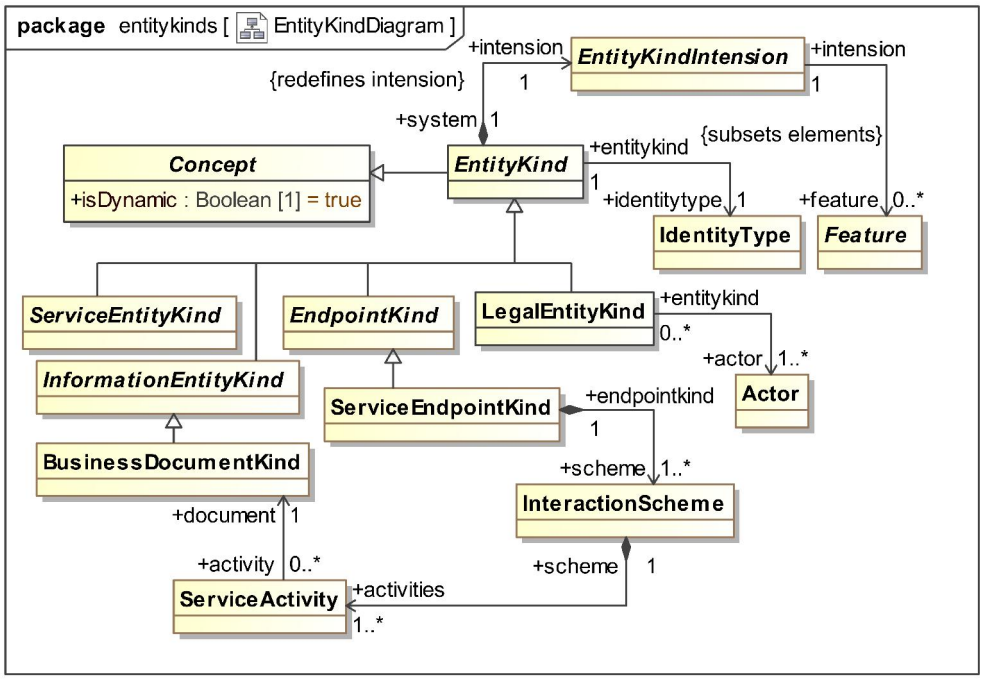

Figure 3.22: Entity kind diagram.

The domain ontology model provides definitions for four distinct kinds of entities: *1)* service entities, *2)* information entities, *3)* endpoint kinds, and *4)* legal entities. These entities illustrated in Figure 3.23 conform with the corresponding entity kinds in Figure 3.22. Each entity is associated with an identity, which must conform with the identity type declared for the corresponding entity kind. The intensional definitions for entities are specified as collections of properties.

The service ecosystem meta-model defines the notion of service as an abstract system that delivers one or more capabilities. This definition aligns with the common interpretation of services, such as utilized in the OASIS SOA-RM standardization [154]. The meta-model elaborates the service definition with a further classification of services to infrastructure services and service entities. The service definition and categorization is formalized in the meta-model with constructs that are illustrated in Figure 3.24.

Infrastructure services are used for delivering ecosystem capabilities. Infrastructure services include the category of knowledge repositories needed for managing the global knowledge base of the service ecosystem. As illustrated in Figure 3.24, the notion of *KnowledgeBase* is introduced in the meta-model as a capability for representing knowledge base management. Each knowledge base comprises a selection of life-cycle products, that is, artefacts or artefact sets. Additional infrastructure services, such as trust or reputation management systems,

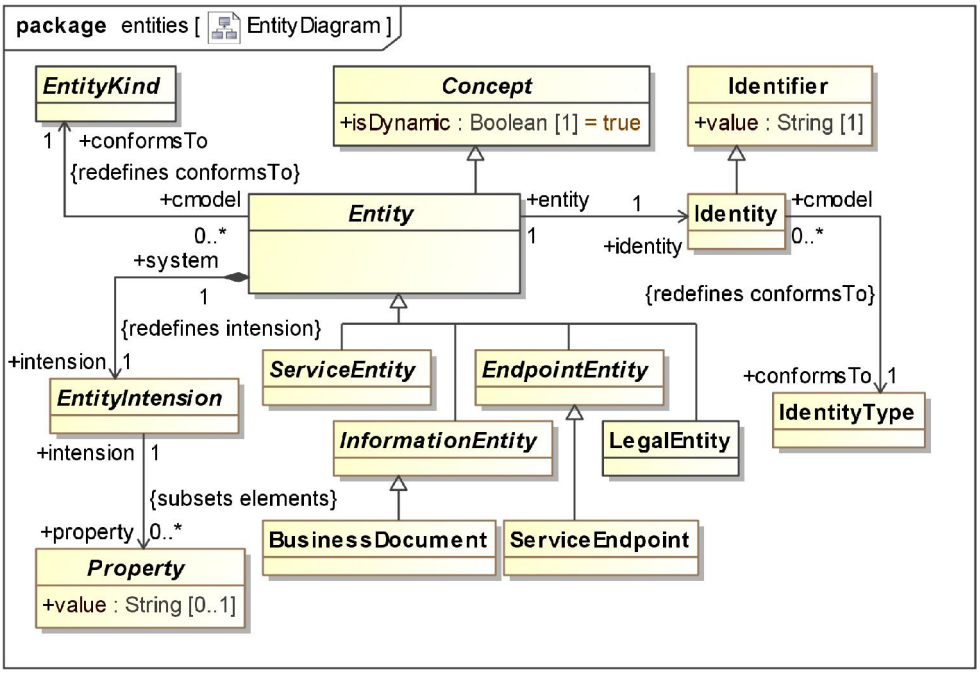

Figure 3.23: Entity diagram.

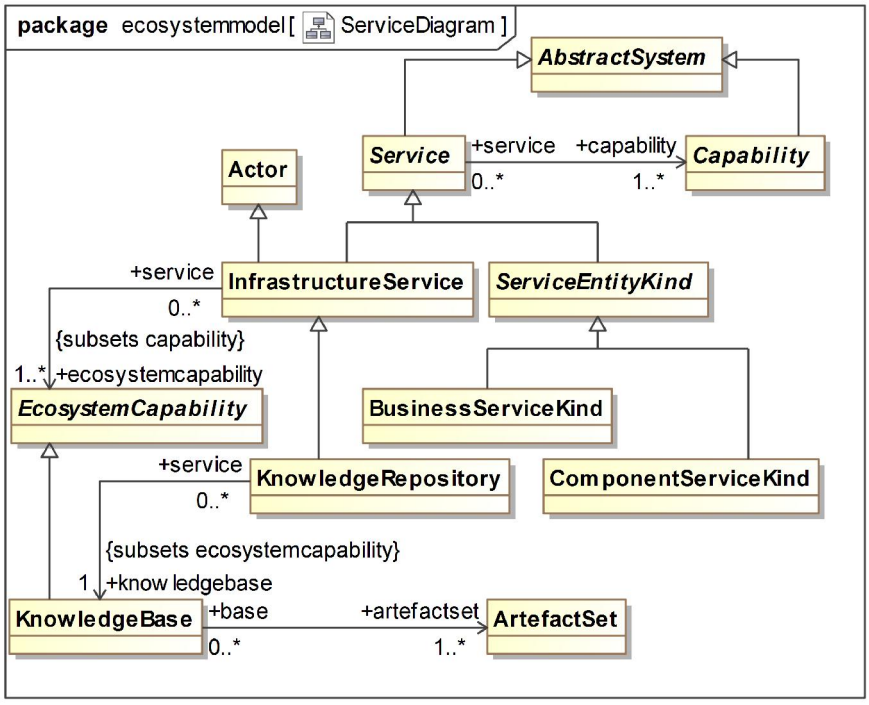

Figure 3.24: Service diagram.

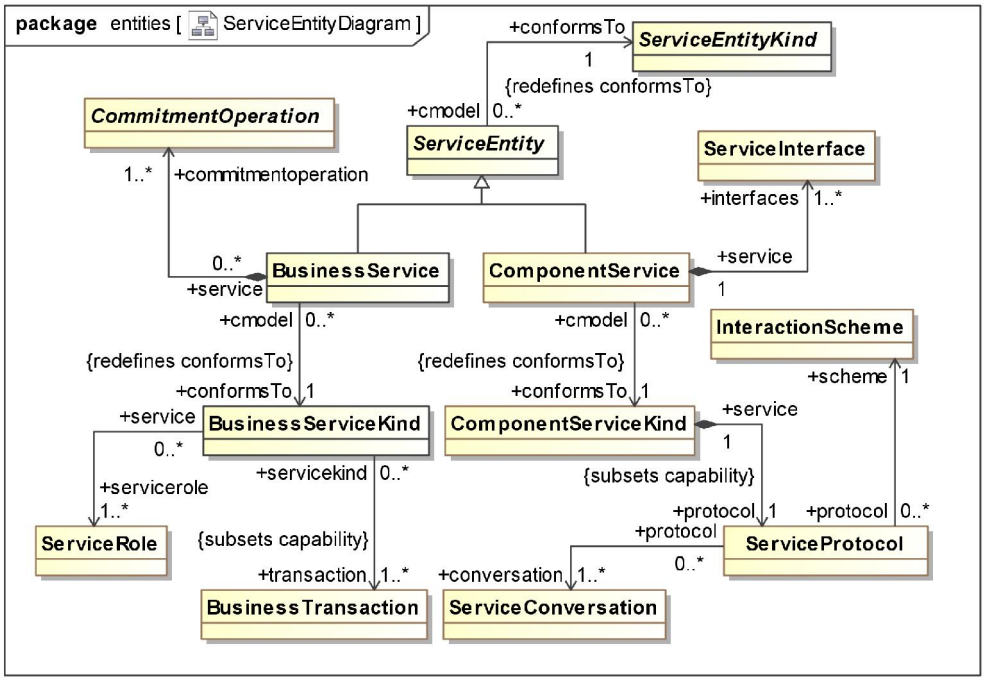

Figure 3.25: Service entity diagram.

may be provided for facilitating collaboration establishment processes in open service ecosystems.

Service entities are used in the meta-model for representing identifiable ecosystem elements that are utilizable for service provisioning and engineering. Service provisioning is enabled with the notion of *business services* whereas service engineering concerns are addressed with the notion of *component services*. The corresponding meta-model constructs are illustrated in Figure 3.25.

A business service denotes a software-supported service that is provisioned by an independent entity. A business service delivers the business transactions required to fulfill the commitments of a service choreography role. Component services are utilized in business services to implement their functionality. Component services represent reusable service engineering assets which deliver functionality defined by service protocols. A service protocol provides a definition of component service operations and behaviour. Exemplary component services may include services for domain-specific business protocol exchanges, e.g. banking protocols, or more technically oriented services such as key exchange protocols for encryption. A component service represents reusable service engineering assets which are utilized for realizing the interactions required by business transactions.

Interactions in service ecosystems are used for exchanging information be-

tween ecosystem members. Information entities in the domain ontology model are used for representing the information contents of interactions. Information entities are described with the concepts of *InformationEntityKind* and *InformationEntity*. These abstract concepts are further classified into concepts of *BusinessDocumentKind* and *BusinessDocument*, representing business document definitions and their instances, correspondingly. Standardised document description languages such as XML [172] and XML-Schema [170] can be used for definition of the structural features of an information entity kind. Semantic features for information entities can be prescribed using vocabularies such as ebXML [80] or RosettaNet [130], common taxonomies such as North American Industry Classification System (NAICS) [106] or generic ontology description languages, such as the OWL [167].

Service interactions are enabled by interaction endpoints that provide the necessary knowledge, such as identification and location information, for establishing bindings with the corresponding services. Interaction endpoints in service ecosystems are represented by the *EndpointEntityKind* and *EndpointEntity* concepts which manifest the local facilities reserved and provided by the cooperating parties for the purpose of delivering the required interactions. These concepts are further classified into service endpoints represented in the meta-model by the concepts of *ServiceEndpointKind* and *ServiceEndpoint*, as illustrated in Figure 3.22 and Figure 3.23. Service endpoint kinds provide definitions for interaction schemes, which declare the kinds of activities supported by the corresponding kinds of endpoints. The notion of *InteractionScheme*, illustrated in Figure 3.22, comprises a collection of *ServiceActivity* elements. Each *ServiceActivity* defines an activity for enabling service communication by exchange of business documents; the kind of business documents that can be exchanged is specified by an association to a *BusinessDocumentKind* element.

Service ecosystem members are represented in the domain ontology model by the concept of legal entities. Legal entities include for example organizations and individuals. A *LegalEntityKind* illustrated in Figure 3.22 represents the kinds of ecosystem members that can be obligated to deliver the required functionality and behaviour through contracts and agreements. Each legal entity kind is associated with a set of actors defined in the service ecosystem model (see Section 3.5). The set of actor definitions denotes the kinds of roles the corresponding kinds of legal entities can take in service ecosystem life cycles and service choreographies. The intensional definition for legal entity kinds comprises a set of cooperative features. Cooperative features include different collections of policies regarding legislation, business rules, operational policies and reputation management; cooperative features are discussed in Section 3.7.

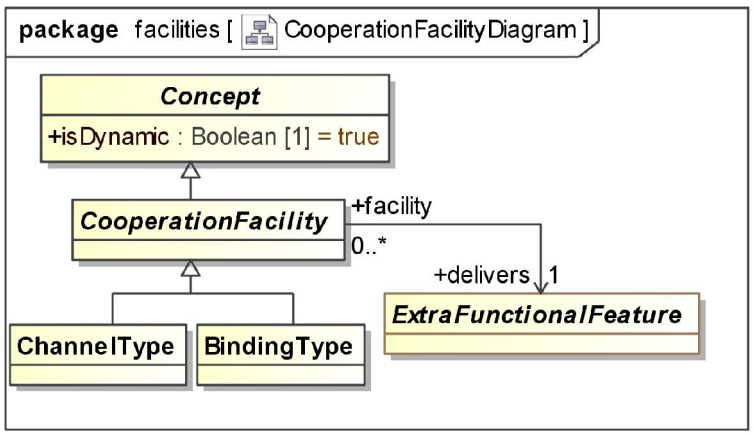

Figure 3.26: Cooperation facility diagram.

3.6.4 Cooperation facilities

Cooperation facilities provide elements for describing the abstract platform of a service ecosystem. These abstractions representations for interaction and communication which are agnostic with respect to the actual technological platforms (e.g. web services or other middleware platform) used. The categorization of cooperation facilities is illustrated in Figure 3.26. As defined in the meta-model, a cooperation facility *delivers* an extra-functional feature; this category of ecosystem features is discussed in Section 3.7. Intensions of cooperation facilities are defined by a set of *facility features*.

There are two categories of cooperation facilities: channel types and binding types. Binding types represent interaction relationships taking place between two or more service endpoints. A binding type provides an abstraction for declaring interaction characteristics, such as if interaction is to be taken in a one-to-one or one-to-many setting. Binding types provide especially an abstraction for interception mechanisms that can be utilized for adaptation (e.g. mappings in different representation formats), exogenous coordination (e.g. notifications about specific communication activities), or implementing enterprise integration patterns.

Channel types are used for declaring abstract communication media. A channel type intension comprises an ordered set of channel features. Each feature represents an individual activity that must be taken for propagating the communication payload from one interaction endpoint to another, such as encryption and decryption of the payload.

3.7 Intension model

The intension model provides support for continuation of viability in service ecosystems. Support is provided with modelling constructs that enable evolution and specialization of service ecosystem architectures. Semantic and pragmatic interoperability aspects are addressed in the intension model. Semantic interoperability is especially addressed by feature declarations utilized in the intensional definitions of ecosystem entities. Pragmatic interoperability is addressed by binding specifications and a category of bindable features. These modelling constructs are used for declaring what kinds of features are dynamically bindable, and possibly negotiable, in a service ecosystem architecture. The binding specifications let ecosystem architects explicitly declare and control the level of dynamism during service ecosystem design.

The constructs of the intension model for supporting ecosystem evolution and specialization are elaborated below. After that, the feature model subsumed by the service ecosystem intension model is described.

3.7.1 Support for ecosystem evolution and specialization

For enabling ecosystem evolution that is needed for supporting progressive business environments, the concrete intension model elements (i.e. features, feature interactions and user elements) are defined as specializations of two modelling constructs. *GenerizableElement* and *InstantiatableElement*, as illustrated in Figure 3.27, enable generalization hierarchies and deep instantiation [7] of intension model elements, correspondingly.

The service ecosystem meta-model supports natively two-level instantiation hierarchies, which are represented in the domain ontology with *EntityKind* - *Entity* pairs, and in the intension model with *Feature* - *Property* pairs. Deeper instantiation hierarchies can be defined by exploiting higher potencies for features and defining conformance relationships between *EntityKind*s and *Feature*s. Features and properties are elaborated in the next section where the feature model is described.

Service ecosystem specialization is addressed in the service ecosystem meta-model by providing means for representation of domain-specific extensions of the meta-model. The concept intensions and ecosystem features can be extended with domain-specific properties by exploiting the *ExtendableElement*-modelling element illustrated in Figure 3.28. Each *ExtendableElement* may contain several extensions represented by the element named *Extension*. An extension in practice represents a property of a concept intension or an intensional element (e.g. a feature, property or structural element as illustrated in Figure 3.27).

An extension is associated with an *ExtensionDefinition* which prescribes the

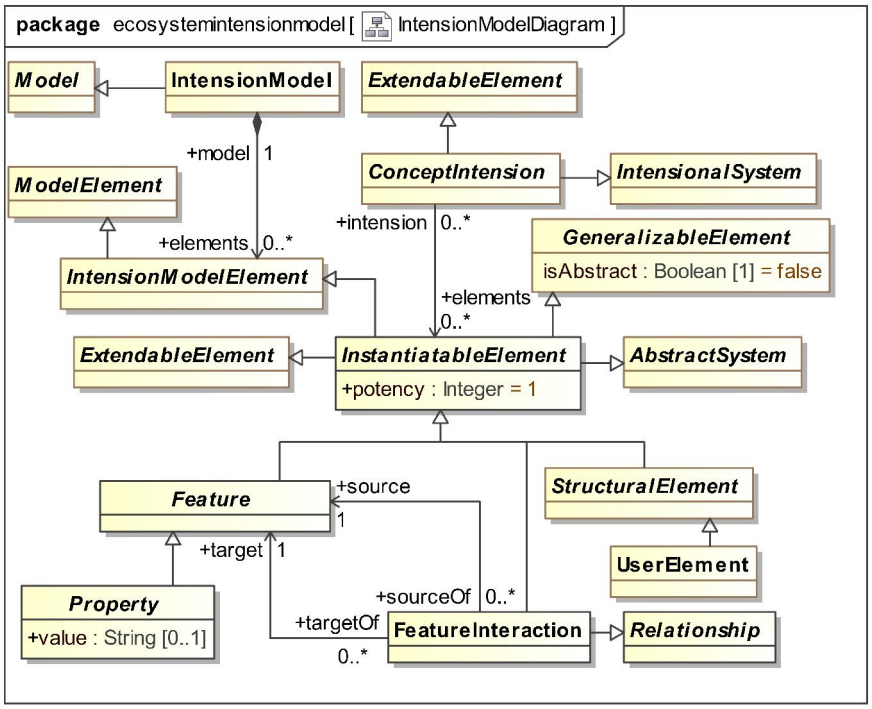

Figure 3.27: Intension model diagram.

multiplicity, containment kind, and type of the corresponding property. Extension definitions are further classified into redefinitions, sub-setting operators, or undefinitions. Redefinitions and sub-setting operators are similar to the corresponding UML [111] concepts, providing means for example to redefine or refine relationships defined in the domain ontology model. Undefinitions can be utilized for rejecting properties higher in concept intension or intension model element hierarchies. For example, a concept intension for a service kind may declare that services are not associatable with specific features in the corresponding service ecosystem.

3.7.2 Service ecosystem features

The intension model includes feature definitions that are used for defining the intensions of domain ontology concepts. Feature definitions are modelled with elements named *Feature* and *Property*, where the latter is considered as a kind of feature that conforms with another feature. Features can interact with each other; such feature interactions are domain-specific and they are described with the concept of *FeatureInteraction* illustrated in Figure 3.27. In addition, the intension model includes so-called structural elements, which allow modelling of sub-structures of features and domain-specific extensions of concepts.

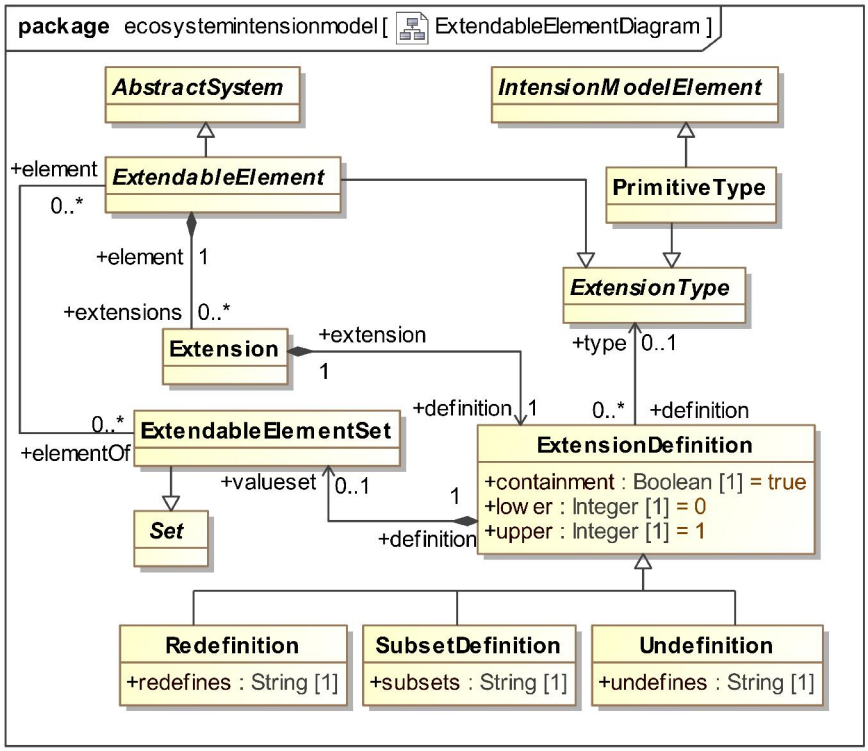

Figure 3.28: Extendable element diagram.

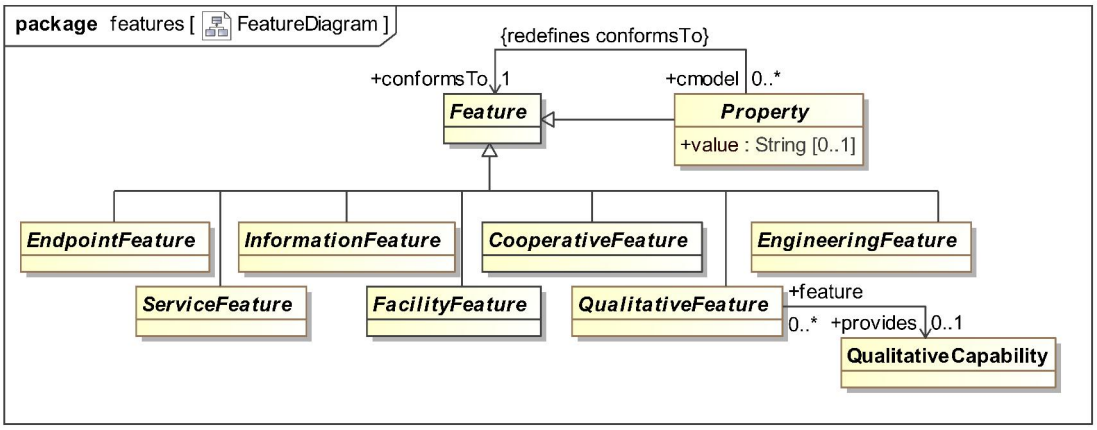

Figure 3.29: Feature diagram.

The service ecosystem meta-model subsumes a feature model which provides means for formalizing concept intensions, qualitative characteristics and engineering assets in service ecosystems. The categorization of service ecosystem features identifies seven different categories, namely *1)* endpoint features, *2)* service features, *3)* information features, *4)* facility features, *5)* cooperative features, *6)* qualitative features, and *7)* engineering features. The categorization is illustrated in Figure 3.29. The feature model is partitioned to features and properties: a *Property* represents a kind of a feature which conforms with another feature, and has a literal value associated with it. The interpretation of the value is domain-specific and depends on the semantics of the corresponding feature. A property which conforms to a feature representing temporal availability may have a value range consisting of week-days, for example.

Endpoint features are used in the intensional definitions of endpoint entities. Endpoint features declare the characteristics for a kind of a interaction endpoint. The service ecosystem meta-model defines endpoint address kinds as a specialization of endpoint features. An endpoint address kind provides a declaration of the addressing scheme used with an endpoint kind, such as URI-based addressing [5].

Service features include categories for expressing intensional definitions for business and component services. The service ecosystem meta-model does not include any specializations for business service features. However, business service features can be utilized for declaration of (business) domain-specific characteristics of business services for example with industry taxonomies such as North American Industry Classification System (NAICS) [106].

The service ecosystem meta-model includes a sub-category of component

[5]Uniform Resource Identifier: http://www.ietf.org/rfc/rfc2396.txt

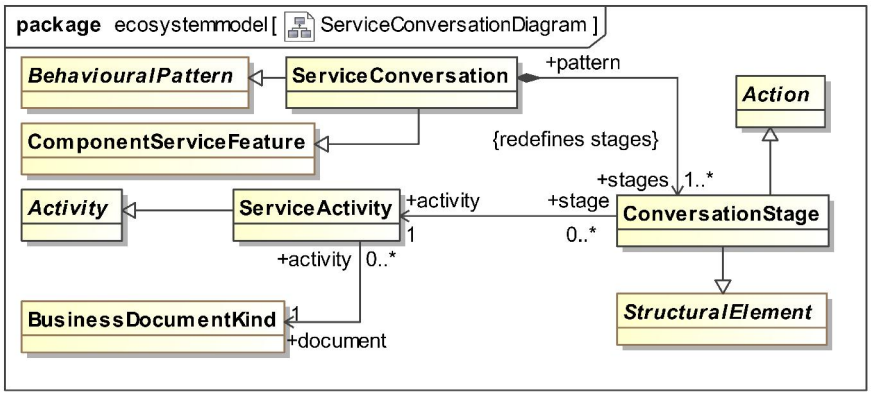

Figure 3.30: Service conversation diagram.

service features, namely service conversations. Service conversations are behavioural pattern specifications which comprise a collection of conversation stages. The *ServiceConversation* model constructs defined in the meta-model are illustrated in Figure 3.30. Each service conversation stage is associated with a service activity, which are specified in the interaction schemes of service endpoint kinds. The service conversation stages comprise a partially ordered set of actions.

Information features are used in intensional definitions of information entity kinds. The service ecosystem meta-model includes an information feature specialization named *BusinessDocumentType*, which is used for representing the types of business documents exchanged in service interactions.

Facility features define the characteristics of cooperation facilities and thus, the abstract platform. There are two categories of facility features, namely binding port types and channel features. Binding port types are used for specifying the intensions of binding types. Each binding port type represents an endpoint of an interaction relationship. A binding port type can be associated with an endpoint entity kind (e.g. a service endpoint), or another binding port type. These different associations of binding port types provide representations for typical interaction and exogenous coordination patterns, correspondingly.

The intension of a channel type is declared by an ordered set of *channel features*. The ordering is provided by the *predecessor*-association inherited from the concept of *Event*. Each channel feature declared in a channel type intension is associated with a binding port type defined in a binding type. This effectively makes the set of channel features a bipartite collection, each feature now belonging to a set associated with one of the two binding port types.

Cooperative features represent a category of service ecosystem features that regulate the activities of legal entities. Cooperative features essentially declare

constraints that are bindable to service roles. Cooperative features are further categorized into rule bases and reputation kinds. Rule bases declare different kinds of rule collections, namely business models, policy frameworks and legislation.

Business models are collections of business rules, which are declarative statements defining or constraining some aspect of a business. Policy frameworks comprise operational policies, or practices, that are characteristic for a certain kind of organization or individual. Operational policies regulate the use of business functionality and knowledge provided by a legal entity, such as an enterprise. For example rules addressing accessibility, authorization, trust and privacy with respect to the provided business services and information are typical examples of organizational policies. Legislation comprises legal acts that must be obeyed by the corresponding kind of legal entities.

Reputation kinds are utilized for describing means for evaluating the trustworthiness of legal entities acting in service roles. Different kinds of reputation models or criteria, such as recommendations or ratings, can be categorized under the concept of reputation kinds.

Cooperative features are utilized in the decision-making phase of collaboration establishment processes for evaluating the feasibility of a potential service provider. During the operation of a business network community the rules declared by cooperative features are monitored dynamically. Finally at the dissolution phase of a community the reputation of community members can be updated corresponding to the quality of their performance [85].

Cooperative features address the pragmatic interoperability issues, that is, policies and methods of decision-making on collaborations, such as risk, business value, trust and reputation. Again, there is need to define policies that are commonly understandable but dependent on all business domains involved. Cooperative properties are subject to business service owners' autonomic intentions. For cooperative properties to be truly usable within an open business service ecosystem, facilities for identity, trust and reputation management should also exist, since assertions of cooperative features can not usually be validated in advance.

The category of qualitative service ecosystem features includes contractual and extra-functional features. Contractual features represent qualitative characteristics of business service transactions. Contractual features comprise availability constraints and different charging styles, in addition to different models for settling about the service usage. In addition, contractual features include a category for coordination of business transactions. Contractual features are instantiated to contractual properties. A contractual property is a declaration of a concrete value or value constraint over some contractual feature. For example, response time can be considered as a temporal availability feature with values declared in milliseconds; now the corresponding property can be for example a declaration of the

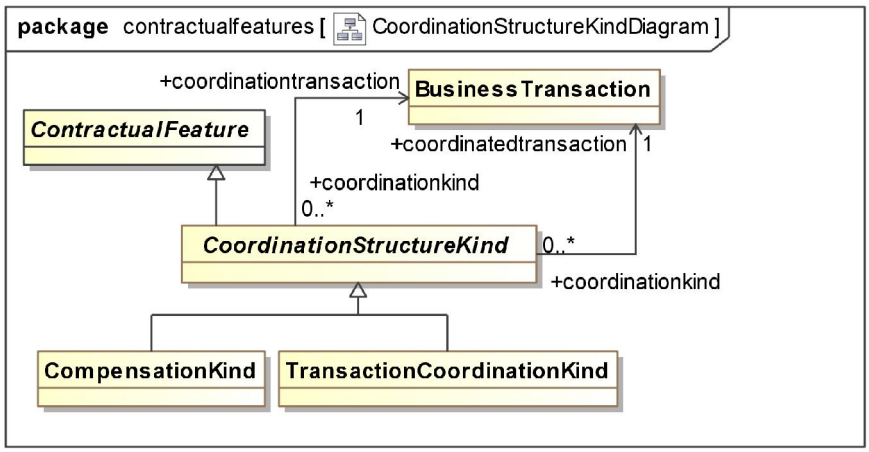

Figure 3.31: Coordination structure kind diagram.

constraint *"response time must be less than 200 ms"*.

Coordination structure kind is a category of contractual features which provides means for specification of business transaction coordination. A coordination structure kind associates a coordination transaction with a coordinated transaction, as illustrated in Figure 3.31. The coordination transaction includes additional business actions that are related temporally with the business actions of the coordination transaction. The service ecosystem meta-model defines two categories of coordination transaction kinds: *1)* compensation kinds, and *2)* transaction coordination kinds. Compensation kinds represent coordination structures with no interleaving allowed between activities in coordination and coordinated transactions. Typically an external trigger (rule) is provided in the coordination transaction to initiate the compensation transaction. Transaction coordination kinds represent coordination structures where the actions in coordination and coordinated transactions may interleave. Transaction coordination kinds can be utilized for example to introduce a notary protocol over a business transactions defined in a service choreography.

Extra-functional features represent qualitative characteristics of cooperation facilities. We identify two categories of extra-functional features: interaction features and communication features. Interaction features defines intensions of binding types and they represent interaction characteristics, such as functionality related to messaging and encoding. Communication features define intensions of channel types and represent functionality such as encryption, decryption or monitoring of behaviour. Communication features must be introduced in a certain order to be feasible, that is, they can have mutual ordering dependencies: information monitoring must be executed before encryption, for example.

Extra-functional features address semantic and technical interoperability issues relevant for managing the dependability of the underlying communication platform. These features are controllable by the service realisation provider by using the computational platform. Modifying these features requires technical administrative authority over the local communication platform, and they are closely intertwined with the computational services administered within administrative domains. Extra-functional features manifest static aspects of interaction and communication that are selectable during service binding and collaboration contract establishment.

Contractual features address especially the semantic interoperability concerns related to the qualitative characteristics of business services and operations. Contractual features are agreed upon during the negotiation phases of collaboration establishment life cycles. The features and property values that have been agreed upon negotiations are used during the operational phase of the community as monitoring criteria. If the agreed qualities are not met, compensations or other mechanisms for recovering from the contract breach can be used. Contractual features are controllable by the business service provider and modifying these features requires business administrative authority over the service. Moreover, for enabling loosely coupled and dynamic business collaborations, contractual features should be dynamically configurable in the local systems.

Finally, the service ecosystem meta-model includes a notion of engineering features utilizable for expressing the characteristics of engineering domains. There are two kinds of engineering features available: *1)* method chunks, and *2)* tool types. Method chunks are autonomous and coherent parts of methods [103]. Tool types are utilized for representing the kinds of engineering tools that are required for delivery of product parts in engineering domains. Method chunks and tool types are elaborated in Section 3.8.

The feature model for service ecosystems is summarized in Table 3.1 and Table 3.2. The tables include all the features defined in the service ecosystem meta-model in an alphabetical order. Names of abstract features (i.e. not instantiable in service ecosystem architecture models) are written in emphasis. The categorization of features is represented by division of the tables. Direct generalizations of features are defined in the second column of the table. Finally, for each feature the corresponding property is specified in the third column of the table. The property conforms to the corresponding feature given in the first column.

As an example, a *BusinessModel* is a specialization of *RuleBase* in the feature category of cooperative features; the property corresponding to a business model is represented by a conformance point. It should be noted that there are no property level elements defined for engineering or extra-functional features. Both of these preceding feature categories are strictly type-level concepts with no feasible

Feature name	Generalization	Instantiation
CooperativeFeature	*Feature*	*CooperativeProperty*
RuleBase	*CooperativeFeature*	ConformancePoint
BusinessModel	RuleBase	ConformancePoint
PolicyFramework	RuleBase	ConformancePoint
Legislation	RuleBase	ConformancePoint
ReputationKind	*CooperativeFeature*	EntityReputation
EndpointFeature	*Feature*	*EndpointProperty*
EndpointAddressKind	*EndpointFeature*	EndpointAddress
EngineeringFeature	*Feature*	N/A
ToolType	*EngineeringFeature*	N/A
MethodChunk	*EngineeringFeature*	N/A
Method	*MethodChunk*	N/A
FacilityFeature	*Feature*	*FacilityProperty*
BindingPortType	*FacilityFeature*	BindingPort
ChannelFeature	*FacilityFeature*	ChannelPhase
InformationFeature	*Feature*	*InformationProperty*
BusinessDocumentType	*InformationFeature*	BusinessDocumentElement

Table 3.1: Hierarchy of service ecosystem features and properties.

conformance relationships defined in the meta-model.

3.8 Engineering model

The engineering model included in the service ecosystem meta-model defines modelling constructs addressing especially the concerns associated with service engineering. The modelling constructs formalize assets required for supporting well-advised service engineering practices. Well-advised practices are supported by providing means for specification of methods and tools required for domain-specific service engineering, and by associating the these engineering capabilities with business- and engineering-driven product models. The engineering model constructs are used for specification of service ecosystem engineering knowledge.

Engineering models are formalized in the service ecosystem meta-model with constructs illustrated in Figure 3.32. An engineering model comprises engineering model elements, which are further categorized into engineering spaces, engineering elements and product models, product parts and product elements.

The notion of engineering space defined by the service ecosystem meta-model

Feature name	**Generalization**	**Instantiation**
QualitativeFeature	*Feature*	N/A
ContractualFeature	*QualitativeFeature*	*ContractualProperty*
AvailabilityKind	*ContractualFeature*	*Availability*
SpatialAvailabilityKind	*AvailabilityKind*	SpatialAvailability
TemporalAvailabilityKind	*AvailabilityKind*	TemporalAvailability
CoordinationStructureKind	*ContractualFeature*	*CoordinationStructure*
CompensationKind	*CoordinationStructureKind*	Compensation
TransactionCoordinationKind	*CoordinationStructureKind*	TransactionCoordination
SettlementModelKind	*ContractualFeature*	SettlementModel
ChargingStyle	*ContractualFeature*	Charging
ExtraFunctionalFeature	*QualitativeFeature*	N/A
InteractionFeature	*ExtraFunctionalFeature*	N/A
CommunicationFeature	*ExtraFunctionalFeature*	N/A
ServiceFeature	*Feature*	*ServiceProperty*
BusinessServiceFeature	*ServiceFeature*	BusinessServiceProperty
ComponentServiceFeature	*ServiceFeature*	ComponentServiceProperty
ServiceConversation	*ComponentServiceFeature*	MessageExchange

Table 3.2: Hierarchy of service ecosystem features and properties (continued: qualitative and service features).

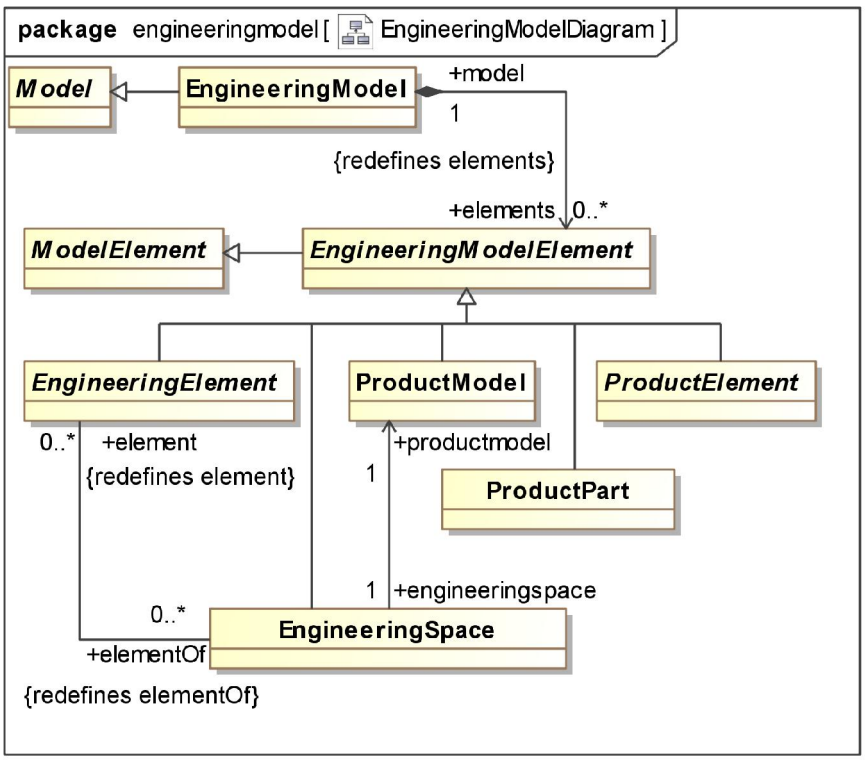

Figure 3.32: Engineering model diagram.

aligns with the concept of technical space. A technical space is *"a working context with a set of associated concepts, body of knowledge, tools, required skills and possibilities"* [83]. All engineering domains are associated with a corresponding technical space comprising domain-specific best practices and tools that are applied for the engineering activities. Correspondingly, each service ecosystem domain is associated with an engineering space. An engineering space is formalized in the meta-model by the modelling construct of *EngineeringSpace*, as illustrated in Figure 3.32. It comprises a collection of engineering elements and is associated with a product model.

Specializations of engineering elements include modelling constructs for specification of identifiers and tool types, methods and their parts, and binding models. The corresponding class hierarchy is illustrated in Figure 3.33. Identifiers are used in service ecosystem domains for resolution of service ecosystem elements, such as entities and their features. An *IdentifierType* can be utilized for specification types of identifiers, such as URI:s or subsets of thereof. An *Identifier* is considered as a simple string-valued element, which conforms to an identifier type. The conformance rules with respect to syntax and semantics of identifiers and their types are domain-dependent.

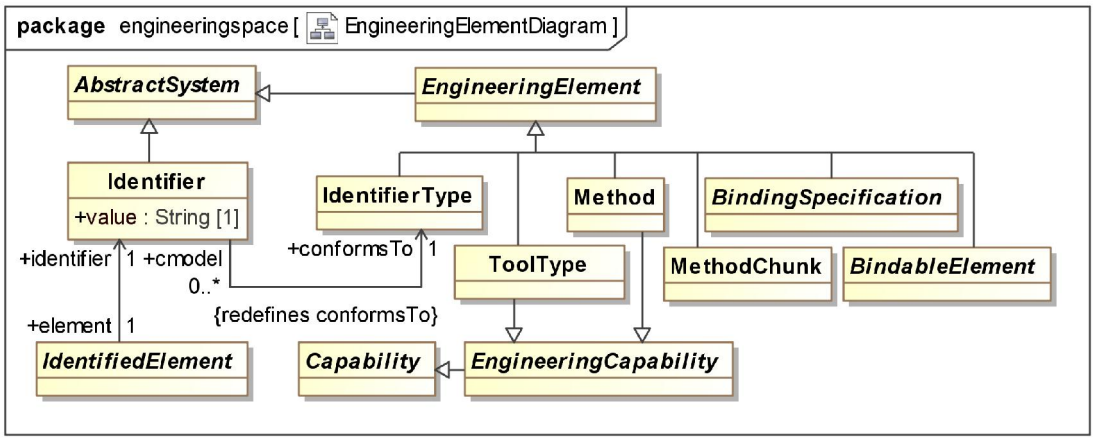

Figure 3.33: Engineering element diagram.

Engineering capabilities are used in the conceptual model for service ecosystem to denote engineering abilities available for ecosystem domains. They are formalized in the service ecosystem meta-model as abstract systems and specialization of the capability element, as illustrated in Figure 3.34. There are two kinds of engineering capabilities: tool types and methods. A tool type represents a capability to enact and take part in a coherent set of engineering activities. *ToolType* is formalized as a collection of *EngineeringActivity*-elements (see Figure 3.34) where each engineering activity is defined as a kind of activity that produces a product element based on an optional set of product elements consumed as input.

Engineering methods are prescribed using a situational method engineering framework based on the notion of method chunks [103]. A method chunk is a reusable engineering asset utilizable for constructing engineering methods. A method chunk is an autonomous and coherent part of a method supporting realization of engineering activities [103]. As illustrated in Figure 3.34, a *Method* is composed of one or more *MethodChunk* elements.

The service ecosystem meta-model formalizes engineering methods and method chunks with model elements illustrated in Figure 3.35. The constructs are based on the situational method engineering approach introduced in [103]. However, some alternations of the original concepts have been made. Especially, in the service ecosystem meta-model the guidelines are classified to simple and production guidelines. The class of production guidelines includes tactical guidelines of the original work [103], and a new class of engineering guidelines. Engineering guidelines are associated with engineering activities, as illustrated in Figure 3.35.

A *MethodChunk* represents a reusable, coherent part of an engineering method. Each method chunk comprises a single guideline and is associated with a product

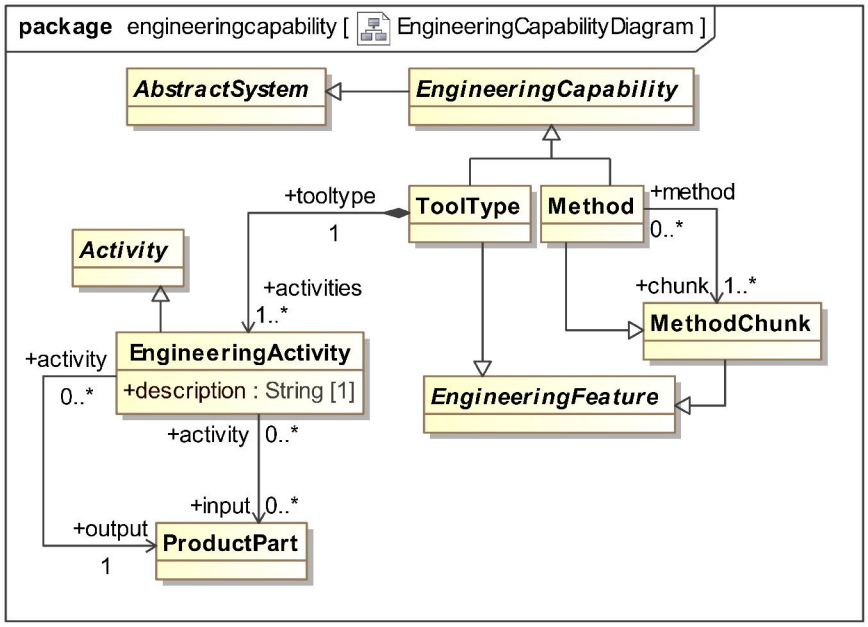

Figure 3.34: Engineering capability diagram.

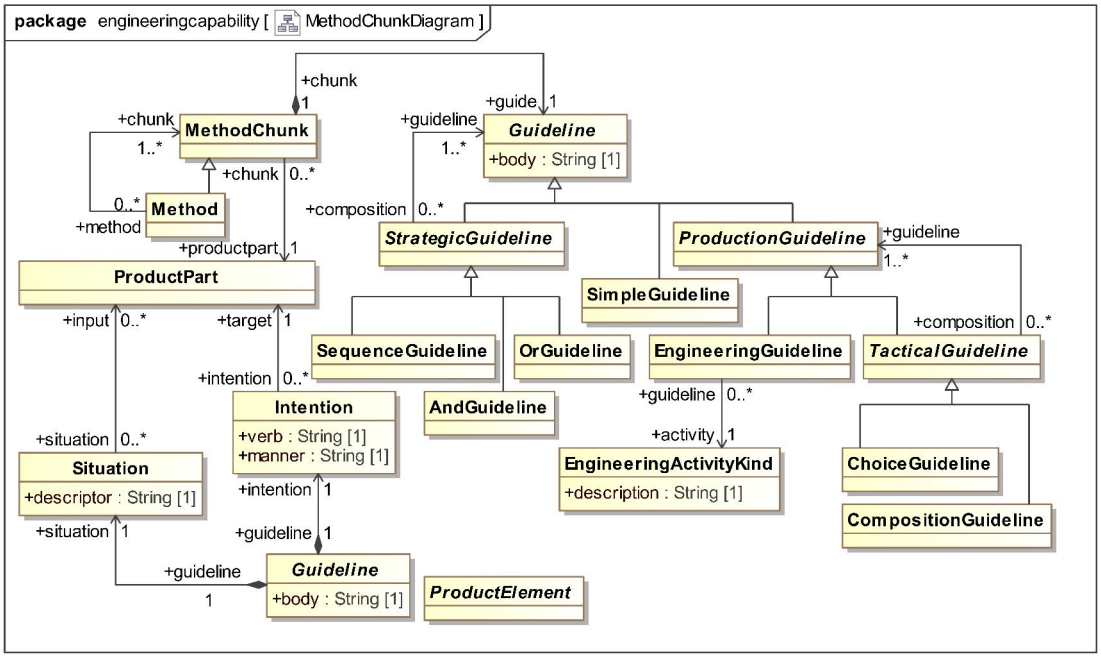

Figure 3.35: Method chunk diagram.

part. The associated product part represents the outcome of a successful enactment of the method chunk.

A *Guideline* represents a specification of a method fragment. There are two kinds of guidelines defined in the service ecosystem meta-model: strategic and tactical guidelines. A *StrategicGuideline* provides a strategic view of the engineering process by telling which intention can be achieved following which strategy [103]. Strategic guidelines are further classified into sequences and alternatives (*AndGuideline* and *OrGuidelinne*). Production guidelines are classified to tactical guidelines and engineering guidelines. A tactical guideline comprises a complex, tree-structured set of production guidelines [103]. Tactical guidelines are classified into choices and compositions (sequences). An *EngineeringGuideline* provides means for specification engineering instructions associated with a kind of an engineering activity. Finally, a *SimpleGuideline* represents a simple declaration of methodological advice, which can be utilized in strategic guidelines.

Each guideline is associated with specifications for a reuse situation and an intention. The reuse situation is formalized by the element named *Situation*, as illustrated in Figure 3.35. A reuse situation specification provides a description when the corresponding guideline is applicable, and identifies the reuse context as an association to the appropriate product part. The intention of a guideline is represented by the element named *Intention*. The intention defines what has to be done (verb) and how it should be enacted (manner).

Product models are utilized in situational method engineering approaches [103] for representation of the engineering artefacts. The service ecosystem meta-model formalizes product models with modelling constructs that are illustrated in Figure 3.36. A *ProductModel* is considered as a set consisting of product parts. Each *ProductPart* represents a coherent set of product elements which is applicable for utilization in engineering methods. Product elements are presented with the modelling construct named *ProductElement*; product elements include features, structural elements, and feature interactions. Typically an intensional definition of an ecosystem concept would constitute a single product part.

The service ecosystem meta-model supports control over the dynamism of service ecosystems, and specification of abstract service platforms with so-called binding models. The notion of abstract service platform is similar to the more generic notion of abstract platforms described in [2]. An abstract platform defines an ideal platform from an application developers' point of view and declares the platform's characteristics [2]. Similarly, the abstract service platform defines the characteristics of the service ecosystem platform especially from the viewpoint of stakeholders associated with service delivery (i.e. service providers, service consumers, and service engineers). Binding models declare the rules for encapsu-

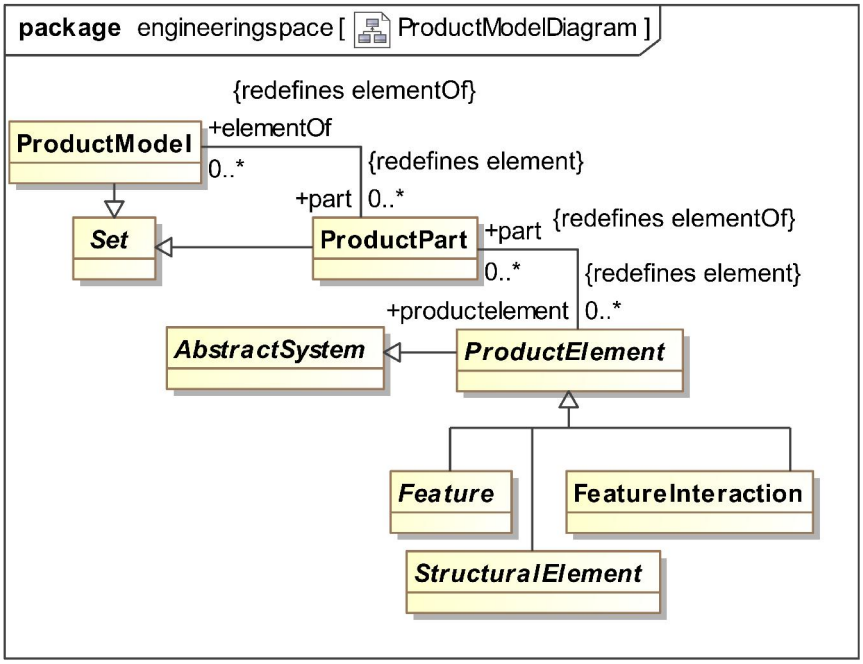

Figure 3.36: Product model diagram.

lation of services interactions with bindable features: binding specifications prescribe what kinds of features can be bound to service roles, business transactions or service connections, for example.

The binding models are specified as part of the engineering models discussed previously. Binding models comprise bindable elements and binding specifications, both of which are specializations of *EngineeringElements* illustrated in Figure 3.33. Binding elements are classified into bindable features and bindable properties. The category of bindable features includes cooperative and qualitative features that were discussed in Section 3.7. Correspondingly, bindable properties include cooperative and contractual properties.

Binding specifications are formalized in the service ecosystem meta-model with constructs that are illustrated in Figure 3.37. Binding specifications are further classified into feature bindings and property bindings. A *FeatureBinding* is formalized as a modelling construct which associates a feature binding target with a set of bindable features. Correspondingly, a *PropertyBinding* associates a property binding target with a set of bindable properties.

Feature and property binding specifications are elaborated in Table 3.3. The first column of the table specifies the names for the binding specifications, the second column denotes the binding targets, and third column the category of bindable features or properties. The tables are divisioned with respect to the binding

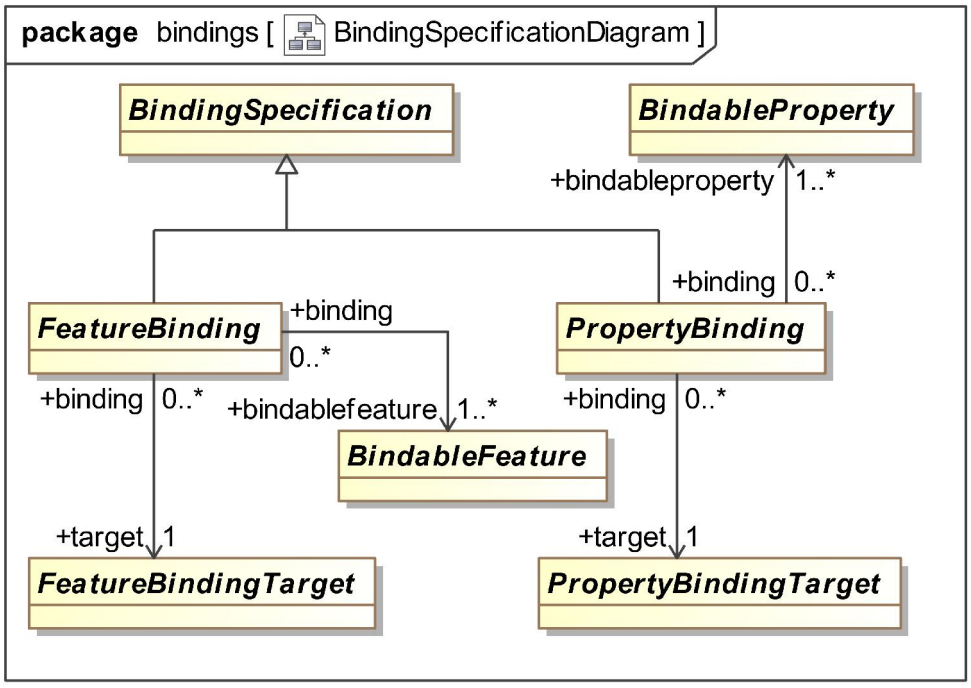

Figure 3.37: Binding specification diagram.

specification constructs. Emphasized terms denote abstract classes in the service ecosystem meta-model. Indentation is utilized in the binding target columns for expressing generalization hierarchies; e.g. *BusinessServiceKind* is a specialization of the *ContractualFeatureTarget* class in the meta-model. The first three rows in Table 3.3 describe binding specifications that are specializations of the *FeatureBinding* class illustrated in Figure 3.37. The latter three rows correspond to specializations of the *PropertyBinding* class.

3.9 Knowledge management model

Especially in open service ecosystems, a coherent knowledge management infrastructure is required for enabling interoperation of dynamically formed, loosely coupled service collaborations. Consequently, the service ecosystem meta-model defines constructs for instrumentation of service ecosystems with domain-specific knowledge management infrastructure. The constructs are defined in the knowledge management model whose main elements are illustrated in Figure 3.38. The knowledge management model comprises a collection of knowledge management elements. Knowledge management elements are further classified into knowledge repositories, knowledge item types and knowledge relationship types.

Binding name	Binding target	Bindable feature
ContractualFeatureBinding	*ContractualFeatureTarget*	*ContractualFeature*
	BusinessServiceKind	*ContractualFeature*
	BusinessTransaction	*ContractualFeature*
CooperativeFeatureBinding	*CooperativeFeatureTarget*	*CooperativeFeature*
	ServiceRole	*CooperativeFeature*
ExtraFunctionalFeatureBinding	*ExtraFunctionalFeatureTarget*	*ExtraFunctionalFeature*
	ServiceConnection	*ExtraFunctionalFeature*
ContractualPropertyBinding	*ContractualPropertyTarget*	*ContractualProperty*
	BusinessService	*ContractualProperty*
CooperativePropertyBinding	*CooperativePropertyTarget*	*CooperativeProperty*
	LegalEntity	*CooperativeProperty*

Table 3.3: Binding specifications in service ecosystem meta-model.

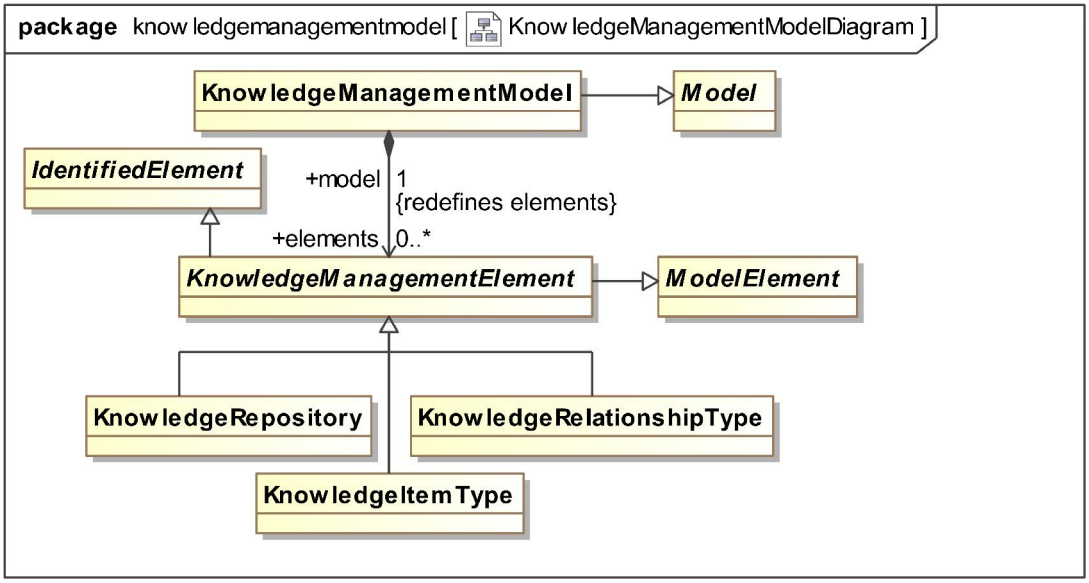

Figure 3.38: Knowledge management model diagram.

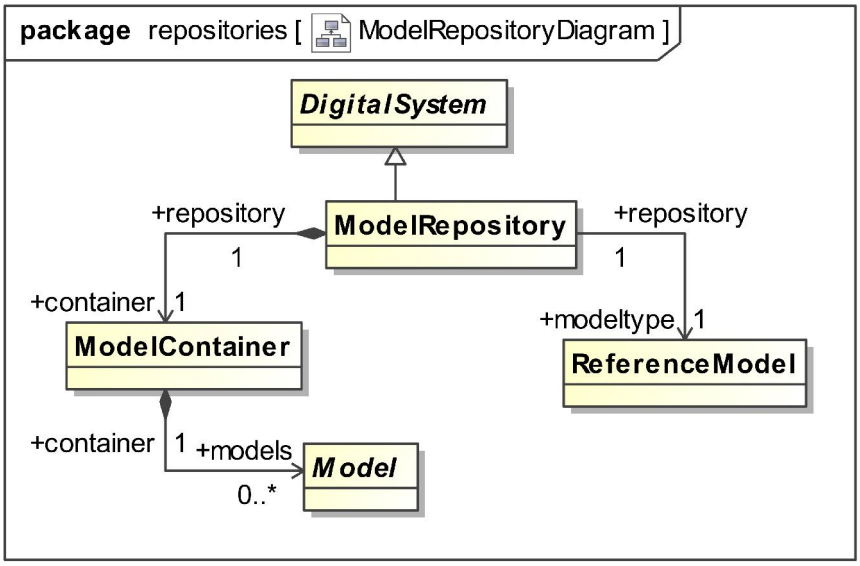

Figure 3.39: Model repository diagram.

A knowledge repository is considered in the service ecosystem meta-model as a kind of a model repository. A model repository is defined as a digital system which comprises a model container and is associated with a reference model, as illustrated in Figure 3.39. A *ModelContainer* is simply a collection of *Model*s. Each model included in a container of a model repository must conform to the reference model associated with the repository. That is, the model repository, when implemented, is responsible for validation of the conformance relationship between the models in the container and the associated reference model.

Knowledge repositories provide the backbone for facilitating sustainable service ecosystems. Knowledge repositories are considered as infrastructure services that maintain the semantics of ecosystem knowledge bases. The service ecosystem meta-model formalizes the notion of knowledge repositories with the modelling constructs that are illustrated in Figure 3.40. Knowledge repository is associated with a knowledge item type which is a representation of a product part. In a typical service ecosystem each intensional definition of a concept is provided with a product part, as well as the bindable features and properties are included in appropriate product parts. Consequently, management of these service ecosystem features shall be handled by corresponding knowledge repositories.

In addition to providing means for sharing knowledge about product parts, the knowledge repositories are responsible for maintaining relationships associated with the respective product parts. These relationships are formalized in the knowledge management model with the notion of *KnowledgeRelationshipType*.

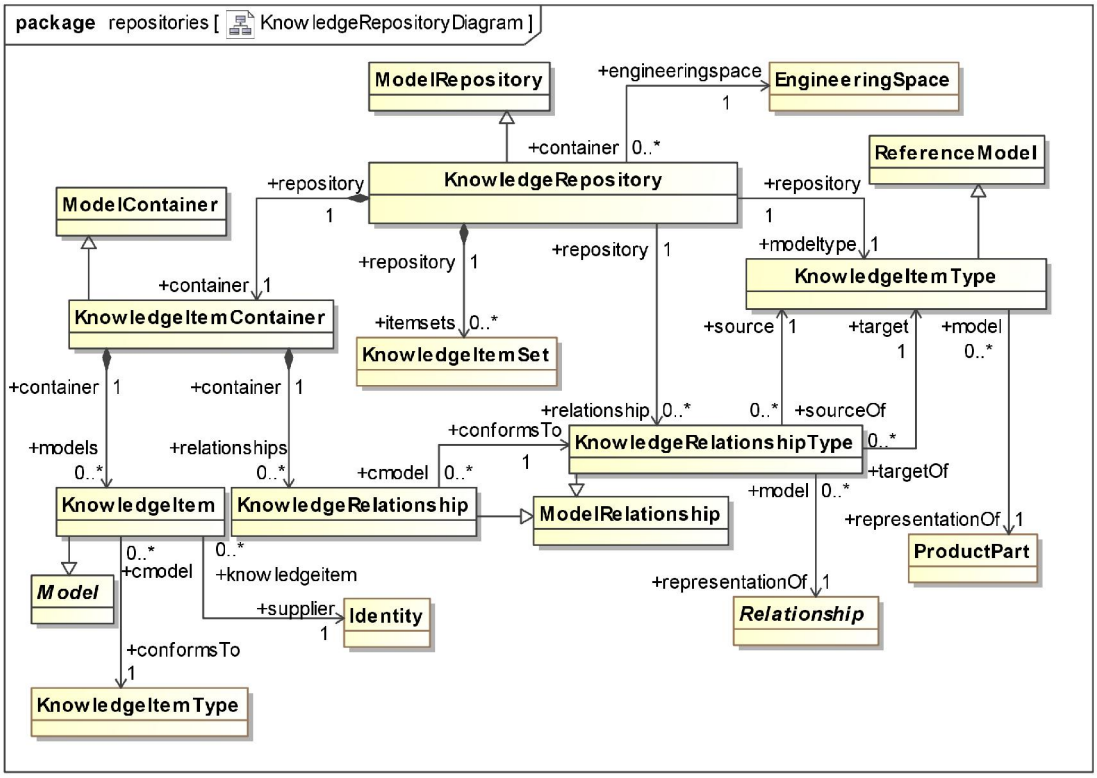

Figure 3.40: Knowledge repository diagram.

A knowledge relationship type is considered as a binary model relationship between knowledge item types. A knowledge relationship type is a representation of some other relationship. There are in essence two kinds of relationships that should be formalized with knowledge relationship types: concept relationships defined in domain ontologies and feature interactions defined in intensional models. A knowledge repository should provide means for maintaining these kinds of relationships that are associated with the corresponding intensional definitions and feature definitions.

A *KnowledgeItemContainer* maintains a set of knowledge items and knowledge relationships. Knowledge items must conform to the knowledge item type, which is considered as a reference model, of the knowledge repository. The suppliers of knowledge items are identified with a reference to their identity. A *KnowledgeRelationship* is a binary model relationship between knowledge items which conforms to a knowledge relationship type maintained by the corresponding knowledge repository.

While all knowledge items are included in the knowledge item container, an additional grouping mechanism over the knowledge items is defined as a collection of knowledge item sets. A *KnowledgeItemSet* is defined as a set which includes knowledge items. Each knowledge item set is a representation of another set. Especially, for each artefact set defined in the service ecosystem model a knowledge item set must be provided. The knowledge item sets are utilized for maintaining the different roles of knowledge items in different phases of service ecosystem life cycles.

The semantics of service ecosystem artefacts are enforced in the knowledge item sets of the knowledge repositories. For example, the Pilarcos service offer repository [85] is a knowledge repository which maintains knowledge items conforming with the intensional definition of service offers. Correspondingly, the Pilarcos service offer repository includes knowledge item sets representing artefact sets for available services and interoperable services [143].

Chapter 4

Tools for model-driven service ecosystem engineering

This chapter defines tools that enable rigorous analysis and design of service ecosystems, and efficient service ecosystem instrumentation. The tools provide implementations for an architecture framework and a model-driven methodology for service ecosystem engineering. The architeture framework can be utilized by domain experts for the analysis and design of service ecosystems, as was illustrated in Figure 1.1. The model-driven methodology for service ecosystem engineering can especially be used for infrastructure development, as was discussed in the home-automation service ecosystem example.

The Service Ecosystem Architecture Framework (SEAF) defined in this chapter facilitates rigorous service ecosystem engineering by providing means for service ecosystem analysis and design based on explicit and formal architecture models. Explicit service ecosystem architecture models are needed for enabling efficient architecture analysis and design practices, assessment of architectural properties with respect to design principles, and traceability of design decisions, for example. The applicability of the architecture framework is evaluated with a case study in Chapter 5.

Sustainability of service ecosystems requires efficient means for instrumentation of domain-specific service ecosystems. Such means are provided by the model-driven methodology for service ecosystem engineering developed in this chapter. The methodology is realized with a series of model transformations that refine and convert service ecosystem architecture models to more detailed and technology-specific models. The methodology is implemented over the Eclipse development platform [38]; however, the model-driven approach of the methodology is technology agnostic.

SEAF is defined in Section 4.1. First an overview of the architecture framework is given. The overview is followed with specification of the viewpoints in-

cluded in the architecture framework. The model-driven methodology for service ecosystem engineering is described in Section 4.2.

4.1 The Service Ecosystem Architecture Framework

Architectural analysis and design of complex systems can be facilitated with use of architecture frameworks. An architecture framework *"establishes a common practice for creating, interpreting, analyzing and using architecture descriptions within a particular domain of application or stakeholder community"* [72]. The domain of application in this context is that of service ecosystems with the stakeholder community as identified in Chapter 2.

An architecture framework consists essentially of architecture viewpoints and correspondence rules between the viewpoints, as well as instructions, best practices and guidelines for architecture work [72]. More specifically, an architecture framework must provide: *1)* identification of one or more concerns with respect to the system of interest, *2)* identification of one or more stakeholders having those concerns, *3)* one or more architecture viewpoints that frame those concerns, and *4)* any correspondence rules needed for ensuring the consistency of architectural descriptions [72].

In this section the Service Ecosystem Architecture Framework is defined. First an overview of the architecture framework is given in Section 4.1.1. The overview briefly characterizes architecture viewpoints and correspondences between them. The architecture framework for service ecosystems comprises several viewpoint specifications that are defined in Section 4.1.2. The viewpoint specifications prescribe the kinds of models, tools and practices required for describing service ecosystem architectures. The models and tools are based on the conceptual model and the service ecosystem meta-model defined in the previous chapters. The UML modelling language and its extension mechanism, that is, UML profiles, are utilized for implementing the modelling notations required by the viewpoints.

4.1.1 Elements of the architecture framework

SEAF provides means for creating service ecosystem architecture descriptions, and for utilizing the descriptions for instrumentation of sustainable service ecosystems. The architecture framework defines seven viewpoints for specifying service ecosystems: *1)* ecosystem capability viewpoint, *2)* service choreography viewpoint, *3)* life-cycle viewpoint, *4)* domain-ontology viewpoint, *5)* concept intension viewpoint, *6)* engineering viewpoint, and *7)* knowledge management viewpoint.

The ecosystem capability viewpoint frames the concerns of service ecosystem purpose, scope and structure. The primary concepts used in this viewpoint

are ecosystem capabilities, ecosystem domains, life cycles and infrastructure services. Ecosystem capabilities define the purpose of a service ecosystem. Ecosystem domains provide means for structuring service ecosystems with different scopes of authorization, visibility or social relationships (e.g. trust), for example. Service ecosystem capabilities are realized with ecosystem life cycles, infrastructure services and qualitative features, as discussed in Section 2.1.

The service choreography viewpoint frames the concern of ecosystem behaviour. The primary concepts used in the service choreography viewpoint include service choreographies and business transactions. Service choreographies specify service roles, their connections and business actions exchanged between participants acting in the roles. A business transaction defines a coherent set of business actions which achieves some business objective in the corresponding domain of interest.

The life-cycle viewpoint frames the concerns of ecosystem responsibilities, knowledge propagation, and behaviour. The life-cycle viewpoint addresses especially pragmatic interoperability [6] concerns during ecosystem operation by prescribing the behaviour expected from ecosystem members. Service ecosystem responsibilities are declared by specifying domain-specific actors and their participation in ecosystem life cycles. Knowledge propagation between life cycles, their separate phases, and actors are declared by specification of artefacts and their inter-dependencies with the previous elements. Finally, ecosystem behaviour is declared by defining temporal dependencies between life-cycle phases.

The domain-ontology viewpoint frames the concern of establishing an explicit, domain-specific, shared conceptualization. The shared conceptualization addresses especially semantic interoperability concerns, and sharing of interoperability knowledge during ecosystem operation. The shared conceptualization is specified by declaring concepts, concept sets and their inter-relationships. The conceptual framework and the corresponding formalization as a domain ontology model provides means for the definition of these domain ontologies: they define top-level concepts such as entities, features and choreographies, which can be extended in domain-specific ontologies (e.g. an ontology for home automation service ecosystems).

The concept intension viewpoint frames the concerns of sharing engineering knowledge, enabling domain-specific modelling practices, and tool interoperability. While the domain-ontology viewpoint is utilized for prescribing concepts and their mutual relationships, the intensional viewpoint defines the meaning of the concepts by declaring their structural properties. For example, the intensional definition for a service endpoint may declare that the corresponding endpoints support REST-like [56] activities.

The engineering viewpoint frames the concern of enabling well-advised and

efficient service engineering in service ecosystems. Well-advised and efficient service engineering is supported by prescribing reusable engineering assets and following a situational method engineering [103] approach. Engineering assets include definitions for product models and binding models. Product models prescribe artefacts and their compositions appropriate to be utilized in service engineering methods. Typically the decompositions of artefacts would follow the structure of knowledge items, but other kinds of structuring allowing more efficient engineering efforts can also be utilized. Methods are composed of method chunks which are autonomous and coherent parts of methods [103]. Binding models declare the rules for late encapsulation of services with bindable features: binding specifications prescribe what kinds of features can be bound to service roles, business transactions or service connections, for example.

The knowledge management viewpoint frames the concerns of enabling ecosystem knowledge management and establishing coherent service ecosystem knowledge bases. Service ecosystem knowledge management is enabled by specifying knowledge repositories responsible for maintaining sets of concepts. The primary concepts addressed in the knowledge management viewpoint are knowledge bases, knowledge repositories, knowledge items, and knowledge item relationships.

4.1.2 Viewpoint specifications

In the following, the previously described viewpoints used are formally specified. The specifications follow the conventions laid in the ISO 42010 standard [72]. Following the conventions, an architecture viewpoint shall specify

1. one or more concerns framed by the viewpoint,

2. typical stakeholders for concerns framed by the viewpoint,

3. one or more model kinds used in the viewpoint, and

4. for each model kind, the languages, notations, conventions, modelling techniques, analytical methods and/or other operations to be used for models of this kind [72].

SEAF is implemented over a UML modelling tool. Correspondingly, the model kinds (i.e. different modelling notations used in viewpoints) are implemented using the profiling mechanism of the UML standard [113]. The UML stereotypes providing modelling notations for the model kinds are described as part of the viewpoint specifications.

Ecosystem capability viewpoint

The ecosystem capability viewpoint enables description of service ecosystem purpose, scope. The elements of the conceptual framework referred to in this viewpoint are ecosystem capabilities, ecosystem domains, life cycles and infrastructure services.

Typical stakeholders for this viewpoint include ecosystem providers and infrastructure providers. Ecosystem providers utilize the capability viewpoint for design and analysis of service ecosystems. Infrastructure providers utilize the capability viewpoint for identifying functional and non-functional requirements of infrastructure services by analysis of the expected ecosystem capabilities and life cycles. The ecosystem capability viewpoint is summarized in Table 4.1.

Framed concerns	Ecosystem purpose, scope and structure
Typical stakeholders	Ecosystem provider; Infrastructure provider
Concepts	Ecosystem capabilities, domains, life cycles and infrastructure services.
Model kinds	Ecosystem capability specification

Table 4.1: Ecosystem capability viewpoint.

The ecosystem capability viewpoint is implemented using a model kind named Ecosystem Capability specification. The Ecosystem Capability specification model kind is implemented over UML Structure Diagram notation [111]. The stereotype definitions, their correspondences with the service ecosystem meta-model elements, and associated constraints are defined in Table 4.2.

Service choreography viewpoint

The service choreography viewpoint addresses the concern of service ecosystem behaviour. This viewpoint is utilized for defining the kinds of service roles, business actions and business transactions available in the service ecosystem.

Typical stakeholders associated with the service choreography viewpoint include ecosystem providers, service providers and service engineers. Ecosystem providers use the viewpoint to declare the kinds of business transactions supported in the ecosystem. Service providers and service engineers utilize the service choreography viewpoint for directing business service design to conform with the business transactions accepted by a service ecosystem. The service choreography viewpoint is summarized in Table 4.3.

The service choreography viewpoint is implemented using four different model

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
Capability	s_capability		Element	
EcosystemCapability	s_ecosystemcapability	s_capability	Interface	Owner is an ecosystem model or domain.
EcosystemDomain	s_ecosystemdomain		Package	Owner is an ecosystem model or domain.
EcosystemModel	s_ecosystemmodel		Package	
FunctionalCapability	s_functionalcapability	s_ecosystemcapability	Interface	
InfrastructureServiceKind	s_infrastructureservicekind		Actor	
LifeCycle	s_lifecycle		Component	
QualitativeCapability	s_qualitativecapability	s_ecosystemcapability	Interface	
N/A	capability_uses_capability		Dependency	Source and target must be similar kinds of capabilities (i.e. functional or qualitative).
N/A	ecosystem_domain_provides		Dependency	From domain to capability.
N/A	ecosystem_domain_requires		Dependency	From domain to capability.
N/A	infrastructureservicekind-_realizes_capability		Association	From infrastructure service to capability.
N/A	s_lifecycle_realizes_capability		InterfaceRealization	From lifecycle to capability.

Table 4.2: Stereotypes for Ecosystem Capability specification (UML notation: Structure (Component) Diagram)

Framed concerns	Ecosystem behaviour
Typical stakeholders	Ecosystem provider; Service provider; Service engineer
Concepts	Service choreographies, roles and service connections; Business actions and business transactions.
Model kinds	Service Choreography Structure specification; Static / Dynamic Choreography Behaviour specification; Business Transaction specification.

Table 4.3: Service choreography viewpoint.

kinds: *1)* Service Choreography Structure specification, *2)* Static Choreography Behaviour Specification, *3*) Dynamic Choreography Behaviour Specification, and *4)* Business Transaction Specification.

Service Choreography Structure specification is implemented using the UML Composite Structure Diagram notation extended with the stereotypes defined in Table 4.4.

The Static Choreography Behaviour specification is implemented with UML Sequence Diagram notation which is extended with the stereotypes defined in Table 4.5.

The Dynamic Service Choreography Behaviour specification is implemented with UML Class Diagram notation which is extended with the stereotypes defined in Table 4.6.

The Business Transaction specification is utilized for defining the kinds of business transactions acceptable in a service ecosystem. This model kind is implemented over UML Class Diagram notation with the stereotypes characterized in Table 4.7. Each business transaction is modelled as a UML Interface extended with the *s_businesstransaction_interface*; the *action* tagged value defines the set of business actions included in the corresponding business transaction.

Service interactions define acceptable message exchanges for component services. Static service interactions are modelled using two model kinds: *1)* Service Interaction specification, and *2)* Service Conversation specification. The first model kind is used for modelling interaction schemes and service protocols utilizing the schemes. The second model kind is used for modelling service conversations of the service protocols.

The Service Interaction specification is implemented with UML Class Diagram notation with the stereotypes described in Table 4.8.

Service conversation specifications are used for modelling service conversa-

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
ServiceChoreography	*s_servicechoreography*		Element	
	s_servicechoreography _class	s_servicechoreography	Class	
ServiceConnection	*s_serviceconnection*		Element	Owner must be of kind *s_servicechoreography.*
	s_serviceconnection _connector	s_serviceconnection	Connector	Owner must be of kind *s_servicechoreography _class.*
ServiceRole	*s_servicerole*		Element	
	s_servicerole_property	s_servicerole	Property	*UML::Property::type* must refer to a model element of kind *s_actor.*

Table 4.4: Stereotypes for Service Choreography Structure specification (UML notation: Composite Structure Diagram)

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
BusinessAction	*s_businessaction*		Element	
	s_businessaction_message	s_businessaction	Message	*messageSort* is either *asynchCall* or *synchcall*; The optional *Message::argument* property must refer to an model element of kind *s_lifecycleproduct*.
	s_servicechoreography _interaction		Interaction	The owner must be a model element of kind *s_servicechoreography*.
ServiceRole	*s_servicerole*		Element	
	s_servicerole _lifeline	s_servicerole	Lifeline	*UML::Lifeline::represents* must refer to a model element of kind *s_servicerole_property*.

Table 4.5: Stereotypes for Static Choreography Behaviour specification (UML notation: Sequence Diagram)

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
BusinessAction	*s_businessaction*		Element	
	s_businessaction_class	s_businessaction	Class	Tagged value *commitment* must refer to a model element of kind *s_servicecommitment*. Tagged value *transaction* must refer to a model element of kind *s_businesstransaction*.
ServiceChoreography	*s_servicechoreography*		Element	
	s_servicechoreography _class	s_servicechoreography	Class	The element must own at least one model element of kind *s_businessaction_class* and *s_businesstransaction*.
ServiceCommitment	*s_servicecommitment*		Element	
	s_servicecommitment _class	s_servicecommitment	Class	Tagged value *connection* must refer to a model element of kind *s_serviceconnection*. Tagged values *creditor* and *debtor* must refer to a model element of kind *s_servicerole*. Tagged value *product* must refer to a model element of kind *s_lifecycleproduct*.

Table 4.6: Stereotypes for Dynamic Service Choreography specification (UML notation: Class Diagram)

Meta-model element name	Stereotype name	Parent stereotype	UML metaclass	Constraints
BusinessTransaction	s_businesstransaction		Element	
	s_businesstransaction_interface	s_businesstransaction	Interface	Owner must be a model element stereotyped with *s_servicechoreography*. Tagged value *action* refers to one or more model elements of kind *s_businessaction* defining the business actions included in the business transactions.

Table 4.7: Stereotypes for Business Transaction specification (UML notation: Class Diagram)

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
InteractionScheme	*s_interactionscheme*		Element	Owner must be a model element of kind *s_serviceendpointkind.*
	s_interactionscheme _class	s_interactionscheme	Class	Tagged value *activities* must refer to a model element of kind *s_serviceactivity.*
ServiceActivity	*s_serviceactivity*		Element	The owner must be a model element of kind *s_interactionscheme.*
	s_serviceactivity _class	s_serviceactivity	Class	Tagged value *document* must refer to a model element of kind *s_businessdocumentkind.*
ServiceProtocol	*s_serviceprotocol*		Element	The owner must be a model element of kind *s_componentservicekind.*
	s_serviceprotocol _class	s_serviceprotocol	Class	Tagged value *scheme* must refer to a model element of kind *s_interactionscheme.* Tagged value *conversation* must refer to one or more model elements of kind *s_serviceconversation.*

Table 4.8: Stereotypes for Service Interaction specification (UML notation: Class Diagram)

tions which are a feature of component service kinds. Static service conversations are modelled Static Service Conversation Specification; this model kind is implemented with the UML Activity Diagram notation extended with stereotypes defined in Table 4.10. Dynamic service conversations are modelled with the Dynamic Service Conversation Specification model kind described in Table 4.10.

Life-cycle viewpoint

The life-cycle viewpoint is used for describing service ecosystem life cycles and their structure, actors and their responsibilities, and required artefacts. Artefacts are used to propagate knowledge elements between life-cycle actors, that is, ecosystem members and infrastructure services. Artefact sets are described in the life-cycle viewpoint; they provide means for specifying the consistency rules for the different roles of knowledge elements in ecosystem life cycles. In the implementation of the architecture framework defined in this thesis the consistency rules are not explicitly addressed. The artefact sets can be associated with descriptions of the consistency rules (e.g. as comments associated with UML model elements), and further utilized as requirements documentation for the corresponding infrastructure service implementations.

Typical stakeholders associated with the life-cycle viewpoint include ecosystem providers, infrastructure provider, service providers and consumers, and engineering tool providers. Ecosystem providers use the life-cycle viewpoint to declare the behaviour of the service ecosystem, as well as the propagation and semantics of knowledge elements. Infrastructure providers utilize the life-cycle viewpoint for guiding the design and implementation of infrastructure services. Service providers and consumers can use the views provided by the life-cycle viewpoint for identifying the responsibilities associated with different actors during ecosystem operation. The life-cycle viewpoint is summarized in Table 4.11.

The life-cycle viewpoint comprises two model kinds: *1)* Life-cycle structure specification, and *2)* Life-cycle behaviour specification. The UML-based modelling language for Life-cycle structure specification is implemented with stereotypes described in Table 4.12; UML Use Case diagram [111] is used as the basis for the notation. The Life-cycle structure specification provides means for describing the actors (including infrastructure services), artefacts, artefact sets, life cycles, and their inter-dependencies in a service ecosystem.

The Life-cycle behaviour specification provides means for modelling the behaviour of ecosystem life cycles. Each life cycle comprises one or more life-cycle phases with temporal dependencies. The Life-cycle behaviour specification is implemented with the stereotypes defined in Table 4.13. The notation is based on the UML State Machine diagram [111].

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
ServiceConversation	s_serviceconversation		Element	
	s_serviceconversation_activity	s_serviceconversation	Activity	The owner must be a model element stereotyped with *s_serviceprotocol*.
ConversationStage	s_conversationstage		Element	
	s_conversationstage_calloperationaction	s_conversationstage	CallOperationAction	The operation of the CallOperationAction must be an Operation stereotyped with *s_serviceactivity* in the Interface that is realized by the owner of this Activity.

Table 4.9: Stereotypes for Static Service Conversation specification (UML notation: Activity Diagram)

Meta-model element name	Stereotype name	Parent stereotype	UML metaclass	Constraints
ServiceConversation	s_serviceconversation		Element	
	s_serviceconversation _class	s_serviceconversation	Class	The owner must be a model element stereotyped with *s_serviceprotocol*. Tagged value *stages* must refer to a model element of kind *s_conversationstage_class*.
ConversationStage	s_conversationstage		Element	
	s_conversationstage _class	s_conversationstage	Class	Tagged value *activity* must refer to a model element of kind *s_serviceactivity*.

Table 4.10: Stereotypes for Dynamic Service Conversation specification (UML notation: Class Diagram)

Framed concerns	Ecosystem responsibilities; knowledge propagation; pragmatic interoperability.
Typical stakeholders	Ecosystem provider; Infrastructure provider; Service provider; Service consumer
Concepts	Life cycles, actors, artefacts, artefact sets.
Model kinds	Life-cycle structure specification; Life-cycle behaviour specification

Table 4.11: Life-cycle viewpoint.

Domain-ontology viewpoint

The domain-ontology viewpoint is utilized for defining service ecosystem-specific concepts and their relationships. Service ecosystem domain ontologies are based on the the conceptual model and the domain ontology model described in previous chapters. The domain ontologies extend on the concepts and model elements prescribed in the domain ontology model. The domain ontologies especially serve the purpose of establishing semantic interoperability (see Section 2.2) in service ecosystems. Domain ontologies provide unified vocabulary in service ecosystems, as well as define consistency rules to be maintained in the (distributed) ecosystem knowledge base. Especially, the domain ontologies enable sharing of interoperability knowledge: service ecosystem-specific domain ontologies enable knowledge sharing about service compatibility with other services and their applicability with respect to different models of collaboration, for example.

Typical stakeholders utilizing the domain-ontology viewpoint include ecosystem providers, infrastructure providers, service providers and consumers, as well as service engineers. Ecosystem providers typically define the ecosystem domain ontologies. Infrastructure providers use these definitions to implement infrastructure services which maintain the service ecosystem knowledge bases and establish semantic interoperability in the service ecosystem. Service providers and consumers use the domain ontologies as a foundation for maintaining and sharing interoperability knowledge during ecosystem operation.

Concepts used in the definition of domain ontologies include ecosystem concepts, concept sets, and concept relationships. A domain-ontology for service ecosystems must provide definitions for at least service liaisons, service contracts and service offers. Domain ontologies are defined using the model kind of Domain-ontology specification. The domain-ontology viewpoint is summarized in Table 4.1.2.

The Domain-ontology specification model kind is implemented over UML

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
Actor	s_actor		Namespace	
	s_infrastructureservicekind_actor	s_actor	Actor	
	s_actor_actor	s_actor	Actor	
Artefact	s_artefact	s_lifecycleproduct	Class	Owner is of kind *s_ecosystemmodel*, *s_ecosystemdomain* or *s_artefactset*.
ArtefactSet	s_artefactset	s_lifecycleproduct	Package	Owner is of kind *s_ecosystemmodel*, *s_ecosystemdomain* or *s_artefactset*.
Lifecycle	s_lifecycle		Element	
	s_lifecycle_component	s_lifecycle	Component	
LifecyclePhase	s_lifecyclephase		Element	
	s_lifecyclephase_usecase	s_lifecyclephase	UseCase	
LifecycleProduct	s_lifecycleproduct		Element	
N/A	actor_participates _phase		Association	From an actor to a lifecycle phase.
N/A	lifecyclephase_input		Association	From a lifecycle phase of artefact or artefact set.
N/A	lifecyclephase_output		Association	From a lifecycle phase to artefact or artefact set.

Table 4.12: Stereotypes for Life-cycle structure specification (UML notation: Use Case Diagram)

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
Lifecycle	s_lifecycle		Namespace	
	s_lifecycle_statemachine	s_lifecycle	StateMachine	Owner must be a model element stereotyped with *s_lifecycle_component.*
LifecyclePhase	s_lifecyclephase		Element	
	s_lifecyclephase_initialnode	s_lifecyclephase	InitialNode	
	s_lifecyclephase_pseudostate	s_lifecyclephase	Pseudostate	
	s_lifecyclephase_state	s_lifecyclephase	State	
N/A	lifecycle_transition		Transition	From a model element of kind *s_lifecyclephase* to another.

Table 4.13: Stereotypes for Life-cycle behaviour specification (UML notation: State Machine Diagram)

Framed concerns	Establishing a domain-specific shared conceptualization; semantic interoperability; sharing interoperability knowledge.
Typical stakeholders	Ecosystem provider; Infrastructure provider; Service provider; Service consumer
Concepts	Ecosystem concepts, concept sets, and concept relationships.
Model kinds	Domain-ontology specification

Table 4.14: Domain-ontology viewpoint.

Class Diagram notation. The stereotype definitions associated with the Domain-ontology specification are summarized in Table 4.15 and Table 4.16. In the summary tables the top level of the generalization hierarchy for *s_concept* stereotype is described. Each model element defined in the service ecosystem meta-model is represented in the model kind as a UML Stereotype with tag definitions reflecting the meta-model structure. Relationships between concepts defined in the service ecosystem meta-model are obligatory and represented as tag definitions in the corresponding stereotypes. Reflecting the service ecosystem meta-model, the stereotype *s_concept* has a boolean tag definition named *isDynamic* which declares if the corresponding concept (e.g. entity kind or service contract) is a dynamic (i.e. a template to be instantiated during ecosystem operation) or a static concept.

Each concept defined with the Domain-ontology viewpoint must be a representation of (a tagged value in the stereotypes) of at least on life-cycle product (i.e. artefact or artefact set).

Concept intension viewpoint

The concept intension viewpoint is used for providing the intensional definitions of concepts defined in a service ecosystem domain-ontology. The intensional definitions are specially used for sharing engineering knowledge, enabling domain-specific modelling practices, and for facilitating engineering tool interoperability. The service ecosystem meta-model defined in this thesis provides templates for the definitions of concept intensions. The pre-defined templates can be extended in an additive manner using the concept intension viewpoint. Consequently, the concept intension viewpoint facilitates model extension [11] within a well-defined, restricted part of the service ecosystem meta-model. The intensional definitions prescribe abstract syntaxes for a family of domain-specific languages,

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
Concept	s_concept		Element	Boolean tagged value *isDynamic* has a default value of *false*. The owner must be a model element of kind *s_domainontology*.
	s_concept_class	s_concept	Class	Optional tagged value *representationOf* must refer to a model element of kind *s_artefact*.
ConceptRelationship	s_conceptrelationship		Association	Applicable only between subtypes of *s_concept* stereotype.
ConceptSet	s_conceptset	s_concept	Package	Optional tagged value *representationOf* must refer to a model element of kind *s_artefactset*. Each artefact set and their hierarchies must be provided with a corresponding concept sets and their hierarchies.
CooperationFacility	s_cooperationfacility	s_concept_class	Element	
DomainConcept	s_domainconcept	s_concept_class	Class	
DomainOntology	s_domainontology		Package	Owner must be a model element of kind *s_ecosystemmodel*.
Entity	s_entity	s_concept_class	Element	
EntityKind	s_entitykind	s_concept_class	Element	

Table 4.15: Stereotypes for Domain-ontology specification (UML notation: Class Diagram)

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
ServiceLiaison	s_serviceliaison	s_concept_class	Class	Tagged value *phase* must refer to a model element of kind *s_lifecyclephase_usecase*.
ServiceContract	s_servicecontract	s_concept_class	Class	Tagged value *liaison* must refer to a model element of kind *s_serviceliaison*. Tagged value *channel* must refer to one or more model elements of kind *s_servicechannel*.
ServiceOffer	s_serviceoffer	s_concept_class	Class	Tagged value *provider* must refer to a model element stereotyped with *s_legalentity*. Tagged value *service* must refer to a model element stereotyped with *s_businessservice*.
ServiceChannel	s_servicechannel	s_concept_class	Class	
N/A	s_concept _generalization		Generalization	Applicable between model elements having similar kind of subkind *s_concept_class*.
N/A	s_concept _relationship		Association	Applicable between model elements of kind *s_concept_class*.

Table 4.16: Stereotypes for Domain-ontology specification (UML notation: Class Diagram; continued)

thus providing means for developing corresponding domain-specific modelling infrastructures. Moreover, the set of concept intensions serves enables tool interoperability: mappings to these unifying meta-models can be utilized to bridge organization and domain specific modelling notations and languages.

Typical stakeholders associated with the concept intension viewpoint include ecosystem providers, engineering tool providers, and service providers. Ecosystem providers declare the meaning of ecosystem concepts by providing their intensional definitions. Service engineers can utilize the concept intension viewpoint to extract meta-models enabling model-driven production of ecosystem-specific services. Finally, engineering tool providers can utilize this viewpoint for identifying relationships between different linguistic elements, and requirements for integrating service engineering tools with the ecosystem infrastructure (e.g. lifecycle support and bridging modelling languages).

The primary concepts involved in this viewpoint include features and feature interactions. Concept intensions are defined in this conceptual and architecture framework with features. The feature definitions as well as declarations of relationships between different features, or feature interactions, are defined in using the concept intension viewpoint defined in Table 4.17.

Framed concerns	Enabling domain-specific modelling practices; utilization of engineering knowledge; tool interoperability.
Typical stakeholders	Ecosystem provider; Service engineer; Engineering tool provider
Concepts	Features and feature interactions.
Model kinds	Feature specification

Table 4.17: Concept intension viewpoint.

The Concept Intension specification model kind is implemented over UML Class Diagram notation with the stereotypes defined in Table 4.18. Only the upper layer of the generalization hierarchy for the stereotypes is described in Table 4.18 and Table 4.19. For each concept defined in the domain-ontology a corresponding intension is provided as a UML Class or UML Enumeration (see [111]). Using this notation hierarchies of features can be declared. Feature generalization is utilized for specifying structural variations of concept intensions, e.g. sequence, choice or primitive types for business document types. UML Classes are utilized for defining dynamic features of concepts while UML Enumerations are used for defining static features.

All features defined with the Feature specification model kind must be owned by a model element stereotyped with a subtype of *s_concept_class*. Only features that are compatible with the consistency rules declared by the service ecosystem meta-model can be provided for a concept; for example, endpoint features are acceptable as elements owned by a model element representing an endpoint kind.

Engineering viewpoint

The engineering viewpoint addresses the concerns of enabling efficient service engineering, utilization of engineering knowledge, and tool interoperability. This viewpoint is used especially by engineering tool providers and service engineers for enabling their operation, as it describes the methodological elements available in the service ecosystem. The primary concepts used in the engineering viewpoint include engineering spaces, product models, binding models and engineering capabilities.

Engineering spaces are used for declaring engineering elements utilized in the service ecosystem for enabling service engineering practices. Engineering elements represent technical artefacts that are needed for instrumenting sharing of engineering knowledge and tool interoperability. In the current service ecosystem meta-model and the architecture framework, engineering elements include definitions for the types of identifiers and tools that are utilized in the service ecosystem.

Product models define engineering artefacts used as inputs and outputs of engineering activities. Typically the composition of engineering artefacts align with the ecosystem knowledge element structures, but other decompositions can also be utilized if required by the engineering methods. A product model comprises products parts (e.g. a meta-model representing business services), which again are composed of product elements (e.g. model elements of the business service meta-model). The concepts of product models, product parts and product elements as methodological artefacts are based on an approach of situational method engineering [103].

Bindings models prescribe what kinds of features are available for dynamic binding in engineering spaces associated with ecosystem domains. A binding model declares binding specifications which associate feature binding targets (e.g. service roles or business transactions) with appropriate bindable features (e.g. cooperative or contractual features).

Engineering methods are prescribed using a situational method engineering framework based on the notion of method chunks [103]. A method chunk is a reusable engineering asset utilizable for constructing engineering methods. A method chunk is an autonomous and coherent part of a method supporting realization of engineering activities [103].

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
Feature	s_feature		Element	The owner of the feature must of a model element stereotyped with *s_concept_class*.
	s_featuretype	s_feature	Element	
	s_featuretype_class	s_featuretype	Class	
	s_featuretype _enumeration	s_featuretype	Enumeration	
	s_featuretype _enumerationliteral		EnumerationLiteral	Owner must be a model element of kind *s_featuretype_enumeration*.
Property	s_property	s_feature	Element	Tagged value *conformsTo* must refer to a model element of kind *s_featuretype*.
	s_property_class	s_property	Class	
	s_property _enumeration	s_property	Enumeration	
	s_property _enumerationliteral		EnumerationLiteral	Owner must be a model element of kind *s_property_enumeration*.

Table 4.18: Stereotypes for Feature specification (UML notation: Class Diagram)

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
UserElement	s_userelement		Element	
	s_userelement _class	s_userelement	Class	
	s_userelement _enumeration	s_userelement	Enumeration	
	s_userelement _enumerationliteral		EnumerationLiteral	Owner must be a model element of kind *s_userelement _enumeration*.
FeatureInteraction	s_featureinteraction		Element	
	s_featureinteraction _association	s_featureinteraction	Association	Applicable only between model elements of kind *s_feature*.
	s_featureinteraction _class	s_featureinteraction	Class	Tag values *source* and *target* must refer to model elements of kind *s_feature*.
N/A	s_feature _generalization		Generalization	Applicable between model elements of kind *s_feature*. Applicable only between model elements with identical stereotypes (e.g. business document types).

Table 4.19: Stereotypes for Feature specification (UML notation: Class Diagram; continued)

The engineering viewpoint is implemented with four model kinds: *1)* Engineering space specification, *2)* Product model specification, *3)* Binding model specification, and *4)* Engineering method specification. The engineering viewpoint is summarized in Table 4.20.

Framed concerns	Enabling efficient service engineering; instrumenting utilization of engineering knowledge; tool interoperability.
Typical stakeholders	Engineering tool provider; Service Engineer
Concepts	Engineering spaces, product models, binding models, engineering methods
Model kinds	Engineering space specification; Product model specification; Binding model specification; Engineering method specification

Table 4.20: Engineering viewpoint.

The Engineering space specification model kind is implemented using the UML Class Diagram notation extended with the stereotypes defined in Table 4.21.

The Product model specification model kind is implemented using the UML Class diagram notation extended with the stereotypes defined in Table 4.22.

The Binding model specification model kind is implemented with the UML Class diagram notation extended with the stereotypes defined in Table 4.23, Table 4.24 and Table 4.25.

The Engineering method specification model kind is implemented over the UML Class Diagram notation with the stereotypes defined in Table 4.26 and Table 4.27.

Knowledge management viewpoint

The knowledge management viewpoint is utilized for enabling management and sharing of knowledge in a service ecosystem. The knowledge management viewpoint is used for defining the knowledge repositories, knowledge items and knowledge item sets required for maintaining the knowledge base for the service ecosystem. Knowledge repositories represent infrastructure services that maintain the consistency of the domain-ontology and validate correctness of intensional descriptions (e.g. service descriptions) to be published in the service ecosystem.

Knowledge items and knowledge item sets provide means for encoding and managing the different roles of knowledge elements in service ecosystem life cycles, and relationships between their ontological and intensional interpretations.

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
EngineeringSpace	s_engineeringspace		Package	Tagged value *productmodel* must be associated with a element of kind *s_productmodel*.
EngineeringActivity	s_engineeringactivity		Operation	Owner must be a model element of kind *s_tooltype*. Optional input parameters must be of kind *s_productelement*. The output parameter must be of kind *s_productelement*. The operation must own an element of kind *s_engineeringactivity _description*.
N/A	s_engineeringactivity _description		Constraint	The owner must be a model element of kind *s_engineeringactivity*. The constraint contains a textual description of the engineering activity.
IdentifierType	s_identifiertype		Class	Owner must be a model element of kind *s_engineeringspace*.
ToolType	s_tooltype		Interface	Owner must be a model element of kind *s_engineeringspace*.

Table 4.21: Stereotypes for Engineering space specification (UML notation: Class Diagram)

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
ProductModel	s_productmodel		Package	
ProductPart	s_productpart		Class	The owner must be a model element of kind *s_productmodel*. The tagged value *productelement* must refer to one or more model elements of kinds *s_productelement*.
ProductElement	s_productelement		Element	

Table 4.22: Stereotypes for Product model specification (UML notation: Class Diagram)

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
BindingSpecification	s_bindingspecification		Element	The owner must be a model element of kind *s_engineeringspace*.
FeatureBinding	s_featurebinding	*s_bindingspecification*	Element	
BindableFeature	s_bindablefeature		Element	The optional tagged value *provides* must refer to a model element of kind *s_qualitativecapability*.
CooperativeFeatureTarget	s_cooperativefeaturetarget		Element	
ServiceRole	s_servicerole	s_cooperativefeaturetarget	Element	
ContractualFeatureTarget	s_contractualfeaturetarget		Element	
BusinessTransaction	s_businesstransaction	s_contractualfeaturetarget	Namespace	
BusinessService	s_businessservice	s_contractualfeaturetarget	Element	
ExtraFunctionalFeatureTarget	s_extrafunctionalfeaturetarget		Element	
ServiceConnection	s_serviceconnection	s_extrafunctionalfeaturetarget	Namespace	
CooperativeFeature	s_cooperativefeature	s_bindablefeature	Element	
ContractualFeature	s_contractualfeature	s_bindablefeature	Element	
ExtraFunctionalFeature	s_extrafunctionalfeature	s_bindablefeature	Element	

Table 4.23: Stereotypes for Binding model specification (UML notation: Class Diagram)

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
CooperativeFeatureBinding	s_cooperativefeaturebinding	s_featurebinding	Class	Tagged value *feature* must refer model elements of kind *s_cooperativefeature*. Tagged value *target* must refer to a *s_cooperativefeaturetarget*.
ContractualFeatureBinding	s_contractualfeaturebinding	s_featurebinding	Class	Tagged value *feature* must model elements of kind *s_contractualfeature*. Tagged value *target* must refer to a *s_contractualfeaturetarget*.
ExtraFunctional-FeatureBinding	s_extrafunctional-featurebinding	s_featurebinding	Class	Tagged value *feature* must refer to model elements of kind *s_extrafunctionalfeature*. Tagged value *target* must refer to a *s_extrafunctionalfeaturetarget*.

Table 4.24: Stereotypes for Binding model specification (UML notation: Class Diagram); continued

Meta-Model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
BindingSpecification	s_bindingspecification		Element	The owner must be a model element of kind *s_engineeringspace*.
PropertyBinding	s_propertybinding	*s_bindingspecification*	Element	
BindableProperty	s_bindableproperty		Element	
CooperativeProperty	s_cooperativeproperty	s_bindableproperty	Element	
ContractualProperty	s_contractualproperty	s_bindableproperty	Element	
CooperativePropertyTarget	s_cooperativepropertytarget		Element	
LegalEntity	s_legalentity	s_cooperativepropertytarget	Element	
ContractualPropertyTarget	s_contractualpropertytarget		Element	
BusinessService	s_businessservice	s_contractualpropertytarget	Class	
CooperativePropertyBinding	s_cooperativepropertybinding	s_propertybinding	Class	Tagged value *property* must refer to at least one model element of kind *s_cooperativeproperty*. Tagged value *target* must refer to a model element of kind *s_cooperativepropertytarget*.
ContractualPropertyBinding	s_contractualpropertybinding	s_featurebinding	Class	Tagged value *property* must refer to at least one model element of kind *s_contractualproperty*. Tagged value *target* must refer to a model element of kind *s_contractualpropertytarget*.

Table 4.25: Stereotypes for Binding model specification (UML notation: Class Diagram); continued

Meta-model element name	**Stereotype name**	**Parent stereotype**	**UML metaclass**	**Constraints**
MethodChunk	s_methodchunk		Class	Tagged value *guide* must refer to a model element of kind *s_guideline*. Tagged value *productpart* must refer to a model element of kind *s_productpart*.
Method	s_method	s_methodchunk	Class	Tagged value *chunk* must refer to at least one element of kind *s_methodchunk*.
Guideline	s_guideline		Element	The element must own exactly two *UML::Constraint* elements of kind *s_intention* and *s_situation*. Tagged value *body* of type *UML::String* must provide a generic description for the guideline.
StrategicGuideline	s_strategicguideline	s_guideline	Element	Tagged value *guideline* must refer to one or more model elements of kind *s_guideline*.
SequenceGuideline	s_sequenceguideline	s_strategicguideline	Class	
AndGuideline	s_andguideline	s_strategicguideline	Class	
OrGuideline	s_orguideline	s_strategicguideline	Class	
TacticalGuideline	s_tacticalguideline	s_guideline	Element	Tagged value *guideline* must refer to one or more model elements of kind *s_guideline*.
CompositionGuideline	s_compositionguideline	s_tacticalguideline	Class	
ChoiceGuideline	s_choiceguideline	s_tacticalguideline	Class	

Table 4.26: Stereotypes for Engineering method specification (UML notation: Class Diagram)

Meta-model element name	Stereotype name	Parent stereotype	UML metaclass	Constraints
SimpleGuideline	s_simpleguideline	s_guideline	Class	Tagged value *activity* must refer to a model element of kind *s_engineeringactivity*.
Intention	s_intention		Constraint	Owner must be a model element of kind *s_guideline*. Both name and specification must be provided for the constraint. Name is interpreted as the *verb* of the intention and specification as the *manner* of the specification, as defined in the ecosystem meta model. Tagged value *target* must refer to a model element of kind *s_productpart*.
Situation	s_situation		Constraint	Owner must be a model element of kind *s_guideline*. The specification of the constraint is interpreted as the *descriptor* of the guideline's situation. Tagged value *input* refers to zero or more model elements of kind *s_productpart*.

Table 4.27: Stereotypes for Engineering method specification (UML notation: Class Diagram); continued

For example, in the Pilarcos service ecosystem [87, 85, 86] a business network model (BNM) has different roles in distinct phases of the eContract establishment life cycle: at the beginning of the population phase it serves as a contract template, after successful population as a contract proposal, and after successful negotiation as a community contract. The intensional meaning of the BNM is similar during all these roles, however, the consistency rules and interpretation about the role of the BNM differs in all these phases. It is the responsibility of knowledge repositories to provide the means for establishing coherent service ecosystem knowledge bases which especially address the multiple interpretations of knowledge elements in ecosystem life cycles. The knowledge management viewpoint is defined in Table 4.1.2.

Framed concerns	Enabling ecosystem knowledge management and sharing; establishing coherent service ecosystem knowledge bases
Typical stakeholders	Ecosystem provider; Infrastructure provider
Concepts	Knowledge repositories, knowledge items and their types, knowledge item sets, knowledge item relationships
Model kinds	Knowledge repository specification; Knowledge item specification

Table 4.28: Knowledge management viewpoint.

The knowledge management viewpoint is implemented with two model kinds: *1)* Knowledge repository specification, *2)* Knowledge item specification.

The Knowledge repository specification model kind is described in Table 4.29. Each knowledge repository provides a representation of a domain-ontology concept and is responsible for maintaining the consistency rules associated with the corresponding concept sets. A knowledge repository may implement one or more concept sets and does not have to represent all the concept sets associated with a domain-ontology concept. The semantics of a concept can be distributed between a set of knowledge repositories, one repository for each concept set definition, for example. Such distribution of concept semantics is useful in situations where the role of the corresponding concept in ecosystem life cycles implies different policies with respect to trust or privacy, for example. However, all concept sets defined in a service ecosystem domain-ontology must be provided with knowledge item set representations.

Meta-model element name	Stereotype name	Parent stereotype	UML metaclass	Constraints
KnowledgeRepository	s_knowledgerepository		Package	Tagged value *representationOf* must refer to a model element of kind *s_concept*. Tagged value *modeltype* must refer to a model element of kind *s_knowledgeitemtype*. Tagged value *engineeringspace* must refer a model element of kind *s_engineeringspace*. Tagged value *relationships* refers to zero or more model elements of kind *s_knowledgerelationshiptype*.
KnowledgeItemSet	s_knowledgeitemset		Package	Owner is a model element of kind *s_knowledgerepository* or *s_knowledgeitemset*. Tagged value *representationOf* must refer to a model element of kind *s_conceptset*.

Table 4.29: Stereotypes for Knowledge repository specification (UML notation: Class Diagram)

The Knowledge item specification model kind is implemented using the UML Class Diagram notation extended with the stereotypes defined in Table 4.30. The Knowledge item specification is used for defining domain-specific representations of concept intensions. For example, representations of business service kind intension may differ in web services architecture or cloud computing-based environments. The knowledge item types in essence define the meta-models for representations of (technology-specific) intensional definitions. The correspondences between knowledge item types and the intensional definitions are not specified in this version of the architectural framework. However, such mappings are essential for instrumenting tool interoperability and mappings between domain-specific languages, as they provide means for unifying different representations to the common intensional definitions. These correspondences could be specified and implemented with use of Declarative QVT [114]; such an approach for defining correspondences between models and modelling elements has been applied for example in [129].

4.2 A model-driven methodology for service ecosystem engineering

The model-driven methodology for service ecosystem engineering enables efficient production of infrastructure services and engineering tools required by specialized service ecosystems. Model transformations are utilized in the methodology to produce first refined versions of the service ecosystem architecture models. The refined architecture models are then applicable as input for model transformations which produce technology-specific engineering artefacts. Technology-specific engineering artefacts include Java interfaces and classes for infrastructure services, deployment descriptors for different middleware and cloud computing environments, and definitions for domain-specific modelling languages, for example. The methodology can be extended with additional or refined model transformations by service ecosystem stakeholders. Such model transformations created by service ecosystem stakeholders may be targeted to utilize specific software product lines [33] or technology platforms, for example.

4.2.1 Engineering activities

The implementation of the model-driven methodology for service ecosystem engineering is founded on the application of SEAF, explicit service ecosystem models, and a collection of model transformations. The methodology comprises four primary engineering activities: *1)* ecosystem analysis and design, *2)* architecture model refinement, *3)* ecosystem instrumentation, and *4)* ecosystem operation, as

Meta-model element name	Stereotype name	Parent stereotype	UML metaclass	Constraints
KnowledgeItemType	s_knowledgeitemtype		Class	Tagged value *representationOf* must refer to a model element of kind *s_concept_class* (defines the location of intensional definition in the UML model).
KnowledgeRelationshipType	s_knowledgerelationshiptype		Association	Tagged value *representationof* must refer to a model element of kind *s_conceptrelationship*.

Table 4.30: Stereotypes for Knowledge item specification (UML notation: Class Diagram)

illustrated in Figure 4.1. The activities are complex, that is, they a composed of other more simple engineering activities. The primary activities cover the service ecosystem engineering phases (analysis, design, instrumentation and operation) introduced in Chapter 1.

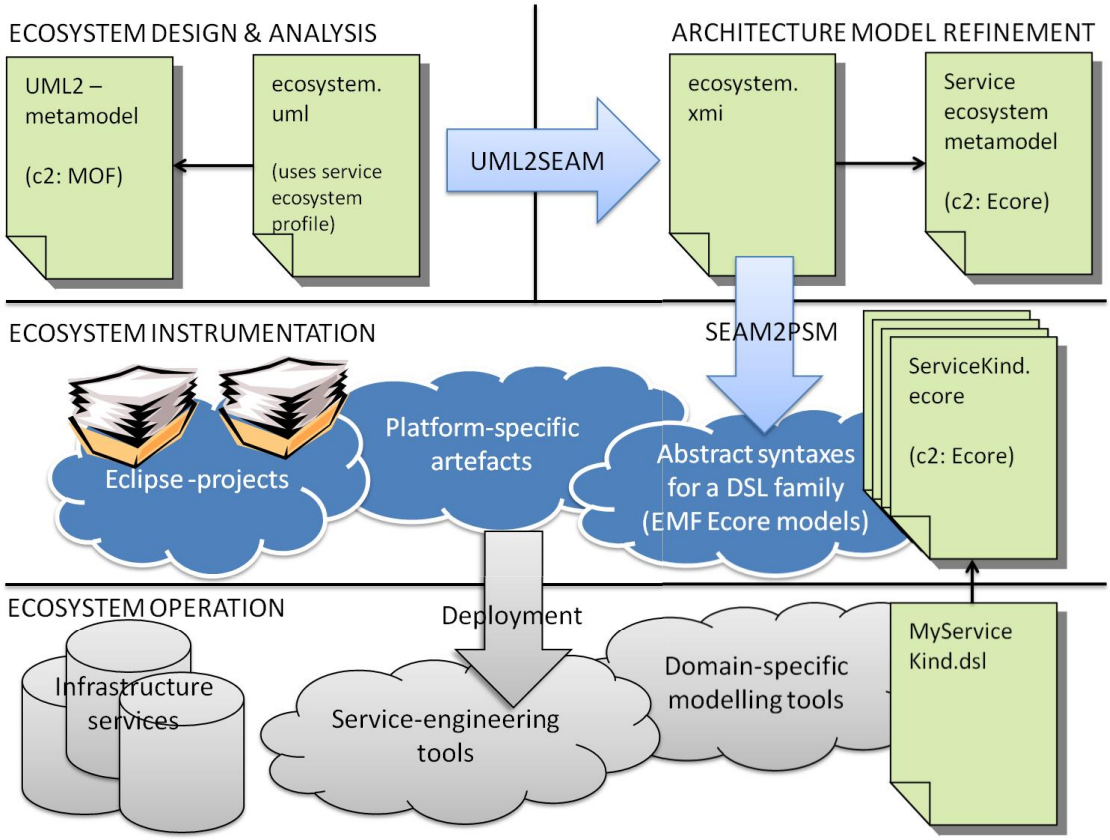

Figure 4.1: Primary and secondary engineering activities in the model-driven methodology for service ecosystem engineering.

The primary activities in the methodology are model-driven: models of service ecosystems and its components are considered as the primary artefacts of engineering activities. The models created and consumed by the engineering activities are explicit and machine-readable, and thus can be utilized efficiently by different engineering tools.

The current methodology implementation covers the primary engineering activities of ecosystem design and analysis, architecture model refinement, and ecosystem instrumentation. The specifics of service ecosystem deployment and operation, e.g. provisioning of infrastructure services and resources such as service-engineering and modelling tools or runtime management of service ecosystems, is not addressed by the current methodology implementation. Runtime management of service ecosystems has been addressed in previous publications [143, 140, 87].

The primary activities are connected by three secondary engineering activities depicted in Figure 4.1: *1)* UML2SEAM, *2)* SEAM2PSM, and *3)* Deployment. These secondary engineering activities enable transitioning between the primary activities. The transitioning is enabled by transforming engineering artefacts of the preceding engineering activities to engineering artefacts required in succeeding activities.

The secondary engineering activities of the methodology are also model-driven: they are implemented as a series of model-transformations consuming and producing models from and to the primary engineering activities. Both model-to-model and model-to-text transformations are utilized for implementation of the secondary engineering activities.

During the ecosystem design and analysis activity depicted in Figure 4.1 the conceptual framework and SEAF developed in this thesis are utilized for producing a UML-model of the service ecosystem. For this purpose, a UML-modelling tool is used in conjunction with the UML-profile defined in Section 4.1. In Figure 4.1 the resulting UML model is represented as a document named *"ecosystem.uml"*. Conformance relationships between models and their meta-models are represented with black arrows between documents when applicable, and with annotations in the documents: the *ecosystem.uml* conforms to the UML2-meta-model [113], while the UML2-meta-model conforms to the MOF [112] (annotation *c2: MOF* in the figure).

The secondary engineering activity named UML2SEAM (see Figure 4.1) takes the UML-model of the service ecosystem and transforms it to an EMF [39] Ecore model. During the same engineering activity, the Ecore representation of the service ecosystem (named *"ecosystem.xmi"* in Figure 4.1) is refined, since the UML-based service ecosystem description does not include all elements of the service ecosystem meta-model defined in Chapter 3. The UML-model does not include any elements of the knowledge management model described in Section 3.9, and includes only binding model specifications of the engineering model described in Section 3.8. Moreover, default knowledge repository elements are produced during the UML2SEAM engineering activity for each domain ontology concept that is not explicitly associated with one in the UML-model. Such a design choice in the methodology implementation was made, since a large portion of the knowledge management and engineering models can be deduced from other parts of the service ecosystem architecture model. In this way the work load of service ecosystem designers can be considerably decreased and made more efficient.

During architecture model refinement the Eclipse EMF model can be refined by service ecosystem engineers. Such refinement may include for example addition of service ecosystem specific methods, and annotating the Ecore model with declarations that could be utilized in succeeding engineering phases. Model anno-

tations may include for example definitions of constraints over the model and the service ecosystem itself with languages such as the Object Constraint Language (OCL) [110] or different business rule languages. In the current implementation of the methodology, such model annotations are not taken into account.

The service ecosystem model provided by the architecture model refinement activity is consumed by the secondary engineering activity named SEAM2PSM in Figure 4.1. The SEAM2PSM activity takes the EMF Ecore model of the service ecosystem and produces several platform-specific outputs. The outputs especially include Ecore models defining meta-models for the service concept intensions, a set of Eclipse projects for the required knowledge repositories, as well as OSGi [120] and Java programming language artefacts for implementation of the repositories. The infrastructure services generated from service ecosystem architecture models are based on the OSGi [120] industry standard, and Java programming language. OSGi [120] is a standard maintained by the OSGi Alliance[1] for creating modular and dynamic component-based platforms.

While the UML2SEAM engineering activity provides a refining model transformation at the same abstraction level (platform-independent level), the SEAM2PSM engineering activity transforms the platform-independent service ecosystem description to a set of platform-specific models and other engineering artefacts.

Ecosystem instrumentation is a primary service ecosystem engineering activity where the artefacts produced by SEAM2PSM activity are utilized for developing the necessary infrastructure services and tools. This activity especially involves implementation of the knowledge repository logic using the Java-classes generated by the SEAM2PSM activity. The other engineering artefact, such as the generated OSGi descriptions, are also subject to refinement by service ecosystem engineers and software developers. The sub-activities of ecosystem instrumentation may involve additional platform-specific model transformations that either refine the artefacts provided by the SEAM2PSM activity, or generate additional software engineering artefacts from the service ecosystem model. The development activities taken inside the ecosystem instrumentation are not in the focus of this thesis and thus not discussed further.

During the deployment activity the artefacts produced during ecosystem instrumentation are installed and configured into a (OSGi-based) service platform. For example, an enterprise service bus (ESB) such as the Apache ServiceMix [2] could be used as a the target platform for deploying the generated OSGi-bundles (the Eclipse projects).

Finally, service ecosystem operation includes engineering activities for managing and governing the service ecosystem [143]. The infrastructure services

[1] http://www.osgi.org
[2] http://servicemix.apache.org/

developed and deployed during the previous activities are utilized by ecosystem members for maintenance of service ecosystem knowledge, service delivery and establishment of service collaboration networks, for example. The service ecosystem model can be utilized for progressing interoperability in the service ecosystem with model-based interoperability approaches (see for example [13, 89]). Service engineers and service developers utilize ecosystem specific tools for increasing the efficiency of service delivery and development. Implementations of these tools are based on the platform-specific models and software engineering artefacts developed during the service ecosystem instrumentation activity. Especially, the domain-specific models created with ecosystem specific modelling languages and tools can be founded on the meta-models generated in the SEAM2PSM activity.

4.2.2 Implementation details

A collection of software engineering tools have been developed for supporting and implementing the activities of the service ecosystem engineering methodology. The tool implementations are based on utilization of open, industry-adopted standards and frameworks, such as UML [111] modelling language, Eclipse [38] framework, Java programming language and the OSGi [120] industry standard. Implementation details of these tools are next discussed following the order of the service ecosystem engineering activities described above.

For supporting the service ecosystem analysis and design a SEAF tool was implemented. The SEAF implementation was developed over MagicDraw UML [3] which is a commercial UML-modelling tool. First, a UML-profile conforming with the definitions given in Section 4.1 was developed for providing a modelling notation for service ecosystem architectures. The MagicDraw UML tool provides means for extending the the tool with modelling notation specific user interface elements and thus creating UML-based domain-specific languages. This capability of the tool was utilized for enhancing the user experience of the resulting service ecosystem modelling tool. For example, the different model kinds specified in Section 4.1 were provided with specialized diagrams and model elements applicable in them. Instead of using generic UML diagrams, its modelling elements and attaching appropriate stereotypes manually to the modelling elements the end-user of the modelling tool can now directly model with concepts such as *"Actor"* or *"Dynamic Service Choreography"*.

The service ecosystem meta-model and the UML-profile developed with the UML-modelling tool were exported to Eclipse UML2 models, and further, to EMF [39] Ecore models. The Ecore models were then utilized for developing a generator plugin for the Eclipse framework. The generator plugin extends the

[3] http://magicdraw.com

Eclipse-tool with operations implementing the secondary engineering activities of the service ecosystem engineering methodology. Basically the plugin provides an user interface for initiating the *UML2SEAM* and *SEAM2PSM* engineering activities discussed in Section 4.2.1.

The user-interface of the generator plugin provides service ecosystem engineers with three operations: *1)* UML2SEAM, *2)* SEAM2PSM, and *3)* SEAM2-Java. The UML2SEAM-operation implements the similarly named ecosystem engineering activity. It takes as an input as service ecosystem architecture model that conforms with the Eclipse UML2 meta-model and which utilizes the service ecosystem architecture UML-profile defined in Section 4.1. As an end-result of running the operation a new project is created in the Eclipse workspace which includes an EMF Ecore representation of the service ecosystem architecture.

The *UML2SEAM* operation produces default product models, engineering methods and tool types, and product models that represent creation of knowledge items in the service ecosystem. For each concept intension in the UML model a product part is created which includes features of the intensional definition. The default method includes a single method chunk which produces a product part corresponding to an intensional definition. For each intensional definition a tool type is generated which includes engineering activities producing the features and with no inputs required. The default elements generated by the *UML2SEAM* model transformation can be re-factored and complemented manually by service ecosystem architects.

The UML2SEAM-operation comprises a collection of sub-activities that are implemented as QVT-transformations. The different transformations for implementing the UML2SEAM-operation are summarized in Table 4.31. Each row in the table summarizes a model transformations by defining its name, kind (M2M or M2T for model-to-model and model-to-text transformations correspondingly), input and output models, and abstraction levels involved in the transformation. All model-to-model transformation are implemented using QVT [114] and model-to-text transformations with the Eclipse Acceleo[4] which is an implementation of the OMG MOF Model to Text Language (MTL) [115].

Input and output columns denote the meta-model or artefact kind consumed and produced by the transformation: *UML* denotes the UML2 meta-model of the Eclipse framework [38], *SEMM* denotes the service ecosystem meta-model (its Ecore-based implementation), *Ecore* denotes the EMF Ecore meta-model while *GenModel* denotes the EMF Ecore Generator meta-model [39], *OSGi* the OSGi service descriptors [120], and finally *Java* denotes that a file containing source code in the Java programming language is generated.

For representation of abstraction levels the three-level abstraction hierarchy

[4] http://www.eclipse.org/acceleo/

Name	Description	Kind	Input	Output	Abs. levels
UML2SEAM	Creates an Ecore model from the UML model	M2M	UML	SEMM	PIM-PIM
SEAMRefinement	Refines the Ecore model	M2M	SEMM	SEMM	PIM-PIM

Table 4.31: *UML2SEAM* transformations.

of the OMG's MDA (Model-Driven Architecture) [53] industry standardization is used. The MDA abstraction levels include (from most abstract to most concrete level): *1)* CIM (Computation-Independent Model), *2)* PIM (Platform-Independent Model), and *3)* PSM (Platform-Specific Model). CIM is usually considered as a business model or domain model which dictates what the system under consideration should do. PIM is typically considered as a model which describes how the goals and requirements stated in a CIM can be achieved. PIM is still platform and technology agnostic and defines for example abstract service interfaces or different kinds of abstract behaviour. Service ecosystem architecture models in this thesis considered primarily as PIM-level models, although some model elements are typical CIM elements, such as service ecosystem capabilities and actors (as stakeholders). Finally, PSM is a technology-specific model. In this thesis the PSMs include for example OSGi models and Java code (which can also be considered as a sort of model as a representation of a system). For example a PIM-PIM transformation in Table 4.31 is a abstraction-preserving transformation, while PIM-PSM transformation creates a more concrete model from a more abstract model. The additional information required for such abstraction level change is encoded in the model transformation implementations.

The SEAM2PSM-operation comprises a collection of activities that generates platform-specific software artefacts from the EMF Ecore service ecosystem architecture model. First, the SEAM2PSM-operation creates Eclipse projects for knowledge repositories. The projects are structured and configured as Eclipse Java projects. Required build scripts and configuration files are provided for each project individually. Especially, dependencies between the knowledge repository projects are configured in the generated files, such as the Eclipse plugin descriptors and OSGi service descriptors.

The transformations implementing the SEAM2PSM operation are described in Table 4.32. The SEAM2PSM transformations operate both between abstraction levels (PIM to PSM transformations) and within the PSM abstraction level as refining transformations.

After the SEAM2PSM-operation the user is provided with Eclipse projects for all knowledge repositories defined in the service ecosystem architecture model. More over a EMF Ecore GenModel for generating Java-code corresponding to the

Name	Description	Kind	Input	Output	Abs. levels
SEAM2EMF	Creates a meta-model representing concept intension within a knowledge repository project.	M2M	SEMM	Ecore	PIM-PSM
SetReferences	Creates model-to-model references between distinct Ecore models.	M2M	Ecore	Ecore	PSM-PSM
CreateGenModel	Creates an EMF GenModel from an Ecore model	M2M	Ecore	Genmodel	PSM-PSM

Table 4.32: *SEAM2PSM* transformations.

Name	Description	Kind	Input	Output	Abs. levels
SEAM2Java	Generates the Java interface and class files.	M2T	SEMM	Java	PIM-PSM
SEAM2OSGi	Creates an OSGi component description for a knowledge repository project.	M2T	SEMM	OSGi	PIM-PSM
SEAM2Manifest	Generates an Eclipse plugin descriptor for the knowledge repository project.	M2T	SEMM	MANIFEST.MF	PIM-PSM

Table 4.33: *SEAM2Java* transformations.

individual Ecore models generated by the SEAM2EMF model-to-model transformation is provided. The user can now do changes to the GenModel, if necessary. As the final activity in the SEAM2PSM-operation, the user applies the default EMF code generation functionality over the provided GenModel for generating model management code for the knowledge repositories.

The SEAM2Java-operation implements rest of the SEAM2PSM engineering activity and comprises a set of model-to-text transformations. The model transformations especially generate files containing code for Java interfaces and classes for the knowledge repositories. The knowledge repository interfaces are based on the REST-idiom [56]. The transformations are summarized in Table 4.33.

The operations described above are launched from the developed Eclipse plug-in. The plug-in enables ecosystem developers to generate the required development projects for infrastructure services. A pop-up action is accessible in the Eclipse user interface for launching the operations, as illustrated by the screen-capture in Figure 4.2. In the figure, an ecosystem engineer has executed the UML2SEAM and SEAM2PSM operations: there are a collection of generated projects available in the Eclipse workspace. In the figure the ecosystem engineer is about the execute the SEAM2Java operation over the EMF Ecore representation of the service ecosystem model.

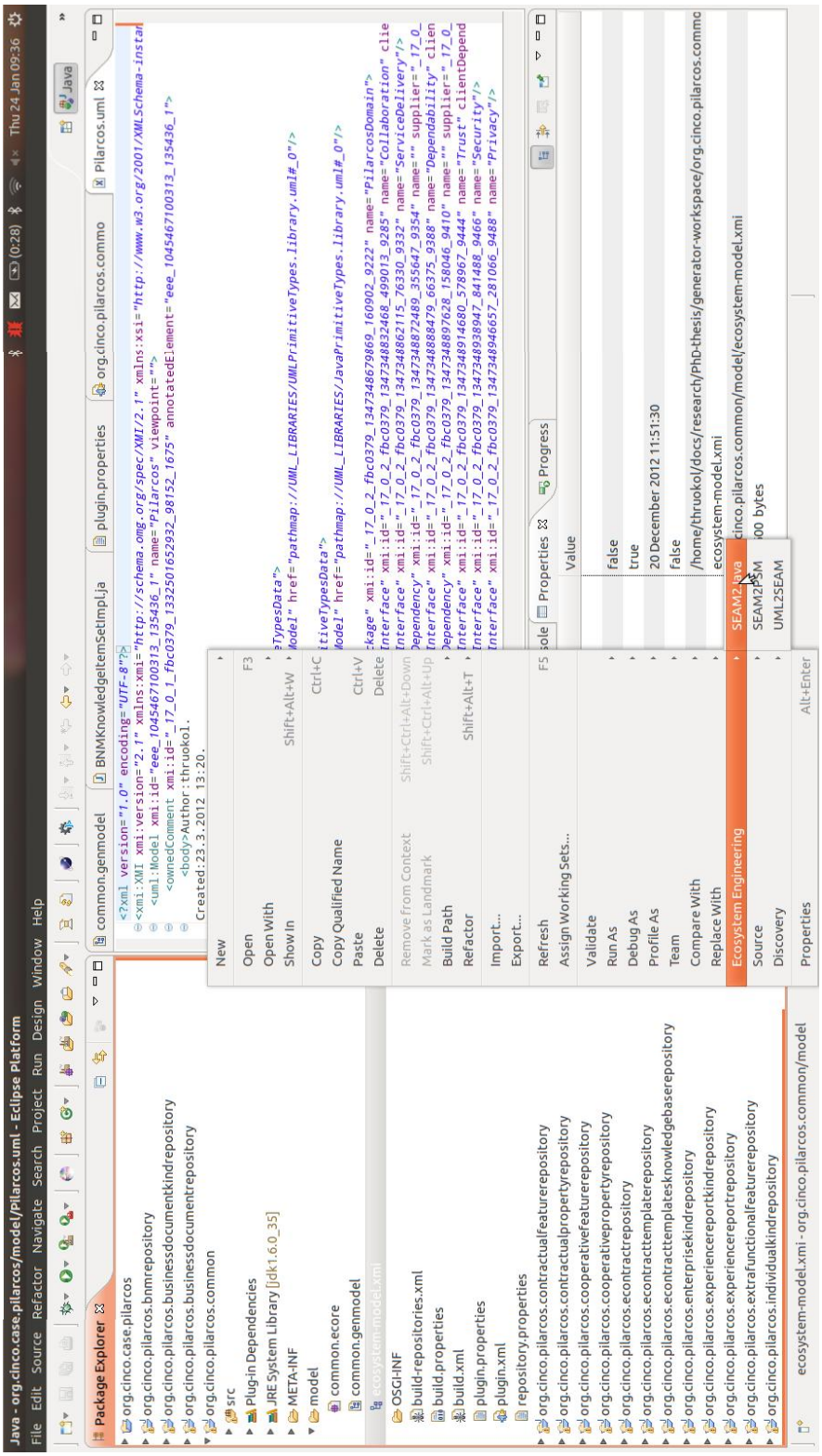

Figure 4.2: An ecosystem engineer about the execute the SEAM2Java operation in the Eclipse environment.

Chapter 5

Validation and evaluation

Validation and evaluation of the model-driven approach for service ecosystem engineering comprises several assessments. The feasibility of the approach is validated by assessing the technical implementability of the individual artefacts. The applicability of the approach is validated with respect to the practical usability of the artefacts for service ecosystem analysis, design and instrumentation. For this purpose a case study is conducted where a state-of-the-art service ecosystem is first modelled. The resulting service ecosystem architecture model is then utilized within the model-driven methodology for service ecosystem engineering. The research method followed in this dissertation is design science [62] whose goal is to provide utility for system stakeholders. The utility of the approach is evaluated with respect to the needs of service ecosystem stakeholders and requirements stemming from the definition of service ecosystem sustainability and vision of open service ecosystems.

In this chapter, the model-driven approach for service ecosystem engineering is assessed. The research process that has been conducted and the evaluation approach is first discussed in Section 5.1. After that in Section 5.2 SEAF is utilized for analysis and design of the Pilarcos service ecosystem. The resulting architecture model is used in the model-driven methodology for service ecosystem engineering for generating engineering artefacts representing Pilarcos-specific infrastructure services. Evaluation results are presented in Section 5.3. The chapter is concluded with a discussion and comparison of related work in Section 5.4.

5.1 Evaluation approach

The primary research question underlying this academic dissertation is how to enable transition from *ad hoc* service ecosystems to sustainable service ecosystems and to support the vision of open service ecosystems. Ecosystem sustainability

was defined as the quality of a service ecosystem to support continued viability (see Definition 2.1 in Section 2.3). The transition is enabled by facilities for rigorous service ecosystem engineering developed in this thesis. Viability of service ecosystems must be addressed during the system engineering life cycles; this implies that the concerns of primary stakeholders must be addressed during the analysis, design, instrumentation phases. Continuation of viability must be established by supporting the activities required for ensuring relevance and sensibility of ecosystem operation.

Methodologically the research process that has been conducted follows the design science paradigm [62]. Design science *"...seeks to create innovations ... through which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished"* [62]. The main innovation of this thesis is the model-driven approach for service ecosystem engineering. In the approach service ecosystems are considered as holistic systems; this is something that is missing in the current state-of-the-art approaches, as will be identified in Section 5.4. The benefits of applying model-driven principles in this context are two-fold. First of all, formalization of service ecosystem concepts as a unified meta-model enables management of the inherent structural complexity of service ecosystems. Secondly, the model-driven artefacts developed in this thesis facilitate rigorous service ecosystem engineering by providing means for supporting analysis, design, instrumentation (implementation) and operation of service ecosystems. In the context of this academic dissertation, the information systems under study are service ecosystems.

In the design science approach, purposeful artefacts are built and evaluated to address unsolved problems [62]. In this thesis, facilitation of service ecosystem sustainability is the unsolved problem to be tackled with. Towards solving this problem, this thesis has defined an approach for service ecosystem engineering comprising four artefacts: *1)* a conceptual model characterized in Chapter 2, *2)* a service ecosystem meta-model described in Chapter 3, *3)* an architecture framework described in Section 4.1, and *4)* a model-driven methodology introduced in Section 4.2.

In the rest of this chapter, the feasibility, applicability and utility of the approach for service ecosystem engineering are evaluated. The feasibility of the approach is validated by assessing the technical implementability of the individual artefacts. The applicability of the approach is validated with respect to the practical usability of the artefacts for service ecosystem engineering. The goal of design-science research is utility [62]. The utility of the approach is evaluated with respect to the needs of service ecosystem stakeholders, and requirements stemming from the definition of service ecosystem sustainability defined in Section 2.3.

5.2 Evaluation case: Pilarcos service ecosystem

The Pilarcos framework has been selected as the evaluation case for assessing the completeness of the conceptual model and applicability of the model-driven approach for service ecosystem engineering. The Pilarcos framework [87, 85] addresses several aspects required in open service ecosystems, such as dynamic collaboration establishment, trust and reputation management, and ecosystem evolution. Due to these characteristics and their inherent structural complexity, Pilarcos is considered especially suitable for evaluating the approach.

In the evaluation the concepts of the Pilarcos service ecosystem are first formalized with SEAF. The corresponding architecture models are constructed using a UML tool and an UML profile [111] implementing the viewpoints and model kinds described in Section 4.1.

In the following, a representative selection of the Pilarcos architecture models are described with a discussion of their characteristics. The primary concepts and characteristics of the Pilarcos service ecosystem are first described in Section 5.2.1. After that, the Pilarcos service ecosystem architecture models are presented in Section 5.2.2.

5.2.1 Pilarcos service ecosystem

The Pilarcos framework [87, 85] for open service ecosystems provides concepts and infrastructure for managing loosely coupled inter-enterprise collaborations. The framework uses federation for establishing interoperability between autonomous ecosystem members, and repositories providing consistency and conformance of interoperability knowledge. The infrastructure services include [87]:

- services for establishing, modifying, monitoring, and terminating collaborations, or looking from the business service point of view, operations for joining and leaving a collaboration either voluntarily or by community decision and leaving a trace in the global business world about the success of the collaboration; and
- a set of repositories for storage of collaboration models, and ontologies of service types and services, for example, to support interoperability validation.

The Pilarcos framework proposes a model of inter-enterprise collaborations as *business networks* consisting of independently developed business services. A *business service* denotes a set of functionalities provided by an enterprise to its clientele and partners. It is governed by the enterprise's own business rules and policies, as well as by business contracts and regulatory systems controlling the

business area. Business services are developed independently, and the provided Pilarcos infrastructure services are used to ensure that technical, semantic, and pragmatic interoperability is maintained in the business network.

A business network is established dynamically to serve a certain business scenario or opportunity that is made commonly known by publishing a *business network model* (BNM) [87]. The business network model captures the roles and business processes that are relevant for the business scenario, for example.

A *network management agent* (NMA) [85] represents a collaboration member in the business network. It handles negotiations with potential new members and re-negotiations if members are changed, it keeps up status information for the collaboration, and determines the suitable reaction to collaboration events such as contract termination. Every member of the service ecosystem has its own network management agent, and they are considered to be fully trusted local agents [85].

The collaboration establishment life cycle in the Pilarcos framework is explicit with service selection, population, negotiation and dissolution phases [87, 143]. When a new service collaboration needs to be established, a service ecosystem member first calls the *populator* [85] which is an infrastructure service responsible for executing collaboration establishment processes. The populator chooses the most suitable candidates for each role from the set of available service offers. The service offers are maintained by *service offer repositories*. Conformance with the constraints and requirements defined by the business network model and the collaboration initiator are used for selecting the set of interoperable service offers. Based on the collaboration proposals given by the populator, the initiating ecosystem member runs a negotiation with the service providers delivering the service offers [85].

After service discovery, the populator chooses the most suitable service offers for each role from the set of dependable services. Conformance with the constraints and requirements defined by the business network model and the collaboration initiator are used for selecting the set of compatible services. Based on the collaboration proposals given by the populator, the initiating NMA runs a negotiation with the NMAs of the other proposed partners [85].

During the business service publication phase *service types* [139] are used for validating conformance of business services with available component service kinds. A service type defines the characteristic features for a kind of component service by expressing its behavioural and non-functional features. Service types are used to constrain the behaviour of business services, to validate consistency of cooperation abstractions, and for verifying behavioural substitutability and compatibility between kinds of services [135].

The interoperability management approach is a federated one: all business services are developed independently, and the provided infrastructure services are

used to ensure that technical, semantic, and pragmatic interoperability is maintained in the business network. A shared service ecosystem architecture model representing the Pilarcos framework provides consistency and conformance criteria to be maintained by the knowledge repositories. This is invaluable for enabling collaboration establishment life cycles in open service ecosystems with autonomic participants.

5.2.2 Modelling Pilarcos service ecosystem architecture

In the following, the architecture model for the Pilarcos service ecosystem is described. A selection of the UML-diagrams presenting the architecture model are illustrated and explained. The selection of diagrams provides a representation of the foundational elements of SEAF as well as the Pilarcos service ecosystem. The discussion is structured in accordance with the viewpoints presented in Section 4.1. It should be noted, that the knowledge management viewpoint is not utilized in the case study, since the refining model transformation (UML2SEAM) described in Section 4.2 automatically generates the required knowledge management model elements.

Pilarcos ecosystem capabilities

The Pilarcos service ecosystem exposes two fundamental capabilities: *1)* collaboration, and *2)* trust management. The collaboration capability represents the ability of the Pilarcos ecosystem for dynamically building up business networks from the available business services offered by autonomous ecosystem members. The collaboration capability is supported by the qualitative trust management capability. Collaboration is considered as a functional capability while trust management is a qualitative capability of the Pilarcos ecosystem. The capabilities are realized by corresponding life cycles and supported by specialized infrastructure services (see [143, 87, 85, 133]).

The Pilarcos capabilities, their realizing life cycles and their inter-relationships are modelled using the Ecosystem capability specification model kind; the resulting model is illustrated in Figure 5.1. The capabilities and the collaboration coordination life cycle are contained in an ecosystem domain named *PilarcosDomain*. The trust management coordination life cycle is contained in a sub-domain named *ReputationSystem*. The ecosystem domains are associated with a domain ontology and an engineering space that are described later in this section.

Pilarcos ecosystem life cycles

Collaboration capability and trust management capability are realized by collaboration coordination and trust management coordination life cycles. The life cycles

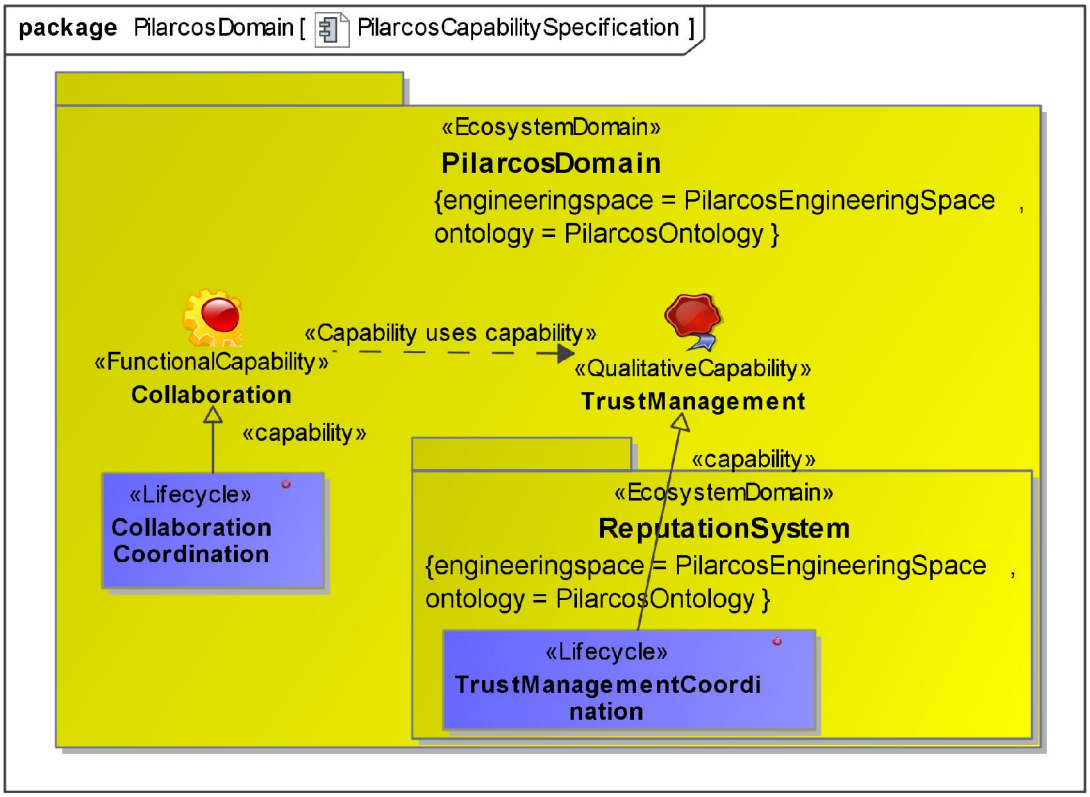

Figure 5.1: Specification of Pilarcos service ecosystem capabilities.

are modelled using the Life-cycle viewpoint of SEAF which comprises life-cycle structure specification and life-cycle behaviour specification model kinds. The corresponding models are presented and discussed below.

The Pilarcos collaboration coordination life cycle comprises four phases: *1) Population*, *2) Negotiation*, *3) Collaboration* and *4) Dissolution* [143], as illustrated in Figure 5.2. The population phase is associated with actors named *Initiator*, *Populator* and *ServiceOfferRepository*, the two latter representing infrastructure services. The population phase inputs a business network model, modelled with an artefact named *BNM*, and a set of available services. The *AvailableServices* artefact set represents the collection of all service offers available in the service ecosystem. The population phase produces two artefacts sets: *1) eContractTemplates* including a set of eContract templates and *2) InteroperableServices* including the set of service offers compatible with the BNM given as input for the population phase.

The negotiation, collaboration and dissolution phases involve only service providers. They consume and produce artefacts and artefact sets as illustrated in Figure 5.2. Finally, the *Dissolution* phase provides a collection of *ExperienceReports*, which are consumed in the *RSOperation* phase of the trust management coordination life cycle discussed below.

The Pilarcos collaboration coordination life-cycle behaviour defines the phasing between the life-cycle phases. The corresponding model is illustrated in Figure 5.3. The behavioural semantics of ecosystem life cycles and their transitions are domain-specific.

The trust management life cycle is contained in the *ReputationSystem* ecosystem domain, as illustrated in Figure 5.1. The life cycle includes three phases named *RSJoin*, *RSOperation*, and *RSLeave* [143]. The life-cycle phases enable service providers to join a reputation system, deliver reputation information, and leave a reputation system, correspondingly. The trust management coordination life-cycle structure is defined using the Life-Cycle structure specification; the resulting model is illustrated in Figure 5.4.

In the *RSJoin* phase the service provider joins a chosen reputation system and signs a reputation system contract. The signed reputation system contract provides the service provider with access to the knowledge contained in the reputation system [143, 133]. Other reputation system members may have a veto right to not allow the service provider to join the system [143].

In the *RSOperation* phase the service provider makes semi-automated trust decisions based on set trust-management policies [132]. To support the trust decisions, the service provider uses reputation information from the reputation system, as well as first-hand experiences about other service providers' behaviour in on-going service collaborations [132, 143]. The service provider shares the ex-

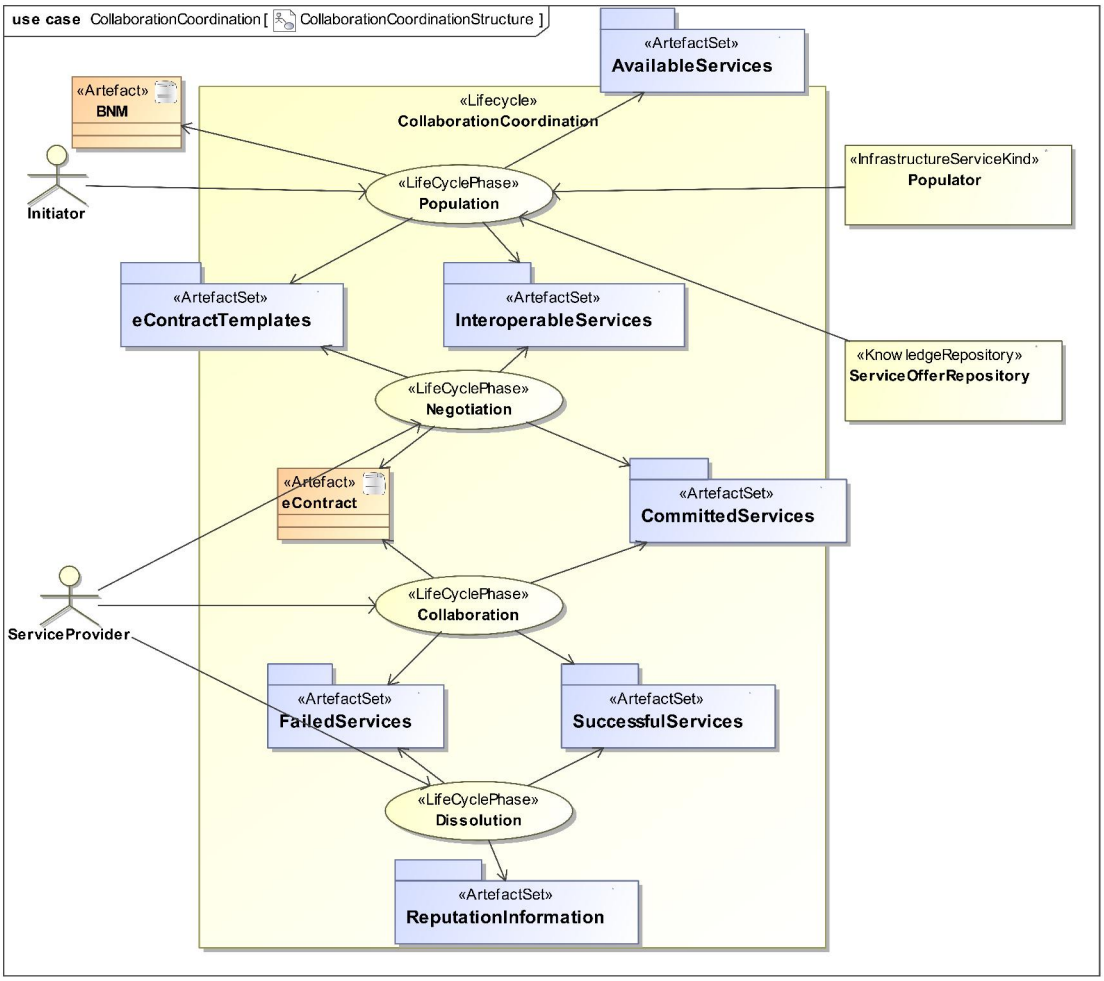

Figure 5.2: Specification of Pilarcos collaboration coordination structure.

periences with the other members via the reputation system in accordance to the reputation system contract [143].

Finally, a service provider may discharge itself from its reputation system membership; such behaviour takes place in the *RSLeave* phase [143].

The behaviour of the trust management life cycle is illustrated in Figure 5.5. After joining a reputation system, the service provider can operate in it until she decides to discharge her membership. Leaving a reputation system is possible after joining it.

Pilarcos service choreographies

Service choreographies are used in SEAF to define the roles, connections and interactions of service-based collaborations. Service choreographies are associated

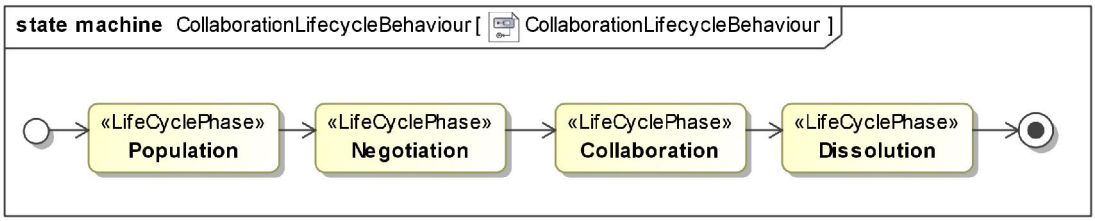

Figure 5.3: Specification of Pilarcos collaboration coordination behaviour.

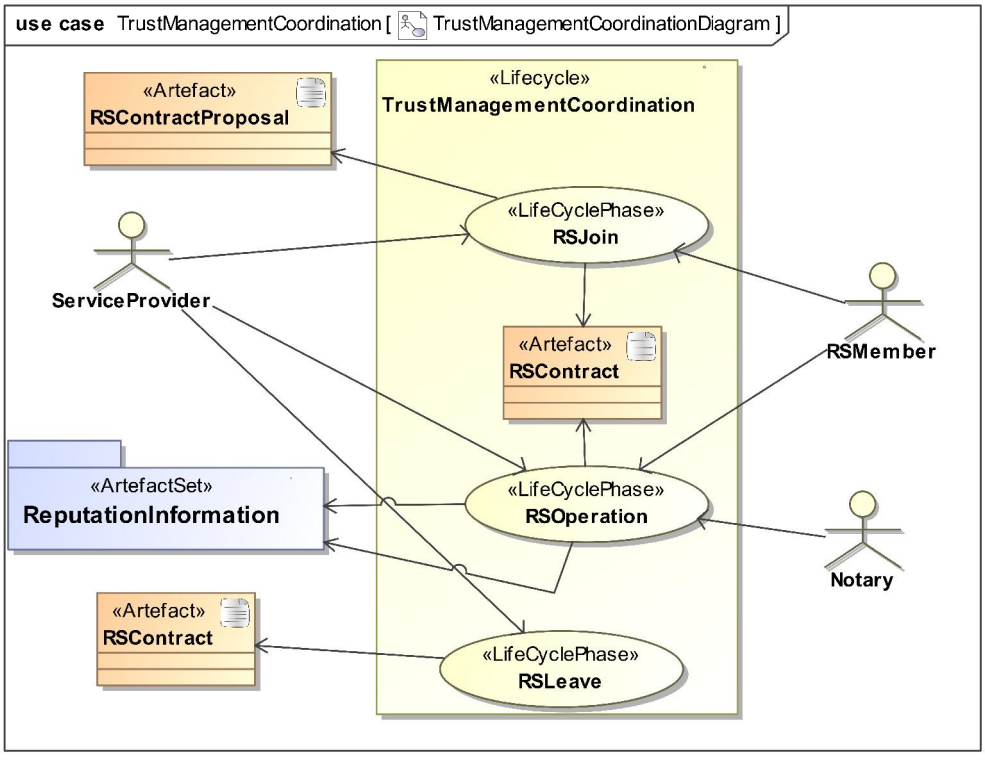

Figure 5.4: Specification of the trust management life-cycle structure.

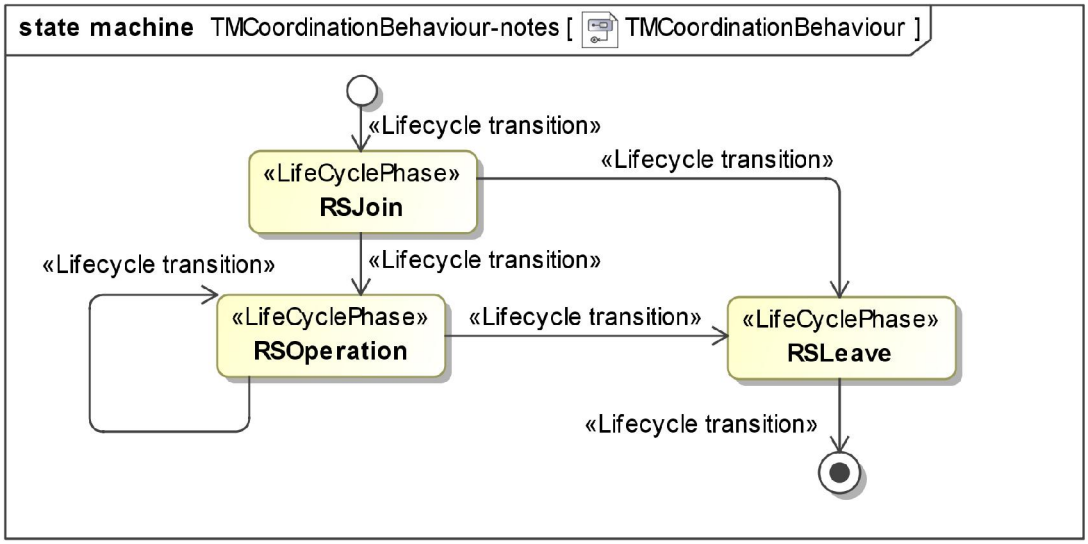

Figure 5.5: Specification of trust management life-cycle behaviour.

to life-cycle phases with the concept of service liaison. The service choreographies can be either static or dynamic, as discussed in Chapter 3. Static service choreographies represent life-cycle phase behaviour with roles and interactions determined during the design of the service ecosystem. Dynamic service choreographies represent life-cycle phase behaviour whose roles and interactions are determined via choreography descriptions published during the service ecosystem operation.

In the Pilarcos ecosystem, the population, negotiation and dissolution phases have static service choreographies. The population choreography defines interactions between a collaboration initiator, populator and service offer repositories. The negotiation choreography defines a multi-lateral negotiation protocol between service providers [85]. The dissolution choreography defines how service providers deliver their experiences about a collaboration to the reputation management system.

As an example of a static service choreography specification, the Pilarcos population choreography structure and behaviour is presented. In the Pilarcos service ecosystem, service-based business networks are established dynamically using the process of population [85], which involves three actors: initiator, populator, and service-offer repository. The initiator is a legal entity willing to establish a business network and initiating the population process. Populator and service-offer repository are infrastructure services provided by the Pilarcos service ecosystem.

The structure of the Pilarcos population service choreography is illustrated in Figure 5.6. The model is specified using the Service choreography structure

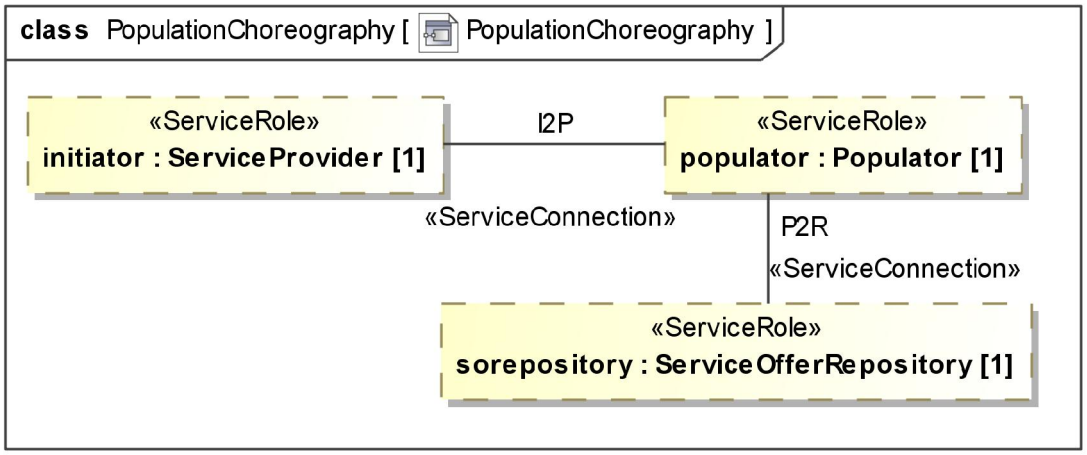

Figure 5.6: Specification of the Pilarcos population service choreography structure.

specification model kind, and it comprises service roles and service connections representing the service choreography roles, actors eligible to act in the roles, and relationships between the roles. Each service role is typed by an ecosystem actor: the *initiator* role is bindable only to actors of the kind *ServiceProvider* in the Pilarcos ecosystem, for example.

The static service choreography behaviour of the population process is illustrated in Figure 5.7. The population process begins with the initiator sending a pre-filled eContract template to the populator. The eContract template may provide initial values for the collaboration roles and features. The populator infrastructure service then queries a service offer repository about service offers that can be used for filling the roles of the business network model defined in the eContract template. The service offer repository returns a set of interoperable service offers to the populator. After receiving the set of interoperable services, the populator uses a constraint-satisfaction algorithm to come up with a set of eContract templates where each of the business network roles are filled with appropriate service offers [85]. The set of templates is then returned to the initiator. The initiator refines the eContract templates with multi-lateral negotiations [85]. If the negotiations are successful, an eContract is signed between the service providers offering the services, after which the actual collaboration can be started.

In the Pilarcos ecosystem the Collaboration phase is associated with dynamic choreographies. A dynamic service choreography prescribes schemata, or a metamodel, for modelling choreographies within a life-cycle phase. In the Pilarcos service ecosystem the choreography models associated with the collaboration phase are called business network architectures. The architecture model representing

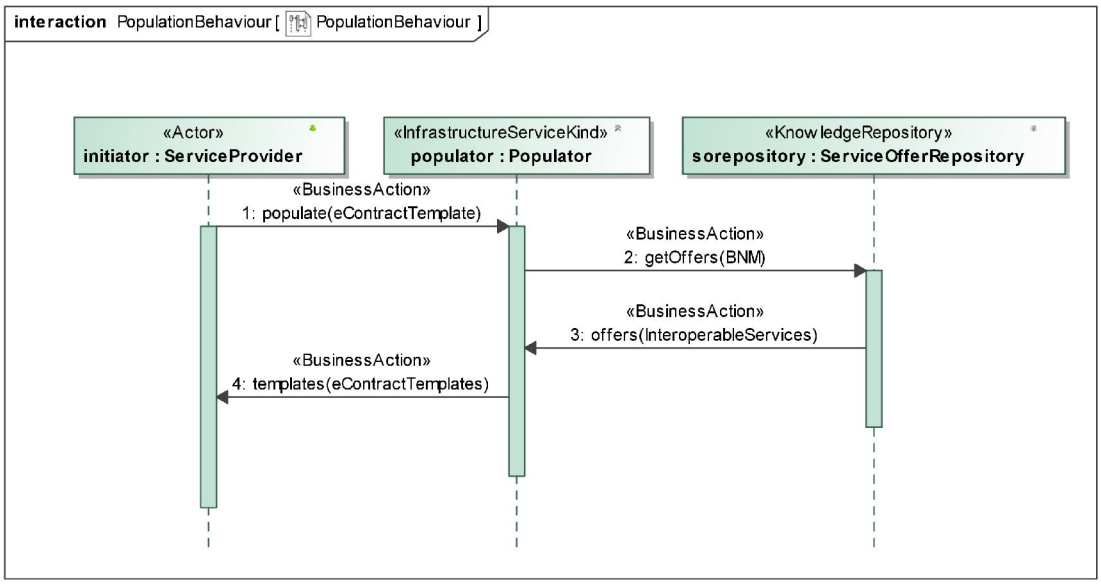

Figure 5.7: Specification of the Pilarcos population service choreography behaviour.

the Pilarcos business network architecture concept is illustrated in Figure 5.8.

The business network architecture concept is formalized using the *Dynamic-ServiceChoreography*-stereotype by providing corresponding tagged values with references to appropriate model elements. A service choreography is defined in the service ecosystem meta-model as a concept comprising a set of service roles, service connections, business actions, and business transactions. Business roles are associated with one or more ecosystem actors that are allowed to fulfill the corresponding kinds of roles. The denotation of a service role is prescribed by a set of service commitments. Each service commitment is associated with a debtor and creditor role, life-cycle product (e.g. an eContract or a business document in the Pilarcos ecosystem), and a service connection utilizable for discharging the commitment. Business actions are partially ordered events labelled with service commitments that define role interactions. Finally, business transactions compose individual business actions into coherent wholes that can be offered by business services.

The Pilarcos service ecosystem defines two kinds of business transactions, represented in the model with elements *BusinessTransaction* and *Epoch*. One or more business transactions are provided for each business network architecture for structuring the business actions in coherent wholes. Typically a business network architecture defines a single epoch which includes all business actions in the

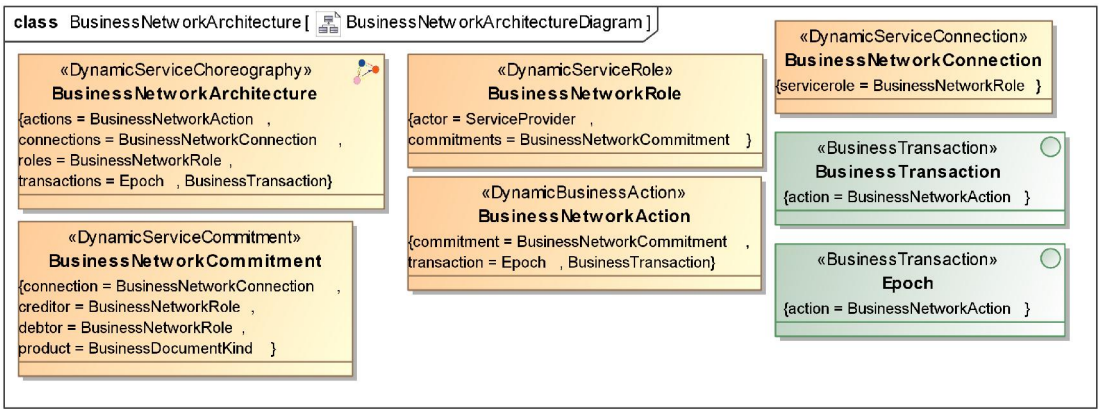

Figure 5.8: Specification of the Pilarcos business network architecture.

choreography. Pilarcos epochs denote periods of activity where the structure of the eCommunity is stable [85]. Separate epochs can be used for breach recovery or otherwise well-limited activity with different sets of roles still progressing the work of the eCommunity [85].

Pilarcos domain ontology

The Pilarcos domain ontology formalizes the conceptual framework presented in Section 5.2.1. The domain ontology includes specifications of concepts representing legal entities, services and contracting concepts. These foundational concepts of the domain ontology and their SEAF-based specifications are presented in the following.

In the Pilarcos framework legal entities are categorized in three groups: individuals, organizations, and enterprises. This categorization is specified with the concept generalization hierarchy between legal entity kinds as illustrated in Figure 5.9. The entity kinds and entities are dynamic in the Pilarcos service ecosystem: for example new kinds of organizations can be introduced to the ecosystem during its operation. Every legal entity kind is subject to legislation and reputation models; these cooperative features are dynamic in nature.

Pilarcos entities represented by the concept of *PilarcosLegalEntity* conform to a single legal entity kind, and are associated with conformance point specifications. Each conformance point specification represents means for assessing the conformance of the legal entity with respect to operational policies (*PolicyConformancePoint*), business rules (*BusinessModelConformancePoint*), and legislative acts (*LegislationConformancePoint*).

The dynamic conformance point specifications associated with Pilarcos legal entities are illustrated in Figure 5.10. Conformance points are cooperative

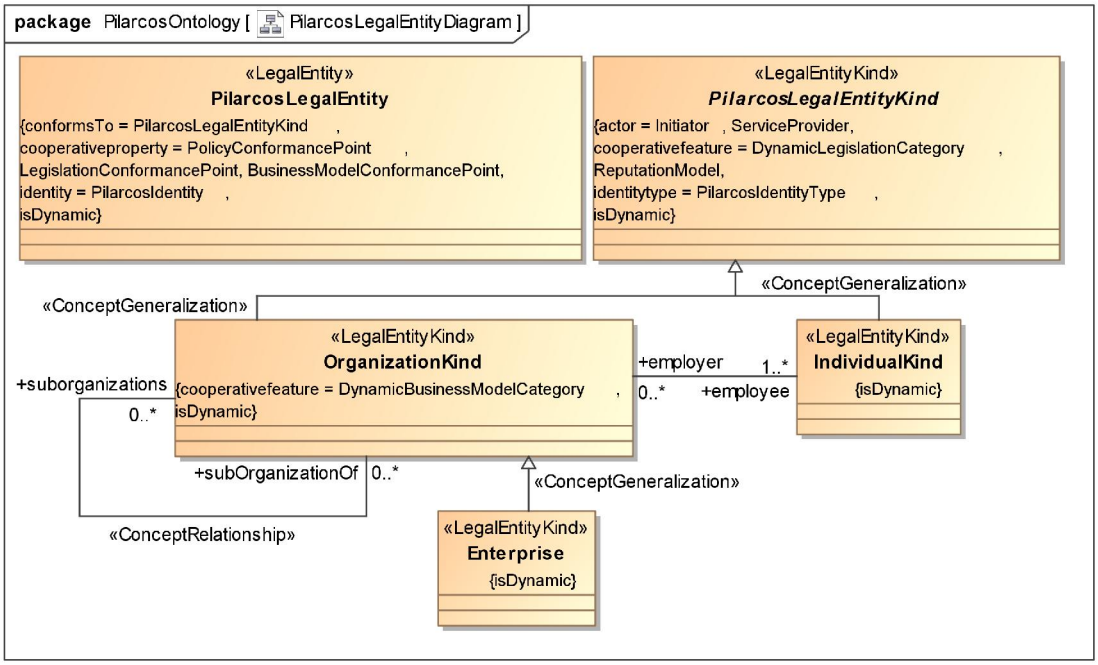

Figure 5.9: Specification of Pilarcos legal entities

properties that conform to corresponding cooperative features, as defined in the service ecosystem meta-model. The cooperative features, i.e. business models, policy frameworks and legislation, are dynamic in the Pilarcos framework. The evidence for conformance claims are provided in the Pilarcos service ecosystem by the *DischargeBusinessAction* elements which represent activities specified in Pilarcos business services.

The service entities in the Pilarcos service ecosystem comprise categories for business and component services. The service entities and their kinds are dynamic in the Pilarcos framework, i.e. the ontology of services can be extended during ecosystem operation with both new service kinds and their instances. The Pilarcos service entities are illustrated in Figure 5.11.

Business service kinds in the Pilarcos ecosystem are represented by the *PilarcosBusinessServiceKind* specification with corresponding associations defined for exposing business transactions and identifying the service role. Business services are represented by the element named *PilarcosBusinessService*; commitments laid by service choreographies are discharged with *DischargeCommitment* elements that are specified as part of the business service intension. The *DischargeCommitment* element is a static business service property which discharges business actions defined in service choreographies (*BusinessNetworkAction*) with service operations (*ServiceOperation*) defined in Pilarcos component service intension.

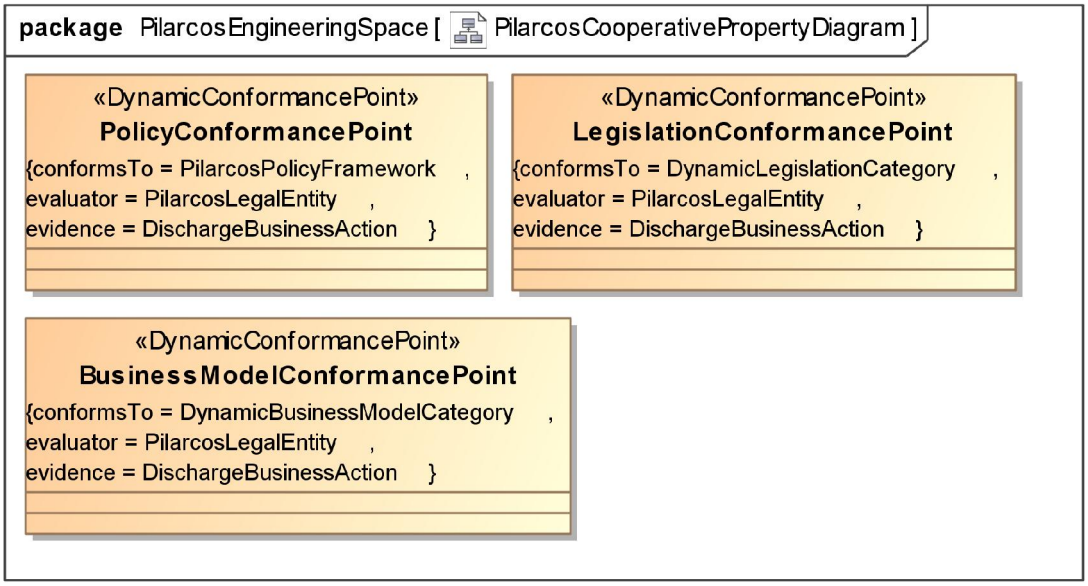

Figure 5.10: Specification of Pilarcos conformance points.

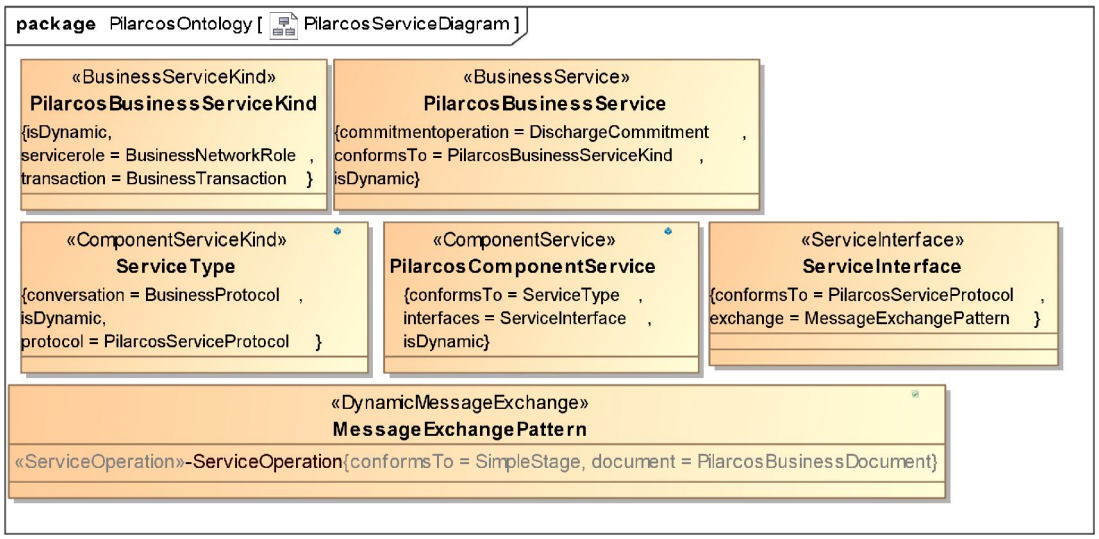

Figure 5.11: Specification of Pilarcos service entities.

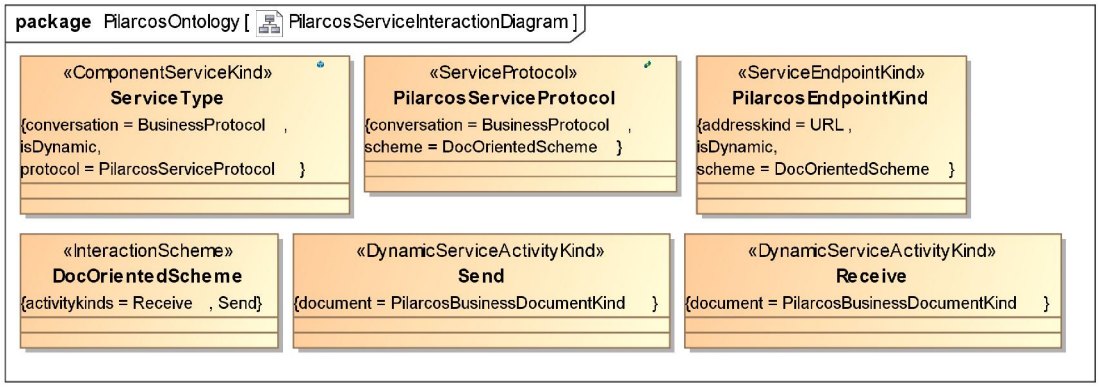

Figure 5.12: Specification of Pilarcos service interactions.

Component services and their kinds are represented in the Pilarcos service ecosystem architecture with by elements of *PilarcosComponentService* and *ServiceType*, correspondingly. Both component services and their kinds are dynamic in the Pilarcos framework. A *PilarcosComponentService* conforms to a *ServiceType*, and exposes message exchange patterns through a service interface specified by element *ServiceInterface*. The *ServiceInterface* is defined as part of the Pilarcos component service intension; it conforms with *PilarcosServiceProtocol* and realizes exchanges with *MessageExchangePattern*s. The communication paradigm followed in the Pilarcos framework is document-oriented. Document-orientation is reflected by the *MessageExchangePattern* specification which defines a single service operation with a payload as defined by a business document specification.

Component service kinds in the Pilarcos service ecosystem architecture are specified with the element named *ServiceType*. A *ServiceType* delivers behavioural capabilities as specified by the *PilarcosServiceProtocol* service protocol illustrated in Figure 5.12. The service protocol utilizes a document-oriented interaction scheme to realize behavioural pattern as specified by the *BusinessProtocol* element. The document-oriented interaction scheme is defined as comprising *Send* and *Receive* service activity kinds, as illustrated in Figure 5.12.

The service conversations delivered by Pilarcos component services are defined with the *PilarcosServiceProtocol* element illustrated in Figure 5.13. The service conversations in the Pilarcos framework comprise *BusinessProtocolStage* elements. A *BusinessProtocolStage* is further classified into simple and choice stages. Simple stages represent atomic message-sending and reception activities while choices are composite stages that represent mutually exclusive alternatives between a collection of business protocol stages. Business protocol *Choice* is an abstract element with two concrete children: a *Branch* corresponds to an input

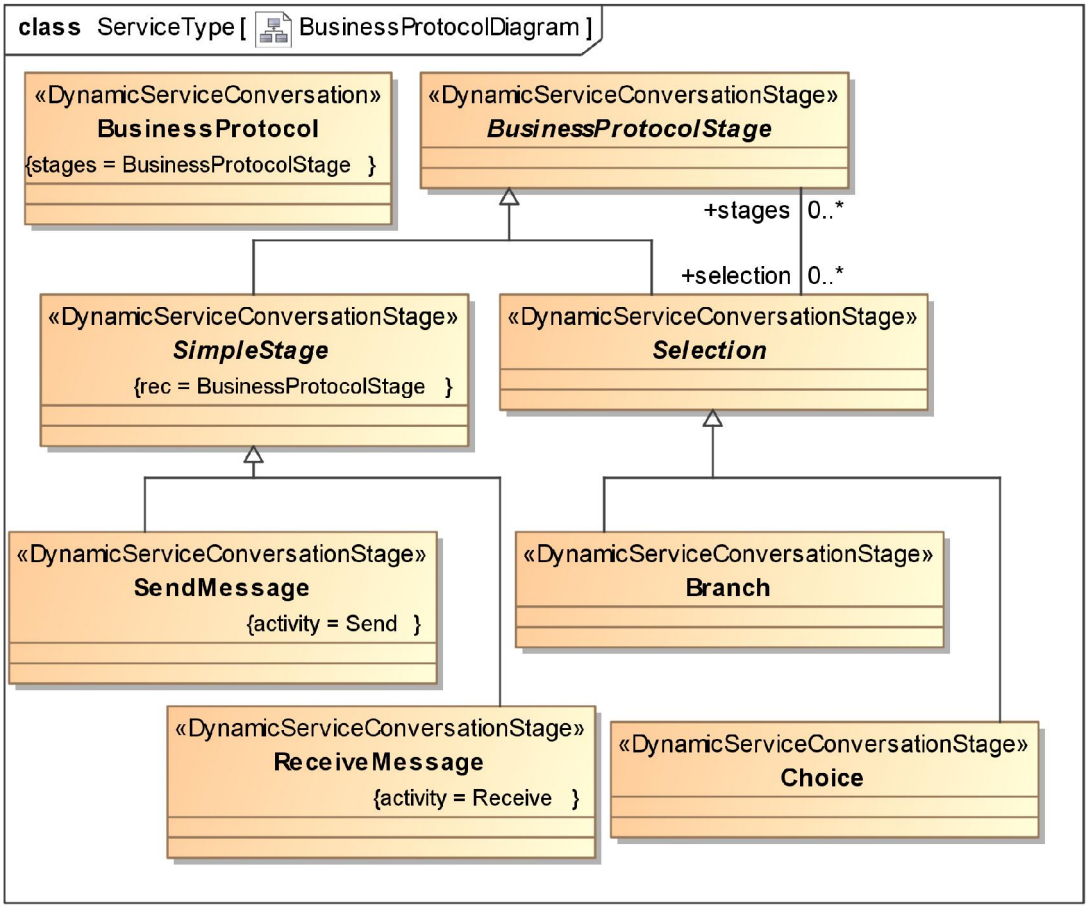

Figure 5.13: Specification of Pilarcos business protocols.

guarded choice while a *Selection* represents its dual operation of consisting of alternate output activities. Consequently, the business protocols define behavioural patterns that align with the syntax of session types [63, 55, 162, 23] which underlies the notion of Pilarcos service types [139, 134, 135].

The Pilarcos contracting concepts include business network architectures, business network models, eContracts and service offers [87, 85]. These artefacts are represented in the Pilarcos ecosystem architecture by the elements of *BusinessNetworkArchitecture* (discussed above and illustrated in Figure 5.8), *PilarcosLiaison*, *PilarcosContract* and *PilarcosServiceOffer*, correspondingly. In the following, the three latter concepts included in the Pilarcos domain ontology are described.

The elements defined in business network architectures are utilized in Pilarcos business network models (BNM) [87]. A BNM composes a set of service roles and connections to represent a business scenario within the service ecosystem. The Pilarcos business network models comprise [87]:

- A set of business network architectures associated with
  - role assignment rules for guiding the populator to retrieve suitable service offers from the service offer repository
  - a recovery process to be used in case of contract breaches.
- role association requirements expressing how a business service is required to simultaneously act in named roles; this allows functional slicing of the model for reuse and design purposes;
- policy rules overarching the business network model providing invariants that need to be fulfilled;
- breach recovery process name and type (type for example "dissolving", "restarting", or "continuable with sanctions")

The Pilarcos BNM is modelled using the concept of *PilarcosLiaison*, as illustrated in Figure 5.14. The specification defines the structure and contents of Pilarcos liaisons, and is declared as a representation of the BNM artefact specified in the Pilarcos life-cycle model. The *PilarcosLiaison* is a dynamic concept which defines that in the *Collaboration* phase of the Pilarcos collaboration coordination life-cycle service choreographies conforming with the *BusinessNetworkArchitecture* specification are used. Moreover, the *PilarcosLiaison* declares that liaisons in the Pilarcos framework are subject to five kinds of feature bindings: *BNMFeatureBinding*, *PolicyFrameworkBinding*, *EFFBinding*, *EpochTransitionBinding* and *BusinessRoleAssociationBinding*. These feature bindings are specified in the Pilarcos binding model which is discussed later in this section.

The Pilarcos service contract is defined by the *PilarcosContract* model element in Figure 5.14. *PilarcosContract* specification defines a service channel to be used (*PilarcosServiceChannel*), a commitment operation to discharge service choreography commitments (*DischargeBusinessAction*), a liaison to follow (*PilarcosLiaison*). Moreover, the specification declares bindings for associating endpoints (*EndpointBinding*) and service ports (*PilarcosPortBinding*) that enable structured service interactions in the Pilarcos framework.

The service channels in the Pilarcos framework are specified as defined in the *PilarcosServiceChannel* element. A Pilarcos service channel comprises a set of channel phases (*ChannelPhase*) and binding ports specified in intensional definitions of Pilarcos channel types and service offers, correspondingly. The intensional definition of *PilarcosServiceChannel* provides specifications for endpoint bindings and service links utilizable in the Pilarcos service ecosystem.

The commitment operations and port bindings in service contracts are taken from service offers published by service providers. The service offers of the Pilarcos framework are defined by the *PilarcosServiceOffer* element illustrated in

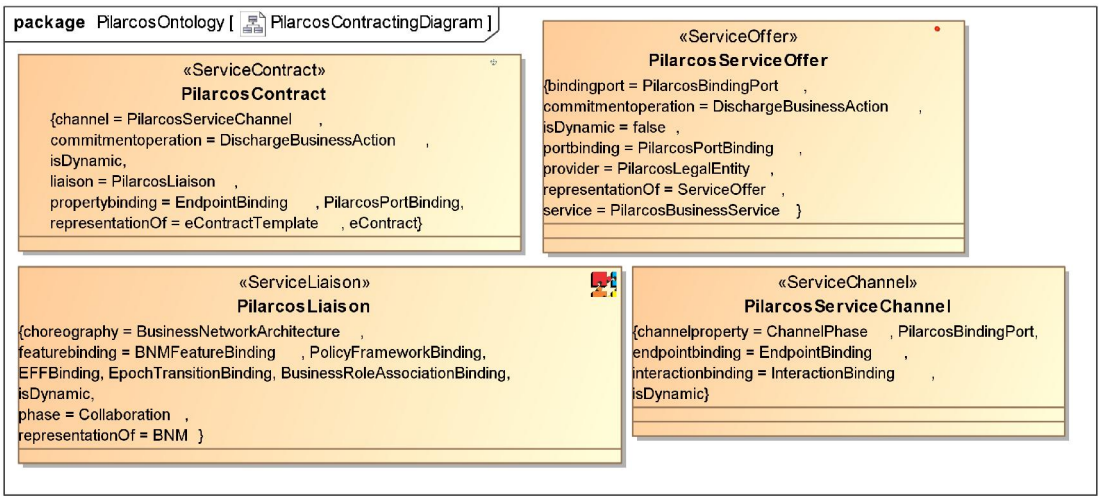

Figure 5.14: Specification of Pilarcos contracting concepts.

Figure 5.14. A service offer in this architecture framework is defined as a concept which associates a business service with a legal entity. Moreover, a service offer specifies service ports as sets of service operations and their bindings to service endpoints, and commitment operations provided by the business service to fulfill the commitments defined in a service choreography.

It should be noted that concepts in SEAF domain ontologies can be declared as representations of multiple life-cycle artefacts. Such one-to-many relationship between concepts and life-cycle artefacts is needed for encoding different roles of knowledge in service ecosystem life cycles. This enables formalization of context- and situation-specific interpretations for concepts and variable consistency criteria for their intensions. For example, the *PilarcosContract* concept in Figure 5.14 represents two life-cycle artefacts, *eContractTemplate* and *eContract*, which have different roles and consistency criteria in the collaboration coordination life cycle.

Pilarcos engineering space

The Pilarcos engineering space provides a definition of the binding model used in the service ecosystem. A binding model includes specifications for bindable features and feature bindings utilized in the ecosystem. Essentially, the binding model declares what kinds of features are bindable (dynamically) to the different contractual subjects of the service ecosystem. Engineering methods, product models or other elements of engineering spaces are not defined using the UML-based modelling tool. Instead, default versions of these service ecosystem architecture model elements are generated automatically by the model transformations, as discussed in Section 4.2.

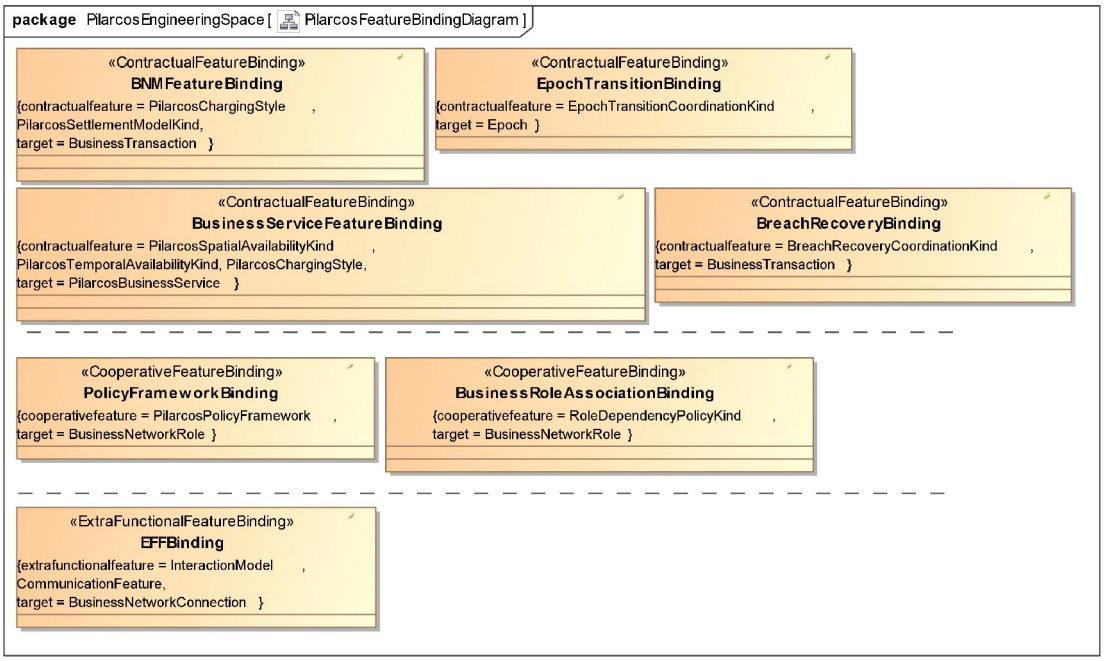

Figure 5.15: Specification of Pilarcos feature bindings.

The Pilarcos ecosystem defines four contractual features, two cooperative features, and one category of extra-functional features. The corresponding binding specifications are illustrated in Figure 5.15. Each feature binding specification declares the source (one or more features) and target (an ecosystem element) of the binding and the purpose of the binding specification.

The *BNMFeatureBinding* element is used for binding charging style and settlement model kinds to business transactions in the Pilarcos service ecosystem. Charging style and settlement model kinds are dynamic contractual features represented by the elements of *PilarcosChargingStyle* and *PilarcosSettlementModelkind*, correspondingly. Dynamic features for declaration of spatial and temporal availability kinds, as well as charging styles, can be bound to Pilarcos business services, as modelled by the *BusinessServiceFeatureBinding* element in Figure 5.15. Specifications of *EpochTransitionBinding* and *BreachRecoveryBinding* represent means for declaring processes for epoch transitions [85] and business transaction recovery as bindable features. As a consequence, service providers may dynamically negotiate about appropriate epoch transition and breach recovery proceedings during the Pilarcos collaboration coordination life cycle.

The Pilarcos service ecosystem architecture declares two binding specifications that address cooperative features, namely *PolicyFrameworkBinding*s and *BusinessRoleAssociationBinding*s, as illustrated in Figure 5.15. The *PolicyFrame-*

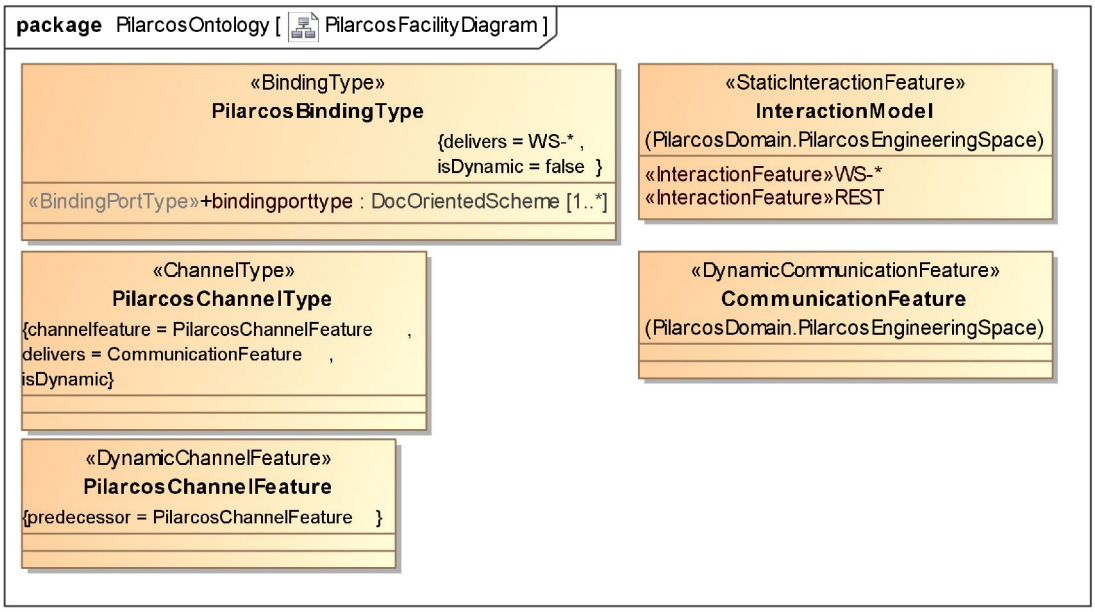

Figure 5.16: Specification of Pilarcos cooperation facilities.

workBinding defines a binding between Pilarcos policy frameworks and business network roles. Policy frameworks in the Pilarcos framework are considered as dynamic collections of operational policies. The *BusinessRoleAssociationBinding* defines requirements for participation of entities in a business network model. A Pilarcos role association requirement expresses how a business service is required to simultaneously act in named roles [87]. For expressing such dependencies the dynamic policy framework named *RoleDependencyPolicyKind* is used, which comprises a collection of role association rules. Each role association rule defines a binary relationship between two service roles. The relationships denote either a requirement for coincidence or mutual exclusion between roles.

Finally, the Pilarcos binding model provides specifications for extra-functional feature bindings. The *EFFBinding* element defines that business network connections can be associated with extra-functional features named *InteractionModel* and *CommunicationFeature*, as illustrated in Figure 5.15.

The *InteractionModel* in the Pilarcos service ecosystem architecture is a static extra-functional feature which specifies two alternatives for service interaction: WS-* [169] or REST [56], as illustrated in Figure 5.16. The *CommunicationFeature* element represents a dynamic communication feature which can be used for declaring requirements for secured communication, for example. The communication features in the Pilarcos framework are delivered by channel type declarations that conform with the *PilarcosChannelType* specification.

5.3 Evaluation results

In this section the feasibility, applicability and utility of the overall approach are assessed. The feasibility of the approach is validated in Section 5.3.1 by demonstrating the implementability of the developed artefacts. The applicability of the approach is evaluated in Section 5.3.2 assessing the practical suitability of the approach for supporting systems engineering life cycles of service ecosystems.

The foundational utility of the approach is facilitation of service ecosystem sustainability. Requirements for service ecosystem viability and principles for supporting the continuation of viability, which were discussed in Section 2.3, provide the evaluation criteria for the utility and added value of the approach. To assess the utility, we demonstrate: *a)* that the approach addresses all relevant service ecosystem stakeholder concerns, and *b)* that the approach enables support for continuation of service ecosystem viability. These assessments are presented in Section 5.3.3 and Section 5.3.4, correspondingly.

5.3.1 Validating the feasibility of the approach

The feasibility of the model-driven approach for service ecosystem engineering defined in this dissertation is implied by the technical implementability of its individual components: conceptual model, service ecosystem meta-model, SEAF, and the model-driven methodology. In the following, the feasibility of these constructs is discussed.

The feasibility of the conceptual model for service ecosystems is addressed with respect to two complementary aspects: *1)* completeness, and *2)* consistency. Completeness of the conceptual model here means that sufficient vocabulary is provided for expressing the characteristics of service ecosystems. Consistency of the conceptual model means that the terms included do not conflict with each other, and that there are no redundant terms in the vocabulary.

The conceptual model for service ecosystems is sufficiently complete for representing the concepts of state-of-the-art service ecosystems. This was validated by the case study presented in Section 5.2 where the Pilarcos framework was modelled using the vocabulary of the conceptual model. The consistency of the conceptual model is validated by its rigorous formalization in the service ecosystem meta-model presented in Chapter 3. The formalization as a meta-model makes explicit the relationships between concepts and their properties, which has enabled analysis of the dependencies between concepts during the design of the meta-model. There are no redundant, or orphan, terms in the vocabulary; this has been validated with the meta-model.

Feasibility as in technical implementability is not an issue when considering the service ecosystem meta-model, since the meta-model was constructed as an

technological artefact from the beginning. The meta-model has been designed using a commercial modelling tool. The UML model of the service ecosystem meta-model comprises over 260 classes. The UML model was then exported to an EMF Ecore [39] model for enabling the implementation of the model-driven methodology for service ecosystem engineering.

The feasibility of SEAF has been validated by the UML-based implementation. Realization of the model-driven methodology necessitated implementation of SEAF over an UML tool; the technical implementability of SEAF is thus validated. Moreover, the model-driven methodology gave evidence of the usability of SEAF as part of a model-driven service ecosystem engineering process.

Feasibility of the model-driven methodology for service ecosystem engineering has been validated by implementing a series of model transformations. The model transformations, as described in Section 4.2, first refine the UML-based architecture descriptions produced with SEAF. The refined architecture descriptions are realized as EMF models [39]. The refined models are then consumed by a family of model transformations which generate technology-specific engineering artefacts, as described in Section 4.2. The family of model transformations generate Java classes and interfaces, EMF meta-models representing the abstract syntaxes of domain-specific languages, and OSGi [120] descriptions. The enabling efficient implementation of corresponding knowledge repositories. The selection of technology-specific model transformations validate that service ecosystem models can be utilized for enabling efficient, model-driven engineering of service ecosystems.

5.3.2 Evaluating the applicability of the approach

The applicability of the conceptual model is demonstrated by its practical usability for service ecosystem design and analysis. The case study conducted in Section 5.2 has demonstrated that the conceptual model is applicable for identification and decomposition of service ecosystem elements. Moreover, the conceptual model provides means for well-defined service ecosystem design by providing concepts such as capabilities, life cycles, and their relationships. The different conceptual hierarchies contained in the conceptual model, especially those for different service ecosystem entities and concepts, support service ecosystem analysis activities.

The applicability of the service ecosystem meta-model is demonstrated especially by the model-driven methodology for service ecosystem engineering, which was described in Section 4.2. Moreover, the design of SEAF presented in Chapter 4 and its implementation were aligned with the meta-model. The applicability of SEAF was validated with its usage in the Pilarcos case study presented in Section 5.2. The Pilarcos case study proved the practical suitability of SEAF for con-

structing service ecosystem architecture descriptions. The viewpoints included in SEAF provide structuring mechanisms for rigorous design and analysis of service ecosystem architectures.

The applicability of the model-driven methodology for service ecosystem instrumentation is demonstrated by production of proof-of-concept engineering artefacts. The selection of engineering artefacts that were produced by the model transformations represent typical elements needed for implementation of infrastructure services (e.g. Java interfaces and OSGi-based [120] deployment descriptors) and engineering tools (e.g. EMF-based [39] meta-models for a family of DSLs). The generated engineering artefacts enable efficient instrumentation of the Pilarcos service ecosystem using contemporary technological platforms, such as Java, OSGi [120] and EMF [39]. Evidently, with different kinds of technology-specific model transformations (e.g. different deployment descriptors instead of OSGi) service ecosystems can be efficiently instrumented over cloud computing platforms, for example.

5.3.3 Evaluating the support for ecosystem viability

Viability is achieved by supporting the operation of stakeholders and addressing their primary concerns. The service ecosystem stakeholders identified in this thesis were: *1)* service ecosystem provider, *2)* infrastructure provider, *3)* engineering tool provider, *4)* service engineer, *5)* service provider, and *6)* service consumer. The support for ecosystem viability is assessed below by considering the stakeholder concerns and how they are addressed in the approach.

From a service ecosystem provider perspective, the approach provides means for analysis and design of service ecosystems. Service-related concepts of SEAF cover a contemporary SOA reference model [154], as will be demonstrated in Section 5.4.1. Thus service ecosystem models produced with the approach can be further analysed with respect to other service-based systems (e.g. local SOA deployments or contemporary service-based cloud systems) aligning with the OASIS SOA-RM reference model [154]. Especially the conceptual model and SEAF enable service ecosystem providers to identify required service ecosystem capabilities and elements, e.g. infrastructure service or engineering methods, needed for delivering those capabilities. The service ecosystem meta-model enables formalization and automation of service ecosystem architecture activities. During service ecosystem operation explicitly available, formal architecture models provide support for governing the operation of service ecosystems [143].

From a service infrastructure provider perspective the approach developed in this thesis enables identification and design, as well as efficient production of infrastructure services. Means for identification and design of infrastructure services are addressed by the conceptual model and the architecture framework. The

model-driven methodology for service ecosystem instrumentation supports efficient production of infrastructure services. Efficiency is gained through utilization of model-driven engineering [144] practices. With respect to the operation phase of service ecosystems, the architecture models can be utilized for establishing interoperability between infrastructure services. The architecture models encode e.g. the dependencies between knowledge items and sharing of duties, as well as roles of infrastructure services. With application of model-driven methods, such as the model-driven methodology developed in this thesis, service infrastructure providers can establish interoperation more effortlessly.

Concerns identified for service engineering tool providers are addressed by the approach. The conceptual model and the architecture framework enable identification and design of engineering capabilities that are needed in a service ecosystem for delivery of component services. The identified engineering capabilities are formalized in service ecosystem architecture models, and thus efficiently shared to engineering tool providers for further analysis, feedback and implementation.

The model-driven methodology demonstrates that the approach supports efficient production and specialization of engineering tools and languages. The ecosystem architecture models are utilizable for generating abstract syntax definitions (i.e. meta-models) of a modelling language family for describing the intensional specifications of concepts. As part of the research evaluation, model transformations have been implemented in the Eclipse Modeling Framework [39] for producing meta-models for each intensional definition included in a service ecosystem architecture model. The abstract syntax definitions provide means for developing service ecosystem-specific modelling tools manually. The Eclipse Modelling Framework [39] can then be utilized for implementing model editors with graphical or textual concrete syntaxes for the generated meta-models.

Within more specialized domains, such as vendor-specific product lines, more specific and complex engineering tool artefacts can be generated. For example, concrete syntax definitions (e.g. UML profiles), modelling tool specification descriptions (e.g. user-interface configurations) and implementation components (e.g. Java-classes) could be generated from the ecosystem models. Following such an approach, the service engineering tool providers can establish model-driven software product lines (SPL) [33] for efficient tool production and specialization. Such a product line architecture can open both economies of scale (i.e. implementing tools for many ecosystems) and economies of scope (i.e. providing specialized tools for each ecosystem) for service engineering tool providers. A similar SPL-based approach would be applicable for other stakeholders, such as infrastructure service providers and service engineers as well.

With respect to service ecosystem operation, the service ecosystem architec-

ture models are utilizable for enabling interoperability and integration between different service engineering tools and languages. In this context, the service ecosystem architecture models are used as unifying, shared models. The processes and syntaxes of engineering tools and languages utilized locally in enterprises are mapped to the methods and abstract syntaxes defined in the service ecosystem architecture models. Model transformations can be utilized for such purposes.

For service engineers the approach developed in this thesis enables well-advised service engineering practices. As discussed in Section 2.3.1, well-advised service engineering refers to both correspondence of service engineering with the business models ("well-advised motivations") and pragmatics ("well-advised practices") of the service ecosystem. With respect to well-advised motivations, the approach provides a holistic framework which unifies business-level service choreographies with elements contributing to service engineering, such as different kinds of service and interaction features. In service ecosystems the service choreographies are based on the business networks and business models relevant in the corresponding business domain. The service engineers can utilize published service choreography descriptions for developing services that correspond closely to the needs of the underlying business models. Service development based on published service choreography descriptions provides well-advised motivations for service engineering. Moreover, utilization of domain-specific service description languages and specialized engineering tools decreases the semantic gap between service engineering and business engineering domains. The engineering viewpoint of SEAF addresses well-advised practices by providing means for specification of methods and tools required for instrumenting service engineering.

For service providers the approach provides expressive concepts for specification of service bundling and service contracting relationships. The service contracting concepts defined in the conceptual model and the service ecosystem meta-model enable exploitation of sub-contracting relationships (through commitment operations aligning with [150]), asymmetric contracts (e.g. definition of consumer-provider, in-sourcing and out-sourcing contracts), and introduction of value-adding operations by service providers. This facilitates efficient service delivery by enabling utilization of provider-specific competencies and business networks (e.g. sub-contracting relationships), autonomy over service bundling, as well as service provider differentiation.

From a service consumer perspective the approach provides means for enabling service identification and selection in open service ecosystems. The conceptual model and service ecosystem meta-model provides means for establishing ecosystem-specific service categorization, service ontologies and variable features in service offers. Service categorization is enabled through use of a two-level on-

tological modelling approach, where services conform to service kinds [135]. The extendability of the service ecosystem meta-model provides means for specifying domain-specific relationships and concepts, and thus ontologies, over the core service concepts. The feature category included in the conceptual model and formalized in the service ecosystem meta-model enables management of functional and non-functional features in open service ecosystems [141], and facilitates feature-based discovery and selection of services.

Table 5.1 summarizes the contributions of this work with respect to service ecosystem viability. Service ecosystem viability is enhanced by addressing the fundamental concerns of primary ecosystem stakeholders. In the table, each stakeholder concern identified in Section 2.2 is associated with an engineering artefact that addresses the corresponding concerns. While the approach developed in this thesis does not address service ecosystem operation, the service ecosystem architecture models can be utilized in the operation phase for addressing several stakeholder concerns. Thus, the approach addresses those concerns by enabling creation of service ecosystem architecture models.

5.3.4 Evaluating the support for continuation of viability

As discussed in Section 2.3.2, continuation of viability is guaranteed during the service ecosystem operation phase by supporting four fundamental service ecosystem objectives: *1)* efficient utilization of core competencies, *2)* opportunistic and flexible business networking, *3)* supporting progressive business environments, and *4)* efficient business decision making. These objectives are supported by the principles for continued viability which were identified, in conjunction with their implications, in Section 2.3.2. While the overall approach described in this thesis does not explicitly consider the service ecosystem operation phase, the support for continuation of viability enabled by the approach is discussed below with respect to the principles and their implications.

Two principles for supporting efficient utilization of core competencies in service ecosystems were identified in Section 2.3.2: ecosystem member autonomy, and dependability of service collaborations. The service ecosystem architecture models can be utilized for facilitating member autonomy. First of all, the architecture models can be used for compatibility analysis and integration between local engineering processes, methods and tools with the engineering spaces in service ecosystems. Secondly, the architecture models enable analysis and integration of business processes and policies with those of the service ecosystems. For example, analysis between local enterprise architectures (see e.g. [88]) and service ecosystem architecture models are enabled. Thirdly, the service ecosystem models are utilizable as shared, unified representations of services, features, and other service ecosystem concepts to overcome semantic interoperability problems

Stakeholder	**Name**	**Addressed by engineering artefact**
Ecosystem provider	Identification of required ecosystem capabilities and elements supporting the delivery of those.	conceptual model; SEAF
	Formalization of service ecosystem architecture descriptions.	service ecosystem meta-model
	Supporting service ecosystem governance.	*service ecosystem architecture models*
Infrastructure provider	Identification and design of required infrastructure services.	conceptual model; SEAF
	Efficient production of infrastructure services.	model-driven methodology
	Infrastructure service interoperability.	*service ecosystem architecture models*
Engineering tool provider	Identification of engineering capabilities required for component service production.	conceptual model; SEAF
	Efficient tool production and specialization.	model-driven methodology
	Tool and language interoperability and integration.	*service ecosystem architecture models*
Service engineer	Identification of engineering capabilities required for component service production (shared with engineering tool providers).	conceptual model; SEAF
	Well-advised service engineering.	*service ecosystem architecture models*
	Efficient component service production.	*service ecosystem architecture models*
Service provider	Service bundling.	conceptual model; *service ecosystem architecture models*
	Service contracting.	conceptual model; *service ecosystem architecture models*
Service consumer	Business service identification, location and selection.	*service ecosystem architecture models*
	Business service dependability.	*service ecosystem architecture models*
	Business service monitoring.	*service ecosystem architecture models*

Table 5.1: Addressing service ecosystem stakeholder concerns in the overall approach.

between local enterprise and ecosystem domains, that is, the service ecosystem architecture models enable interoperability approaches based on unification (see e.g. [138]).

For addressing the dependability of service collaborations, the explicit ecosystem architecture models can be analysed locally for possible conflicts or threats with respect to the organization's operation. Moreover, the architecture models can be utilized for model-driven production and configuration of the local communication and monitoring infrastructure, and model-driven deployment of services (see e.g. [4]).

The principle of dynamic business service collaborations was identified in Section 2.3.2 as the primary means for supporting opportunistic and flexible business networking in service ecosystems. Dynamism required for enabling opportunistic and flexible business networking is addressed in the concept definitions of domain ontologies. With the structures defined in the service ecosystem meta-model, each concept can be declared as either static or dynamic. Static concepts can not be instantiated during service ecosystem operation; they can be utilized to represent for example immutable services in Infrastructure-as-a-Service cloud environments. Dynamic concepts are utilizable for expressing extendable service ecosystem knowledge.

To enable dynamic business service collaborations, means for on-demand establishment of business service collaborations is needed. Towards this purpose, the conceptual model and the service ecosystem meta-model include elements for describing and managing dynamic service choreographies. The mechanisms and processes for achieving dynamic business service collaborations are domain-specific. For example in the Pilarcos framework the specifics of dynamic business service collaboration establishment are addressed in the collaboration coordination life cycle, as described in Section 5.2.

Dynamic business service collaborations also require means for loose coupling of business services. The conceptual model and the service ecosystem meta-model are designed from the ground-up for the purpose of enabling late binding and late encapsulation. This design principle is reflected for example in the categorization of features and services, different behavioural elements, and the design of the binding model elements.

For supporting progressive business environments, two foundational principles were identified in Section 2.3.2: *1)* open model of ecosystem participation, and *2)* extendability of ecosystem knowledge bases. The open model of ecosystem participation is addressed in the overall approach (conceptual model, meta-model, and the model-driven methodology) by mechanisms that enable declaration of dynamism with respect to service ecosystem members (i.e. legal entities) in service ecosystem models, as discussed above.

Extendability of ecosystem knowledge bases is addressed in the meta-model by providing mechanisms that allow service ecosystem specialization, and (dynamic) generalization and instantiation hierarchies. Service ecosystem specialization is addressed in the service ecosystem meta-model by providing means for representation of domain-specific extensions of the meta-model. The concept intensions and ecosystem features can be extended with domain-specific properties by exploiting the *ExtendableElement*-modelling element. Generalization and instantiation hierarchies are enabled with two modelling constructs in the meta-model: *GenerizableElement* and *InstantiatableElement*, as discussed in Chapter 3, for generalization hierarchies and deep instantiation [7] of intension model elements, correspondingly.

Supporting efficient business decision making is founded on two principles, as identified in Section 2.3.2: *1)* predictability of joining and operating in service ecosystems, and *2)* governability of service ecosystem operation.

With respect to predictability, the approach enables rigorous architectural design and analysis of service ecosystems. In the case study presented in Section 5.2 SEAF was successfully utilized for defining the concepts of the Pilarcos framework, analysing the nature and properties of required artefacts, and design of the Pilarcos service ecosystem architecture.

The added value and utility with respect to the contemporary service-oriented architecture engineering approaches is the holistic approach, and the rigour of design and analysis the SEAF provides. During the case study several concepts of the Pilarcos framework were defined more formally than was possible before. In the Pilarcos case presented in Section 5.2, application of the conceptual model revealed gaps between Pilarcos concepts related to contracting, epochs, and the role of contractual and cooperative features in specification of business network models. The conceptual model for service ecosystems was successfully used for formalizing and clarifying these relationships between the Pilarcos concepts.

Especially, the Pilarcos binding model was formalized as part of the case study. With utilization of SEAF a few conceptual gaps were identified in the Pilarcos framework related to epochs and their transitions, business transactions and recovery processes, as well as role assignment rules. The nature and positioning of these Pilarcos concepts was quite implicit before. With the analysis power of SEAF the characteristics of these concepts were identified and consequently formalized as kinds of contractual and cooperative features bindable with different kinds of business transactions and business roles.

Service ecosystem architecture models are utilizable for supporting model-driven interoperability and governance in service ecosystems. For model-driven interoperability approaches the service ecosystem meta-model provides means for creation of service ecosystem architecture models; the architecture models can be

utilized for establishing interoperability through unification and federation (see e.g. [137, 140]). Semantic interoperability is addressed by specification of the domain ontology model which formalizes the conceptual model for service ecosystems. The domain ontology model definition of holistic, unifying ecosystem models which are utilizable for establishing semantic interoperability in service ecosystems. The domain ontology model is extended by service ecosystem architectures to represent the domain-specific versions of the general concepts. Specifications of ecosystem life cycles, choreographies (static or dynamic) and contracting concepts in ecosystem models support pragmatic interoperability [6] in service ecosystems.

Finally, the service ecosystem architecture models enable collaborative model-based governance practices in open service ecosystems [143]. In the Pilarcos framework shared service ecosystem architecture models provide consistency and conformance criteria to be maintained by the knowledge repositories which comprise part of the governance infrastructure in the Pilarcos framework. Such coherent infrastructure specifications are invaluable for the enabling collaboration establishment processes and their governance in open service ecosystems with autonomic participants.

The above discussion about supporting the principles of continued viability is summarized in Table 5.2 and Table 5.3. As can be seen in the tables, all principles that are required for establishing continuation of viability, as discussed in Section 2.3.2, are supported. As a consequence, the approach developed in this thesis delivers additional value for service ecosystem stakeholders by enabling means for efficient utilization of core competencies, opportunistic and flexible business networking, supporting progressive business environments, and efficient business decision making.

As a summary of the evaluation results, the overall approach developed in this thesis enables facilitation of service ecosystem sustainability. The viability of service ecosystems is supported by the approach by addressing all the foundational concerns of service ecosystem stakeholders. Moreover, the approach enables support for continuation of viability during service ecosystem operation.

5.4 Comparison to related work

The challenge of designing, instrumenting and managing service ecosystems has been identified in academia, the standardization community and industry, and is being addressed by different approaches. Moreover, these challenges are considered fundamental for enhancing the competitiveness of European service-based business: they are being addressed in the European Commission FP7-ICT Programme for Research and Development by objective 1.2 [52] which takes steps

Principle	Implication	Supporting usage scenario
Ecosystem member autonomy	Members can follow their established practices to operate in the ecosystem.	Explicit architecture models enable compatibility analysis and integration of local engineering processes, tools and methods with those of the ecosystem.
	Facilities are provided for supporting effortless integration and compatibility of local practices with ecosystem capabilities and policies.	Explicit architecture models enable compatibility analysis and integration of local business processes and policies with those of the ecosystem.
	Interoperability service utilities are provided for identifying and handling interoperability problems between autonomously provided services, and features and policies associated with them.	The architecture models can be utilized as shared, unified representations of the services, features, and other concepts to overcome semantics interoperability problems.
Dependability of service collaborations	Ecosystem provides means for assessing, establishing and monitoring dependability of service collaborations. In open service ecosystems this involves especially the aspects of trust, privacy and security.	Explicit ecosystem architectures can be analysed for possible threats to organization's operation; The architecture models can be utilized for model-driven production and configuration of local communication and monitoring infrastructure.
Dynamic business service collaborations	Ecosystem provides means for establishing business service collaborations on demand.	The conceptual model and the service ecosystem meta-model include elements for describing and managing dynamic service choreographies.
	Mechanisms are provided for loose coupling of business services; this includes especially late binding and late encapsulation of services.	The conceptual model and the meta-model are designed for the purpose of enabling late binding and late encapsulation.

Table 5.2: Supporting the principles for continued viability.

Principle	Implication	Supporting usage scenario
Open model of ecosystem participation.	Autonomic entities may join and leave ecosystems as they wish. Ecosystem members may take any role available in the service ecosystem, in accordance with ecosystem policies.	The service ecosystem architectures can include dynamic entities and entity kinds; Dynamic service roles are supported with dynamic service choreographies.
Extendability of ecosystem knowledge bases.	New kinds of services, cooperation facilities and business networks can be introduced on demand.	The conceptual model and the meta-model include elements and mechanisms for service ecosystem specialization and (dynamic) evolution of service ecosystem knowledge.
Predictability of joining and operating in service ecosystems.	(Potential) ecosystem members are provided with methods for analyzing the maturity and feasibility of service ecosystems, and their capabilities and requirements with respect to local enterprise and service architectures.	Explicit service ecosystem architecture models can be analysed with appropriate domain-specific methods.
Governability of service ecosystem operation.	Local governability: Ecosystem members are able to govern their local service-oriented architectures in accordance with ecosystem requirements and capabilities.	Explicit service ecosystem architecture models can be utilized for architectural cross-analysis with respect to local enterprise architectures, for example.
	Global governability: Ecosystem members are provided with means for collaborative governance of service ecosystem operation.	Explicit service ecosystem architecture models enable collaborative governance practices [143].

Table 5.3: Supporting the principles for continued viability; continued.

towards the vision of “Internet of Services”.

In this section a comparison to related work is given. The approach developed in this thesis is founded on service orientation. Several standardization efforts related to service orientation have been conducted by organizations such as W3C (e.g. web services technologies), OMG (e.g. modelling languages), ISO (e.g. principles and frameworks for service-oriented systems) and OASIS (e.g. reference models and architectures for service orientation). Since the approach taken in this thesis is technology agnostic, and concentrates on conceptual and architectural aspects of service ecosystems, the focus of the comparison is on related work considering reference models and architectural frameworks for service-oriented systems engineering.

In Section 5.4.1 a comparison between the conceptual model developed in this thesis and the OASIS SOA-RM [154] is given. OASIS SOA-RM provides a definition of a reference model for service-oriented architectures. The reference model comprises a set of concepts and their relationships, and a set of conformance criteria.

Section 5.4.2 discusses the differences and commonalities of this work to the RM-ODP standardization [90] which is developed and maintained under ISO [69, 67, 68, 70]. The RM-ODP standardization defines an architecture framework for open distributed systems. The primary concepts, viewpoints and the overall approach are discussed in the comparison.

The architecture framework developed in this thesis shares commonalities with so-called enterprise architectures. An enterprise architecture describes a coherent set of principles, methods and models for guiding the operation and transformation of an enterprise [88]. In Section 5.4.3, the commonalities and differences between enterprise architecture frameworks and SEAF are identified.

The vision and challenges underlying the research efforts resulting in this thesis align with the current research strategy of the European Commission FP7-ICT Programme for Research and Development. Related research projects under the ICT research theme’s Challenge 1: *Pervasive and Trusted Network and Service Infrastructures* are discussed and compared in Section 5.4.4. Finally, a discussion about other related research is given in Section 5.4.5.

Based on the comparison of the related work, it becomes evident that this thesis represents a novel approach for service ecosystem engineering. While there are a lot of seminal work done in the academia, standardization community and industry, none of them provides means for rigorous service ecosystem engineering to the extent of this work. Instead, the related work address individual phases of the service ecosystem engineering life cycles, e.g. analysis and design, instrumentation, or operation. The holistic approach of this work with respect to service ecosystems and their engineering is the most distinguishing feature when compared to other state-of-the-art research.

5.4.1 OASIS SOA Reference Model

The OASIS Reference Model for Service Oriented Architecture (SOA-RM) is a conceptual model for service-oriented architectures produced by the OASIS standardization organization [154]. OASIS SOA-RM defines concepts, their interrelationships, and conformance guidelines which can be utilized for assessing conformance of systems and standards with respect to the reference model. The OASIS SOA-RM essentially defines an ontology for representation of service-oriented architectures. In the following, a comparison between the conceptual models of OASIS SOA-RM and SEAF is given. After that, the properties of SEAF are assessed with respect to the OASIS SOA-RM conformance guidelines.

The OASIS SOA-RM includes seven primary concepts related to services, their visibility and their application environment. A service is defined by the OASIS SOA-RM as a mechanism to access one or more capabilities [154]. A service is provided by an entity, a service provider, and accessed by service consumers through service interfaces. The service concepts of SEAF clearly align with the OASIS-SOA-RM definition of services. However, the definition of services in the SEAF conceptual model is more specific, and involves a categorization of services into infrastructure, business and component services.

Invocation of a service realizes one or more real-world effects which are actualized as returned information or changes in the shared state of defined entities [154]. In SEAF real-world effects of service invocations are specified explicitly by the capabilities associated with a corresponding kind of service. That is, the real-world effects of infrastructure services, business services and component services are denoted by ecosystem capabilities, business transactions and service protocols.

In addition to real-world effects, the OASIS SOA-RM describes dynamics of services with the notions of visibility and interaction [154]. Service visibility in OASIS SOA-RM is a relationship between service providers and consumers that is a composition of willingness, awareness and reachability [154]. Willingness for interaction is addressed in SEAF with the different rules associatable with services and legal entities. Awareness refers to a state where one party has knowledge of the existence of another party [154]. Especially, service awareness requires that service descriptions and policies constraining their usage are available [154]. One of the primary purposes of SEAF is to facilitate such awareness in service ecosystems. Service awareness is addressed explicitly by the concepts of service offers, for example. Reachability is defined in OASIS SOA-RM as the relationship between service participants where they are able to interact. Reachability in this sense is addressed in SEAF by the notions of cooperation facilities, service contracts and features associated with the preceding concepts.

Service interactions in the OASIS SOA-RM are defined as exchange of infor-

mation according to pre-determined behaviour [154]. The form of service interaction is prescribed in the service description, which comprises an information model and a behaviour model. The information model of a service characterizes the information that may be exchanged with the service [154]. The information model describes information structure, structural relationships, and definitions and semantics of terms used [154]. The domain ontologies and concept intensions in SEAF serve a similar purpose of describing exchangeable information.

In OASIS SOA-RM the notion of service behaviour model denotes the selection of actions usable and their temporal dependencies in service interactions [154]. The behaviour model includes an action model and a process model. The action model characterizes the actions that can be invoked against the service [154]. In SEAF such an action model is subsumed by the concepts of interaction schemes and service protocols associated with service endpoints and component services, correspondingly. In OASIS SOA-RM the notion of process model refers to prescriptions of temporal relationships and temporal properties, and events associated with interacting with services [154]. Similar notions are addressed in SEAF by concepts of business transactions and service choreographies.

The OASIS SOA-RM proposes concepts that should be included in service descriptions. While the actual contents and structure of service descriptions depend on the domain of operation and technology used, for example, generic guidelines about requirements for service descriptions can be identified. The OASIS SOA-RM identifies that service descriptions should include information about service existence and reachability, service functionality, constraints and policies affecting service usage, and how to interact with the service [154]. All these aspects required for service descriptions are covered in SEAF. Service existence and reachability is addressed by the concepts of service offers and service channels. Service functionality is prescribed in definitions of different service entities. Constraints and policies affecting service usage can be associated with services, cooperation facilities, and contractual elements in SEAF. Finally, interaction and communication is addressed especially by cooperation facility concepts.

The OASIS SOA-RM considers contracts and policies as foundational notions of service-oriented architectures. A policy represents a *"constraint or condition on the use, deployment or description of an owned entity as defined by any participant"* [154]. The conceptual model of OASIS SOA-RM identifies three aspects of policies: *1)* policy assertions, *2)* policy owners, and *3)* policy enforcement [154]. Policy assertions are measurable norms regarding service interaction and service usage context. In SEAF such policy assertions are represented by concepts of rules and rule bases associatable with different conceptual elements, such as legal entities, business services or service liaisons. Policy owners, or policy subjects, can be traced in SEAF through ownership of business services. Policy enforce-

ment is not explicitly addressed by SEAF, since policy enforcement mechanisms are domain-specific design choices.

A contract in the OASIS SOA-RM prescribes an agreement about service usage between two or more participants. Contracts can cover aspects such as quality of service, interface and choreography properties, or commercial agreements [154]. In SEAF service contracting is addressed by the concept of service liaison which includes elements for describing service quality and service behaviour aspects. While commercial agreements are not addressed in the conceptual model of SEAF, concepts for expressing such additional aspects of service contracts can be included in service architecture descriptions with domain-specific extensions to the corresponding domain ontology.

The conceptual model of the OASIS SOA-RM includes a notion of execution context for service interactions. An execution context of a service interaction is described as *"the set of infrastructure elements, process entities, policy assertions and agreements that are identified as part of an instantiated service interaction, and thus forms a path between those with needs and those with capabilities"* [154]. The execution context of service interactions is addressed in SEAF by the concepts of service contracts and ecosystem domains. Service contracts define the agreements between interaction participants, while ecosystem domain descriptions characterize the operational environment of interactions.

Finally, the OASIS SOA-RM standard prescribes conformance guidelines which state principles for service-oriented architectures aligning with the OASIS definitions. A service-oriented architecture conforming with the OASIS SOA Reference Model should [154]:

- Have entities that can be identified as services as defined by this Reference Model;
- Be able to identify how visibility is established between service providers and consumers;
- Be able to identify how interaction is mediated;
- Be able to identify how the effect of using services is understood;
- Have descriptions associated with services;
- Be able to identify the execution context required to support interaction; and
- It will be possible to identify how policies are handled and how contracts may be modeled and enforced.

It can be seen easily that SEAF is applicable for designing and instrumenting service ecosystems which align with the OASIS SOA-RM conformance guidelines. The conceptual model of SEAF includes concepts of infrastructure, business and component services which are associated with corresponding intensional definitions. Visibility of services between service providers and consumers is represented by service offers. Mediation of interaction is addressed in SEAF by concepts of service channel types, binding type, service channels and their corresponding features.

Real-world effects of services are described in SEAF with various kinds of behavioural descriptions, i.e. business protocols and service operations, and business transactions and commitment operations. The service execution context in the sense of SOA-RM [154] is addressed in the SEAF by the concepts of ecosystem life-cycle definitions, service choreographies, binding models, and definition of ecosystem artefacts and domain ontologies. Finally, SEAF enables rich expression of service contracts, e.g. symmetric and asymmetric service contracts, or in- and out-sourcing contracts, with concepts provided for defining service liaisons, service contracts and their relationships with ecosystem-specific binding models and features. Contract enforcement is not addressed at the ecosystem architecture level explicitly; however, contract enforcement facilities can be modelled in service ecosystem architectures as infrastructure services, for example.

As a summary, SEAF subsumes concepts that comply with the conceptual model of OASIS SOA-RM. Moreover, the conceptual model underlying SEAF aligns with the conformance guidelines of OASIS SOA-RM. However, the conceptual model of SEAF addresses additional concerns not included in OASIS SOA-RM, such as evolution and dynamism in service ecosystems. Moreover, the concepts of SEAF are more specific and formalized in the service ecosystem meta-model for enabling instrumentation and governance of service ecosystems.

5.4.2 Reference Model for Open Distributed Processing

The Reference Model for Open Distributed Processing (RM-ODP) is an international standard jointly created by standardization bodies ISO and ITU-T [90]. The standard provides means for describing and building widely distributed systems and applications in a systematic way [90]. The systems of interest that the standardization addresses are called ODP systems in the RM-ODP. This class of systems includes IT systems, information systems, embedded systems, business systems, or any other large or complex systems.

The RM-ODP standardization comprises four parts under the general title *Information technology – Open Distributed Processing – Reference Model*: *1) Overview* [69] providing overview of ODP, including explanation of key concepts, and application scenarios and examples, *2) Foundations* [67] providing definitions of

ODP concepts, *3) Architecture* [68] containing the specification of required characteristics for ODP system architectures, and *4) Architectural semantics* [70] containing a formalization of the ODP modelling concepts in different standardized formal description techniques.

The RM-ODP standardization defines an architectural framework for design and analysis of distributed processing systems with large or complex structure. As an architectural framework, RM-ODP identifies several stakeholders associated with ODP system life cycles, and provides definitions for architectural viewpoints. The viewpoints are intended to address concerns of five clear groups of users related to ODP systems. The viewpoints are: *1)* enterprise viewpoint, *2)* information viewpoint, *3)* computational viewpoint, *4)* engineering viewpoint, and *5)* technology viewpoint [69, 90]. Each of the viewpoints are associated with corresponding viewpoint languages [69]. An ODP viewpoint language is defined as a set of concepts enabling modelling of systems from that viewpoint.

The enterprise viewpoint addresses the organizational situation in which the design activity of an ODP system is taking place [90]. The enterprise viewpoint focuses on the objectives, business rules and policies that need to be supported by the system [90]. The stakeholders associated with the enterprise viewpoint include owners of the business processes and the managers responsible for the setting of operational policies [90].

An enterprise specification of an ODP system is a description of the system and the relevant parts of its environment [66]. A notion of communities is used in the enterprise viewpoint for structuring and specifying collections of enterprise objects. An enterprise specification can include one or more community definitions. Enterprise specification provides descriptions for the roles and processes enterprise objects take part in; actions are assigned to enterprise objects with roles.

The RM-ODP enterprise viewpoint is used for specifying the purpose, scope and policies for the ODP system [67]. The purpose of an ODP system is declared in the RM-ODP enterprise viewpoint primarily with two constructs. Firstly, ODP communities can be associated with objectives that declare practical advantages or intended effects, expressed as preferences about future states [66]. Secondly, the purpose of an ODP system is defined by the behaviour of the system. In SEAF the purpose of a service ecosystem is defined by ecosystem capabilities and life cycles: capabilities represent objectives of the ecosystem while life cycles define the behaviour expected from ecosystem members.

In RM-ODP the scope of a system is defined in terms of intended behaviour expressed by roles and processes in the enterprise viewpoint [69, 66]. In SEAF the scope of a service ecosystem is defined by ecosystem domains and service choreographies. Ecosystem domains enable division of service ecosystems to sub-systems (corresponding somewhat to the federation community type in RM-

ODP). Service choreographies associated with life-cycle phases (via service liaison specifications) define the kind of behaviour expected from ecosystem participants.

A policy in the RM-ODP standardization is considered as a set of rules related to a particular purpose [66]. A single rule can be expressed in RM-ODP with deontic modalities of obligation, an authorization, a permission or a prohibition [66, Section 6.4.1]. The modality of rules in SEAF is not prescribed. Instead, domain-specific semantics for rules and rule sets expressing policies are expected. However, the concept of cooperative features (see Table 3.1) in SEAF specifies different categories of rules that can be associated with business roles and monitored through conformance points.

The information viewpoint of RM-ODP addresses modelling of the shared information manipulated within the system of interest [90]. Information is represented in this viewpoint by information objects and their inter-relationships specified by a set of invariant, static and dynamic schemata [69]. Invariant schemata are used for specification of relationships between information objects that must hold always in the corresponding system of interest [69]. Static schemata express assertions that must hold in a single point of time [69]; they are utilizable for expressing initial values for information objects, for example. Dynamic schemata define how information may evolve as the system operates [69].

SEAF involves concepts and mechanisms that correspond to the RM-ODP schemata. Invariant schemata are represented in SEAF by domain ontologies and concept intensions. One-to-many relationships between concepts and artefacts in SEAF correspond to static and dynamic schemata of RM-ODP: each artefact specification defines an assertion that must hold in the corresponding life-cycle phase. Evolution of the information is described by usage of distinct artefacts in different life-cycle phases and behavioural contexts.

The computational viewpoint addresses high-level design of the processes and applications in the system of interest [90]. Object-oriented concepts are utilized for expressing different computational elements of the system with strong encapsulation boundaries and well-defined interfaces. The computational viewpoint does not prescribe distribution of the processes and applications; the same computational specification can be implemented differently on different platforms [90]. The computational viewpoint is used for specification of computational interfaces characterized by a signature, behaviour and an environmental contract [69]. Binding objects are specified in the computational viewpoint for enabling interaction between computational objects [69].

Elements of the RM-ODP computational viewpoint correspond to concepts of service endpoints, component services and service contracts in SEAF. Service endpoints in SEAF define the kinds of activities that comprise interactions. Com-

ponent service specifications declare service protocols and interaction schemes that describe expected behaviour. Service contracts define expectations about the environment of service interactions in SEAF.

The engineering viewpoint focuses on how interaction is achieved between computational objects and what kinds of resources are required to support the interactions [69]. The engineering viewpoint provides concepts for specification and analysis of platforms for distributed processing. The major objective of the engineering viewpoint is to support the distribution transparency requirements of the computational objects [90]. Distribution transparencies can be regarded as guarantees provided for computational designers for solving a distribution problem [90], such as access transparency enabling inter-working across heterogeneous computer architectures and programming languages [69].

Concerns aligned with the RM-ODP engineering viewpoint are represented in SEAF with concepts of cooperation facilities and features associated with them. With these concepts, a service ecosystem architecture prescribes in a technology-independent manner how interaction and communication is established between services, and what kinds of properties affect the communication. SEAF does not explicitly address distribution transparencies; such capabilities can be expressed implicitly by specifications of appropriate cooperation facilities, for example.

The technology viewpoint of RM-ODP addresses the choice of technology to support system distribution [69]. The technology viewpoint is utilized for specification of information about the existing environment, current procurement policies and configuration issues [90]. The technology viewpoint specifies how an ODP system is to be implemented. The implementation of an ODP system is specified with a configuration of technology objects that represent the hardware and software components of the implementation [90].

While specifications of endpoint kinds can be considered as technological artefacts, SEAF is primarily a technology agnostic architecture framework. However, SEAF does not forbid representation of technology-specific constructs; appropriate interpretation of the abstraction level used in service ecosystem architecture descriptions is left for the stakeholders to decide.

The fundamental concepts of RM-ODP include object models, instantiation and refinement of system elements and their specifications, composition, grouping, contracts, and policies. ODP system specifications are expressed in terms of objects which can represent physical elements, human beings, or more abstract entities, for example [90]. An object is defined as a model of an entity characterized by its behaviour, and dually, by its state [67]. Object behaviour is modelled as a collection of actions and constraints on when they may occur; the object state prescribes the set of sequences of actions in which the object may take part in a given instant in time [67]. Interfaces are used for prescribing abstractions of ob-

ject behaviour; they comprise a subset of the interactions of the object together with a set of constraints when they may occur.

The domain ontology model underlying SEAF serves a similar purpose to the object model of RM-ODP. While the object model of RM-ODP is very generic, the domain ontology model underlying SEAF is more specific and fine-tuned for the domain of service ecosystems. Entities in SEAF are classified into different sub-categories, such as legal entities or service entities; the RM-ODP object model does not have such conceptual sub-categories due to the more generic focus of the standardization efforts. The specialized and more structured domain ontology model enables more efficient design and analysis of service ecosystems, as more specific concepts offer more expressiveness and stricter semantics when compared to generic modelling concepts of RM-ODP. Domain-specific concepts (e.g. "mobile phone", "resource") not included in the domain ontology model explicitly can be specified in SEAF as *DomainConcept* elements.

Instantiation and refinement of ODP system elements and their specifications are enabled in the RM-ODP standardization by types, classes and templates. ODP objects, actions and interfaces are generally specified in terms of their types [90]. A type in RM-ODP is a predicate that characterizes a collection of elements [67]. Specifications of type predicates are given in SEAF by providing behavioural and feature definitions for concepts. Linguistic instantiation is utilized in SEAF for introduction of concept instances at the same level of abstraction. In addition to linguistic instantiation, ontological typing hierarchies are explicitly defined in SEAF with conformance relationships between concepts. The domain ontology model for service ecosystems specifies constraints over the ontological typing relationships: entities are typed by entity kinds and properties are typed by features, correspondingly. The RM-ODP standardization does not specify such ontological typing relationships or constraints due to the generic nature of the object model.

A class in ODP is the set of all elements satisfying a specific type [67]. In SEAF, classes in the RM-ODP sense are represented in the conceptual model and the service ecosystem meta-model by the notion of concept extensions. Extension of a SEAF concept denotes the set of instances conforming with the semantic and intensional definition of the concept.

Finally, the notion of template in RM-ODP refers to the specification of an element, which includes sufficient detail to allow instantiation of the corresponding kind of elements [90]. For example, in object-oriented programming languages concrete classes are templates [90]. In SEAF each intensional definition of concept comprises a template in the RM-ODP sense: the intensional definition can be used for instantiating a corresponding kind of element.

As a conclusion, the RM-ODP standardization and SEAF have many common characteristics. Many elements defined in the RM-ODP viewpoints have their corresponding elements in SEAF. However, there are some fundamental dif-

ferences between SEAF and RM-ODP. First of all, RM-ODP is constructed for allowing design and analysis of various kinds of complex or large distributed systems. SEAF, on the contrary, concentrates on service ecosystems, which is one kind of a complex and large distributed, service-based system.

Secondly, the conceptual model underlying SEAF is more specific and targeted for addressing service-oriented computing, service delivery and service engineering concerns; RM-ODP provides more generic concepts based on the object-oriented paradigm for representing ODP systems.

Thirdly, SEAF is founded on a formal service ecosystem meta-model which provides means for unified, holistic architecture descriptions that address requirements for service ecosystem sustainability. RM-ODP standardization does not define such a meta-model, but instead provides some guidelines for defining correspondences between different ODP viewpoint models. The RM-ODP standardization is supplemented by the UML4ODP [71] standard, which provides a modelling notation for representation of open distributed systems according to RM-ODP principles. The UML4ODP defines an UML profile [111] that provides each RM-ODP viewpoint with a modelling language. In addition, the UML4ODP defines a notation for description of dependencies between the distinct viewpoint models and their elements [71].

In practise, the UML4ODP defines a MOF-compliant meta-model for each RM-ODP viewpoint language [71]. The meta-models define subsets of the corresponding viewpoint concepts defined in the RM-ODP standardization [67, 68]. Using the corresponding viewpoint models and a model defining their correspondences, a complete representation of an ODP system can be expressed. However, the RM-ODP or UML4ODP standardization does not provide practical means for creation of such a unified and holistic architecture description model from the individual viewpoint models. As a consequence, utilization of RM-ODP architecture models for instrumentation of ODP systems can be more difficult than that of SEAF. The design of SEAF is actually driven by utilization of the resulting holistic architecture models for service ecosystem instrumentation.

Lastly, the SEAF definition includes more viewpoints than the RM-ODP framework: there are seven viewpoints in SEAF while RM-ODP defines five viewpoints. For example, in RM-ODP the purpose, scope and policies of an ODP system are defined using a single viewpoint, namely the enterprise viewpoint. In SEAF, the purpose, scope and policies associated with a service ecosystem are addressed by the Ecosystem capability, Service choreography, Life-Cycle, and Concept intension viewpoints. The more refined division of viewpoints in SEAF is due to a different and richer conceptual model with respect to the RM-ODP. More viewpoints are required in SEAF for handling the structural complexity of the conceptual model, and especially for addressing the aspects of ecosystem dynamism and evolution, which are not in the focus of RM-ODP.

5.4.3 Enterprise Architectures

The Service Ecosystem Architecture Framework shares similarities with so-called enterprise architectures. In the following the definitions and objectives, applicability, and concepts of enterprise architectures are briefly described and paralleled with SEAF.

An enterprise architecture can be defined as *"a coherent whole of principles, methods, and models that are used in the design and realization of an enterprise's organizational structure, business processes, information systems, and infrastructure"* [88]. There are several enterprise architecture frameworks developed in the industry and academia, such as The Open Group Architecture Framework (TOGAF) [118], the US Department of Defense Architecture Framework (DoDAF) [161], or ArchiMate [119]. TOGAF concentrates on enterprise architecture development processes by definition of the Architecture Description Method (ADM), but does not provide a complete language for description of enterprise architecture models. DoDAF is specialized for description of military organizations which follow a certain taxonomy of operations and activities; this is reflected in the DoDAF concepts and selection of viewpoints. ArchiMate provides a methodology agnostic and domain-independent enterprise architecture description language.

SEAF involves the notion of cooperative features which corresponds to principles regulating the operation of service ecosystems and realization of service-based collaborations in them. Methods for service engineering are specified using the SEAF Engineering viewpoint. Moreover, SEAF enables representation of service ecosystem processes and infrastructure with ecosystem life cycles, service choreographies, and infrastructure services and capabilities. In this light, the architecture descriptions produced with SEAF could be considered as a kind of enterprise architecture descriptions for service ecosystems. SEAF is a technology and methodology agnostic architecture framework, and can be applied to different service ecosystem domains, such as product-driven ecosystems, software application ecosystems, or social networking ecosystems.

Enterprise architectures are applied for managing the inherent structural complexity of enterprises, enabling enterprise engineering [65] and directing enterprise transformations [58]. In these application scenarios, enterprise architectures have three different roles: *1)* regulative, *2)* instructive, and *3)* informative [58]. Enterprise architectures are applied as regulative prescriptions which state how enterprises must become. Instructive application of enterprise architectures emphasizes their role as high-level designs providing instructions for directing the activities taking place in development programs and projects. Enterprise architectures are utilized in an informative role for enabling decision making by sharing knowledge on architecture decisions [58].

Similarly to enterprise architectures, SEAF is applicable for managing the complexity of service ecosystems which are complex, socio-technical systems. First of all, the conceptual model of SEAF is utilizable for conceptual decomposition of service ecosystems and for identifying entities and their inter-relationships. Secondly, service ecosystems can be formally modelled with SEAF for enabling further refinement and re-factoring of ecosystem concepts, and design and analysis of ecosystem architectures. While enterprise architectures can be utilized for enabling enterprise engineering, the ecosystem architectures constructed with SEAF are applicable for instrumentation of service ecosystems, as discussed in previous sections.

With respect to enterprise transformations, the ecosystem architectures can be used in a similar manner to guide development of a service ecosystem to a more mature system. For example, different versions of service ecosystem architecture descriptions could be used for representation of the current and target states of service ecosystems. Moreover, the capabilities prescribed in a service ecosystem architecture are not necessarily implemented at the very beginning of service ecosystem existence. In this setting, such "abstract capabilities" can be considered as prescriptions for the target state of the service ecosystem; infrastructure services, service life cycles, and business services delivering the required infrastructure service functionality are then developed in a progressive manner to fulfill the capability requirements.

Service ecosystem architectures also possess regulative, instructive and informative roles, similarly to enterprise architectures. The regulative role is emphasized especially during ecosystem operation: the ecosystem architecture models are applied for delivering model-based interoperability in the ecosystem, and for enabling service ecosystem governance [143]. The instructive role of service ecosystem architectures is emphasized when they are applied for directing engineering of component and business services. Finally, the informative role of service ecosystem architectures emerges in different phases involving business decisions addressing participation in service ecosystems. Service ecosystem architectures can be analysed for example to identify possible conflicts between policies of a service ecosystem and an enterprise considering joining it.

Typical conceptual models of enterprise architecture frameworks enable representation of entities, behaviour and artefacts in business, application and technology domains. Since the primary purpose of enterprise architectures is direction and regulation of design activities in development programs and transformation projects, enterprise architecture frameworks include concepts associated with enterprise strategies such as vision, principle or strategy (see e.g. [26, 57]). Such strategic concepts are not explicitly part of the current version of SEAF; however, visions, principles and strategies can be at least partly represented with differ-

ent kinds of rules and policies associatable to the elements of service ecosystem architectures.

In enterprise architecture frameworks development programs and projects are utilized for implementing enterprise transformations. Programs are collections of transformation projects which fulfill intermediate goals described in the enterprise architecture. The intermediate goals can be for example capabilities or services which progress the transformation of the enterprise from its current state towards the target state. Such management of iterative enterprise progression, and identification of means to enable the progression are part of the foundational motivations for use of enterprise architectures. SEAF does not include concepts for explicitly addressing or managing such iterative progression of service ecosystems. However, SEAF provides means for identifying the fundamental capabilities required from the service ecosystem. Infrastructure services and ecosystem life cycles delivering the capabilities could be considered as concepts parallel to transformation projects and programs, correspondingly.

In summary, enterprise architectures and SEAF share similar objectives and application scenarios in their respective domains. Where enterprise architectures' motivation stem from enterprise engineering and transformations, the primary motivations for SEAF are instrumentation and governance of service ecosystems. Both enterprise architectures and service ecosystem architectures can be applied for enabling business decision making in organizations. Enterprise architectures do not typically address collaboration between autonomous entities, but concentrate on operations within a single enterprise; the design of SEAF is founded on the idea of enabling such collaborations.

5.4.4 European FP7 research

There are several European research initiatives and projects that have objectives and approaches aligning with this academic dissertation. In the following, a selection of initiatives in the ICT (Information and communication technologies) research theme of the Seventh Framework Programme (FP7) of European Commission [41] are introduced. The objectives, approach, results and applicability of the projects are compared with the results of this academic dissertation.

The FP7 ICT programme is divided into eight work programmes on the basis of strategic interest to European society, with an additional research area targeted for future and emerging technologies [41]. The work programmes are framed in the form of research challenges. The work programme which subsumes themes related to this academic dissertation is titled *"ICT Challenge 1: Pervasive and Trusted Network and Service Infrastructures"* within the FP7 ICT research programme. The "ICT Challenge 1" work programme is further structured into seven objectives declaring more specific research areas within the work programme.

Objective 1.2 of the "ICT Challenge 1" work programme is closely related to this academic dissertation with research conducted in areas such as service-oriented architectures, service engineering, and service platforms. Objective 1.2 is currently entitled *"Cloud Computing, Internet of Services and Advanced Software Engineering"*[1] in the 2011-2012 version of the work programme.

INDENICA (Engineering Virtual Domain-Specific Service Platforms) [49] is a research project conducted under Objective 1.2 of "ICT Challenge 1" work programme. The INDENICA aims for creating methods and tools for enabling efficient design and development of service platforms. Efficiency is provided with reuse and specialization of engineering artefacts. More specific objectives of the INDENICA project include simplification of service platform development and service engineering, enabling support for platform convergence and interoperability, and establishing a reusable infrastructure for platform development. These objectives are addressed with approaches such as product-line engineering, variability management and self-adaptation techniques. The results of the project include a design for a virtual, self-adaptive service platform which can be configured using the product-line engineering techniques developed in the project.

The approach developed in this thesis and the contributions of the INDENICA project supplement each other. The approach developed in this thesis provides means for designing, analysing and formalizing requirements for service platforms. By design, the service ecosystem architecture models are situated higher in the abstraction hierarchy when compared to the conceptual framework of INDENICA. The feature-based variability management models of INDENICA could serve as platform-specific models for service ecosystem architectures. In this setting, the service ecosystem architecture models could be utilized used for eliciting requirements for INDENICA-based virtualized service platform. The conceptual model of the INDENICA-project is concentrated on services, their features and re-usability. The results presented in this thesis include a more abstract and holistic view on service ecosystems and especially, service ecosystem sustainability.

The Cloud4SOA initiative [48] focuses on resolution of semantic interoperability problems in current cloud service ecosystems. In addition, the project seeks to develop user-centric approaches to applications which are built and deployed in cloud service ecosystems. The approach taken in the Cloud4SOA project to meet these objectives is based on integration of service-oriented architectures, cloud computing, and lightweight semantics. The main deliverable of the project is the Reference Architecture for Semantically Interoperable Clouds (RASIC) [48]. RASIC comprises three horizontal layers (Service front-end, SOA, and Virtualization and Execution layer), and two vertical layers (Semantic layer and Governance layer) [95]. The service-frontend layer enables users to adapt existing cloud ser-

[1] http://cordis.europa.eu/fp7/ict/ssai/workprogramme-2011-12-obj1-2_en.html

vices and to implement new ones as compositions, or "mash-ups". The SOA-layer of RASIC provides service discovery, mashing and recommendation components that enable design, development and deployment of cloud-based services. The semantic layer enables resolution of semantic interoperability problems and development of intelligent service discovery, mashing and recommendation services. The semantic layer includes a lightweight service and computing resource model, service and resource annotation components, and a semantic interoperability run-time engine [95].

The reference architecture developed in the Cloud4SOA project could be utilized as a platform-specific model for service ecosystem architectures. In such a setting, service ecosystem architecture models are transformed with model transformations to models consumable by the RASIC framework, similarly to the approach taken in the model-driven methodology described in Section 4.2. A cloud infrastructure that conforms with the RASIC framework could then be used for rapid deployment of service ecosystems. The Cloud4SOA project is grounded on utilization of the cloud infrastructure, while the approach developed in this thesis is technology agnostic.

The CHOReOS (Large Scale Choreographies for the Future Internet) [47] research project aims at assisting engineering of service choreographies by delivering appropriate development processes, methods and tools [164]. The project instruments development of large-scale service choreographies with use of a middleware platform which enables definition, deployment and enactment of the choreographies [164]. The results of the project will include a middleware platform enabling deployment and enactment of large-scale choreographies, an integrated development and run-time environment (IDRE), and a dynamic development model utilizing the middleware and the IDRE [47]. The CHOReOS is applicable in the instrumentation and operation phases of service ecosystem engineering life cycles. The CHOReOS middleware provide infrastructure services for a CHOReOS-like service ecosystems. The CHOReOS development tools and methods are utilizable during service ecosystem operation for development of service choreographies, as well as services required for implementing the choreography functionality.

The CHOReOS project is more technology-oriented than the research conducted in this thesis which focuses on service ecosystem architecture descriptions. The objectives and approaches, as well as results of these two research activities are thus quite different. However, the CHOReOS project and the approach developed in this thesis are complementary. The CHOReoS project provides run-time and service engineering constructs that could be utilized for implementing service ecosystem architectures. The approach developed in this thesis could be utilized for design and delivery of specialized service ecosystems over the CHOReOS

middleware and the integrated development and run-time environment.

The SOA4All project aims at enabling operation of large-scale service ecosystems with a framework and infrastructure for service delivery platforms [51]. The SOA4All approach is founded on application of light-weight semantics in description of services and their features [81]. Semantic descriptions in SOA4All are based on Semantic Web-technologies, such as RDF [168] and Web Service Modelling Ontology (WSMO) [128, 178]. In addition to Semantic Web-technologies, the approach integrates service-oriented architectures, context-aware technologies, Web principles [50] and Web 2.0 idioms (see e.g. [105]) for delivering a distributed service bus, a service development environment and a set of infrastructure services [81]. The SOA4All project results are applicable for facilitating service ecosystem instrumentation and operation phases.

The results of the SOA4All project are complementary to the artefacts provided in this academic dissertation. While the SOA4All project addresses mainly the instrumentation and operation phases, the approach developed in this thesis concentrates especially on the analysis and design phases. However, the model-driven methodology for service ecosystem engineering could be implemented over any SOA-based platform, such as the SOA4All service bus. The development environment and set of infrastructure services of the SOA4All project could be included in service ecosystem architectures and their descriptions.

NESSI (*Networked European Software & Services Initiative*) [107, 92] is a European Technology Platform dedicated to software and services. As part of its research activities, the NESSI consortium is developing the NESSI Open Service Framework (NEXOF), which is described as *"a coherent and consistent open service framework leveraging research in the area of service-based systems"* [108]. The NEXOF research initiative develops a conceptual framework and a reference architecture (NEXOF-RA) for service-based systems to consolidate and trigger innovation in service-oriented economies.

While research on the NEXOF, its reference architecture and conceptual model is still going on, some similarities and differences can be identified with respect to this academic work. Both the conceptual model of this academic work and the one in NEXOF-RA facilitate construction of distributed service-based infrastructures. The NEXOF-RA will include a set of different model parts representing different viewpoints to service-oriented systems. However, the concepts of the NEXOF-RA, in the current state of research [108], are quite generic, since the corresponding conceptual model provides a unified terminology that can be used for defining various kinds of NEXOF-compliant infrastructures. In this academic work the conceptual model is more specific and is especially targeted for facilitating sustainable service ecosystems. However, service ecosystem architectures designed with the approach developed in this thesis could be deployable over NEXOF-

compliant infrastructure. Dually, the NEXOF-RA could be modelled with the service ecosystem architecture framework. This could be beneficial for establishing interoperability between NEXOF-compliant, but separately developed service ecosystems.

The objectives, approach, results and applicability of the preceding EU research initiatives are summarized in Table 5.4. The objective of the approach developed in this thesis is to enable transformation from ad-hoc to sustainable service ecosystems. This approach in this research is founded on explicit service ecosystem architectures and their utilization for establishing continued viability of service ecosystems. Service ecosystem viability is established by delivery of facilities that support stakeholder concerns in service ecosystem engineering life cycles. The continuation of viability is then supported by application of the explicit service ecosystem architecture models during the service ecosystem operation phase. The results of this research comprise a conceptual model for service ecosystems, a service ecosystem meta-model, SEAF, and a model-driven methodology for service ecosystem engineering. The approach developed in this thesis covers all phases in the systems engineering life cycles of service ecosystems, that is, analysis, design, instrumentation and operation.

5.4.5 Other related work

The SeCSE (*Service Centric System Engineering*) was an Integrated Project [147] of the 6th Framework Programme that aimed for developing processes, methods and tools to develop service-oriented systems [32]. As part of its research work, the SeCSE project provides a conceptual model for service-oriented systems. The model describes actors, entities and activities relevant to the service domain and relationships between them [32].

While the conceptual model of the SeCSE project addresses the various steps (e.g. publication, discovery, composition, and monitoring) of the service-centric system creation process, the primary purpose of the model is to provide a common understanding for human readers about the main concepts involved [32]. The primary purpose of the conceptual model provided by this academic work is to facilitate the infrastructure services and tools needed for instrumenting service ecosystems. That is, the conceptual model of this academic work is used for formally defining all the required elements for instantiating service ecosystems, while the SeCSE conceptual model is used for describing service-centric system development scenarios.

This academic work has a strong emphasis on service and service-based systems modelling, an area of active research which has been addressed both in academia and industry. The COSMO (*COnceptual Service MOdelling*) [126] project develops a framework for service as a common semantic model to en-

Acronym	Objectives	Approach	Results	Applicability
INDENICA	Simplification of service platform development; Support for platform convergence and interoperability; Reusable infrastructure for platform development.	Product Line Engineering (PLE), variability management and self-adaptation techniques.	Variability design and implementation for service platforms; Virtual Service Platforms	Service ecosystem design and instrumentation; Service engineering.
CLOUD4SOA	Resolution of semantic interoperability problems in current cloud service ecosystems; User-centric approach to applications.	Integration of cloud computing, SOA and lightweight semantics.	Reference Architecture for Semantically Interoperable Clouds (RASIC)	Service ecosystem design and instrumentation; Service ecosystem operation.
CHOReOS	Enabling design and development of ultra-large scale systems as choreographies composed of heterogeneous services.	A framework for scalable choreography development; Model-driven development process and tools.	Abstractions and models for services, interaction protocols, and coordination; Integrated Development Runtime Environment (IDRE); Service-oriented middleware; Governance mechanisms.	Service ecosystem operation (design and development of service choreographies, and business and component services)
SOA4ALL	Enabling operation of large scale service ecosystems with a framework and infrastructure for service delivery platforms.	Integration of SOA, the Web, context-aware technologies, Web 2.0 and Semantic Web.	SOA4All framework.	Service ecosystem operation (design, development and deployment of business and component services).
NEXOF-RA	An open platform for creating and delivering applications enabling establishment of service-based ecosystems.	Architectural patterns and principles; Design patterns; Idioms (implementation level patterns).	Specifications for a reference architecture as a coherent set of patterns and principles.	Service ecosystem analysis, design; Infrastructure service development.

Table 5.4: Characterization of related European FP7 research activities.

able use of different service modelling languages within a service ecosystem. The PIM4SOA (*Platform Independent Model for SOA*) project [12] provides a model for defining services in a technology independent manner. On the industrial side, the UPMS (*UML Profile and Metamodel for Services*) [14] is an on-going standardization work by the Object Management Group (OMG) for delivering a formal and unified service definition language. The OASIS Reference Model for Service-Oriented Architecture [154] defines guidelines for creation of service-based systems.

These approaches concentrate on providing a unifying meta-model for service modelling and for validating the consistency and conformance of service models during design. In comparison, the conceptual model developed in this academic work is designed for formalizing open service ecosystems. In addition to defining concepts related to services it also makes explicit the relationships between the different elements of service ecosystems, and puts emphasis also on the dynamic nature of service ecosystems. It also makes explicit the relationships between the ontological, infrastructure, knowledge management and methodological aspects of service ecosystems.

This dissertation provides a top-level ontology for service ecosystems. From this perspective, this work is related to so-called semantic web services (SWS) initiatives. Semantic web services refers to approaches that combine the use of semantic web technologies and approaches (see e.g. [148]), such as RDF [168] or OWL [167], and Web Service [169] technologies [99]. Semantic web services are used for enabling service discovery, composition and other functions needed for supporting service and service collaboration life cycles in corresponding service-oriented architectures. The SWS solutions utilize generic ontology description languages and associated logics for enabling inference over the concepts defined in domain ontologies. Typically a top-level ontology, for example OWL-S [121] or WSMO [178], is exploited for describing services and attaching concepts of domain ontologies to them.

The World Wide Web Consortium (W3C) has defined a standard called Semantic Annotations for WSDL and XML Schema, SAWSDL [79, 171]. The SAWSDL standard defines how semantic annotation of Web Service Description Language (WSDL) [31] can be accomplished using references to semantic models such as ontologies. A more comprehensive framework for semantic web services has been addressed by the Semantically enabled service-oriented architecture (SESA) [165] initiative. SESA utilizes the Web Service Modelling Ontology (WSMO) and the related WSML and WSMX facilities [128, 178] as an underlying technology for creating a middleware platform for semantically enabled SOAs. The SESA platform provides infrastructure services for service discovery, integration and mediation, for example [165].

Despite the use of service ecosystem-specific top-level ontology, the foundational goal of this academic work is, however, quite different from the SWS approach. This dissertation provides foundations for defining the life cycles, concepts and infrastructure required for realizing service ecosystems. Within such ecosystems, semantic web service approaches could be used for implementing the infrastructure services. Moreover, while semantic web services typically emphasize operation time life-cycle support, this work equally takes into consideration the engineering side of service-based systems. Especially, in the context of this dissertation, concepts are provided with intensions (definitions of their features) that are utilizable also as design and development models. That is, the ontological semantics of concepts and engineering artefacts representing the concepts are unified in the approach taken in this dissertation.

Chapter 6

Conclusions and consequences

This thesis proposed service ecosystem engineering as a novel systems engineering discipline for enabling establishment of sustainable service ecosystems. Moreover, a model-driven approach for service ecosystem engineering was developed and provided with appropriate tools. The discipline and the approach provides means for establishing sustainability in envisioned open service ecosystems.

In this chapter conclusions are drawn and the consequences of the research are discussed. First in Section 6.1 a summary of the results is given. In Section 6.2 the objectives and evaluation criteria for this work are recalled. An analysis is then given about how these objectives were met, how mature the provided solution is, and what the restrictions of the solution are. In Section 6.3 the impacts of this dissertation are discussed in a wider context. Finally, in Section 6.4 further prospects of this dissertation are discussed with topics ranging from semi-automatic generation of ecosystem-specific development tools to enabling open innovation in service ecosystems.

6.1 Summary of results

The motivation for this academic work is to enable a transition from closed, *ad hoc* service ecosystems to open, sustainable service ecosystems. Contemporary service ecosystems, such as based on cloud computing environments [3, 155, 10], are typically based on pre-negotiated agreements on the business relationships between partners. For enabling collaboration, the vocabulary, processes, and policies constraining and directing operations in these service ecosystems are predetermined and encoded in technical artefacts and implicit conventions to be followed. Correspondingly, the ecosystem capabilities are tightly coupled with technological infrastructure. Due to the lack of a rigorous service ecosystem engineer-

ing discipline, contemporary service ecosystems are based on architectures which are *a)* implicit, and *b)* concentrate on technological aspects of ecosystems.

For supporting ever growing and progressive service-based business and networked business models, as well as the "services everywhere" visions [42, 43], a transition from such closed and *ad hoc* solutions to open and sustainable service ecosystem is needed. Openness of a service ecosystem is characterized by enablement of member autonomy, and support for ecosystem evolution and dynamism. Sustainability means quality of a service ecosystem to support continued viability, where viability depends on the level of business-supporting capabilities it provides for its members. A viable service ecosystem must provide capabilities for *a)* efficient utilization of core competencies, *b)* opportunistic and flexible business networking, *c)* supporting progressive business environments, and *d)* efficient business decision-making.

For fulfilling the requirements set by the vision of open service ecosystems and their sustainability, this thesis presented a model-driven approach for engineering sustainable service ecosystems. From engineering perspective, the solution presented enables rigorous service ecosystem engineering disciplines. Rigorous service ecosystem engineering is enabled by explicit and formal service ecosystem architecture models. The architecture models enable formalization of architectural designs and knowledge, their assessment, as well as traceability over design decisions and their implications.

From business perspective, the solution supports establishment of sustainability in service ecosystems. The support is provided by addressing the stakeholder concerns appropriately in the artefacts developed in this dissertation and service ecosystem engineering life cycles. This dissertation concentrated especially on the analysis, design and instrumentation phases of service ecosystem engineering life cycles. For analysis and design, an architecture framework was defined and implemented. For service ecosystem instrumentation, methodological support was provided in form of engineering tools such as model transformations and an Eclipse [38] plugin for using the transformations in a user-friendly manner.

Finally, support for open service ecosystems was addressed especially in the formalization of the conceptual model. The design principles of the service ecosystem meta-model were driven by the vision of open service ecosystems. Consequently, the service ecosystem architecture models enable management of engineering and interoperability knowledge in open service ecosystems.

6.2 Analysis of the results

The results of this thesis are analysed below. First, the research objectives are recalled in Section 6.2.1 and an analysis is made how the objectives have been

met. Service ecosystem sustainability depends on how stakeholder concerns are addressed in service ecosystems. Consequently, Section 6.2.2 analyses how the stakeholder concerns identified in Chapter 2 are satisfied by the research described in this thesis. Finally, Section 6.2.3 discusses the demarcation made for the thesis research.

6.2.1 Meeting the research objectives

Three main objectives were set for this academic dissertation to enable the transition from *ad hoc* service ecosystems to sustainable ones. First, a rigorous service ecosystem engineering discipline is required. Secondly, support for service ecosystem sustainability needs to be addressed in the service ecosystem engineering life-cycle phases. Thirdly, means for enabling establishment of open service ecosystems must be delivered. To meet these objectives, this thesis presented model-driven facilities for design, analysis and instrumentation of service ecosystems.

Rigorous service ecosystem engineering discipline is supported by SEAF which enables analysis and design of service ecosystem architectures, and the model-driven methodology for service ecosystem engineering. These contributions are founded on the conceptual model and the meta-model for service ecosystems. Support for service ecosystem sustainability is addressed by the artefacts as discussed in Chapter 5.

Means for enabling establishment of open service ecosystems are provided by the service ecosystem meta-model. The design principles for service ecosystem meta-model, as discussed in Chapter 3, especially address service ecosystem evolution and dynamism. Autonomy of service ecosystem participants can not be addressed in the artefacts developed in this thesis. However, the explicit and formal service ecosystem architecture models, i.e. instantiations of the meta-model, can be utilized for enabling collaborative governance of open service ecosystems. Service ecosystem governance comprises collaborative activities between service ecosystem members for directing, managing and monitoring ecosystem operation [143]. Such distributed and collaborative governance practices can be only established, if the responsibilities and behaviour of participants are unambiguously declared; the explicit service ecosystem architecture models serve such a purpose. Moreover, service ecosystem architecture models enable so-called *model-driven interoperability* approaches where unifying, shared models are used for bridging heterogeneous systems, such as development tools and modelling languages [35, 25, 34], or organizations and enterprise systems [13, 89].

The main artefacts developed in this thesis are *1)* a conceptual model for service ecosystems, *2)* formalization of the conceptual framework as a meta-model, *3)* an architecture framework for analysis and design of ecosystem models, and *4)* a model-driven methodology for service ecosystem engineering. These arte-

facts and their completeness, feasibility and applicability for enabling sustainable service ecosystems are analyzed in the following.

This thesis has presented a sufficiently complete conceptual framework for service ecosystems. The completeness of the concepts was evaluated through a case study representing a state-of-the-art (i.e. Pilarcos service ecosystem) service ecosystem. As presented in Chapter 5, the conceptual framework includes sufficient vocabulary for expressing the characteristics of such diverse service ecosystems.

The feasibility of the conceptual framework was validated constructively by providing a meta-model, an architecture framework for creating service ecosystem models, and a methodology for service ecosystem engineering. The service ecosystem meta-model formalizes the conceptual framework and the semantics of the vocabulary used. The architecture framework includes a set of viewpoints and associated model-kinds. The required modelling tools were implemented by extending a standard compliant UML tool (i.e. MagicDraw 17[1]) with UML-profiling mechanism and tool-specific extensions for enhancing the user experience. The UML profile is based on the service ecosystem meta-model. The methodology for service ecosystem engineering is driven by model transformations that refine the user-provided models to more specific ones, and finally, to technology-specific service engineering artefacts (e.g. descriptions of domain-specific abstract syntaxes).

The applicability of the model-driven approach for service ecosystem engineering was demonstrated in Chapter 5. Model transformations were implemented that produced a selection of engineering artefacts from the service ecosystem models. The produced engineering artefacts represented different concerns of the service ecosystem stakeholders, especially infrastructure provider and engineering tool provider, identified in Section 2.2.

For addressing challenges laid by the vision of open service ecosystems, the artefacts developed in this thesis are designed especially to support different kinds of service ecosystem life cycles, service-oriented software engineering processes, and management and evolution of the service ecosystem knowledge. Without these properties of the concepts and the modelling framework, the viability and applicability of the resulting service ecosystems would be compromised.

The conceptual model and the meta-model for service ecosystem supports varying kinds of service ecosystem life cycles and life-cycle phases. The life cycles are defined as partially ordered collections of life-cycle phases. This gives ecosystem designers freedom to describe the ecosystem life cycles; domain-specific ecosystem models can constraint this freedom and introduce new kinds of relationships (e.g. recursive behaviour) using meta-model extensions. The life cy-

[1] http://magicdraw.com

cle phases can be classified into three categories (i.e. rigid, static and dynamic) depending on the level of dynamism supported in the life-cycle phase choreographies, as described in Chapter 3. In the case study the Pilarcos ecosystem utilizes both static (population phase) and dynamic (collaboration enactment phase) life-cycle phases.

Support for service-oriented software engineering processes is addressed in the service ecosystem meta-model. The meta-model is founded on the core principles of model-driven engineering, and especially, on the unification of ontological and linguistic modelling practices (see Chapter 3). As such, the meta-model enables efficient utilization of ecosystem knowledge in model-driven service engineering processes, and provides means for managing the engineering knowledge in service ecosystems. The feasibility and applicability of the approach is demonstrated in Chapter 5 by providing model transformations generating service engineering artefacts from ecosystem models.

Evolution of service ecosystem knowledge is necessary for maintaining the viability of the ecosystem. Support for knowledge evolution is needed to enable (gradual) service innovation and reactions to emerging business opportunities. Support for ecosystem knowledge evolution is addressed at the ecosystem meta-model. The meta-model defines constructs that can be used for defining static and dynamic knowledge, as well as for meta-model extension and variability. The mechanisms are implementable using contemporary modelling technology, such as Eclipse Modelling Framework (EMF) [39] and QVT [114] that were utilized for implementing the service ecosystem engineering methodology.

Without any means for managing the service ecosystem knowledge, the life cycles and service engineering processes become hard to manage due to interoperability problems. Management of service ecosystem knowledge is addressed in this thesis by providing a knowledge management meta-model utilizable for describing service ecosystem-specific knowledge repository infrastructures.

6.2.2 Satisfying the stakeholder concerns

For each identified service ecosystem stakeholder several concerns were recognized in Chapter 2. In the following, we go through these requirements and analyze how they were met by this work.

From the service consumer perspective, three fundamental concerns were recognized. The conceptual model for service ecosystems must enable *1)* business service identification, location and selection, *2)* business service dependability, and *3)* business service monitoring.

For enabling service identification and selection, the conceptual model and the modelling framework provides means for establishing ecosystem-specific service categorization, service ontologies and variable features in service offers. Service

categorization is enabled through use of a two-level ontological modelling approach, where services conform to service kinds. The extendability of the service ecosystem meta-model provides means for creating domain-specific relationships and concepts, and thus ontologies, over the core service concepts. The feature model presented in this thesis provides means for managing functional and non-functional features in service ecosystems [141] and facilitates feature-based selection of services with appropriate knowledge repository implementations. Means for efficient model-driven production of such knowledge repositories are also provided as part of this thesis.

Service provider concerns with respect to the conceptual model address especially support for service delivery and operation in service ecosystems. Two primary service provider concerns were identified in Chapter 2: *1)* service bundling, and *2)* service contracting.

The conceptual model and the modelling framework includes the concept of service offer for providing sufficient expressiveness for service bundling. The service offer concept enables exploitation of sub-contracting relationships and introduction of value-adding operations by service providers. This facilitates efficient service delivery by enabling utilization of provider-specific competencies and business networks (e.g. sub-contracting relationships), autonomy over service bundling, as well as service provider differentiation.

The concepts of service offers and service contracts utilize commitment operations to enable expression of complex service contracting relationships between service providers and consumers. This commitment model that is utilized in these concepts is based on an ontology of commitments in multi-agent systems [150]. Moreover, the service connections declared in service choreographies can be utilized for service-contracting purposes, especially regarding non-functional features of service interactions. Enactment of service contracts is facilitated by the behavioural descriptions associated with service kinds and service choreographies identified in the service contracts.

The conceptual model and the modelling framework enables business network management, including collaborative activities such as business network establishment, coordination, and monitoring. The concepts of service choreography, service liaison and service contract provide the primary means for business network establishment. New kinds of business networks can be declared with service choreographies. Service liaisons are used in business network establishment life cycles to represent initial, intermediate and finalized contractual (multi-lateral) agreements between partners. Service contracts represent mutually agreed, bilateral commitments and service agreements between partners. Finalized service liaisons and service contracts can be utilized for business network coordination and monitoring service interactions in the operational phase of the business net-

work. This is the approach used in the Pilarcos-ecosystem [87], for example.

Service engineers produce technological artefacts (e.g. web services) that are utilized by service providers to implement their business services. In Chapter 2 three concerns were identified with respect to service engineers: *1)* identification of engineering capabilities required for service production, *2)* well-advised service engineering, and *3)* efficient production of component services.

The engineering concepts, such as tool types and methods, included in the service ecosystem meta-model enable identification of required engineering capabilities. Well-advised service engineering refers to both correspondence of service engineering with the business models ("well advised motivations") and pragmatics ("well advised practices") of the ecosystem. The conceptual model developed in this thesis provides means for such well-advised service engineering.

First of all, the conceptual model provides a holistic framework which unifies business level service choreographies with elements contributing to service engineering, such as different kinds of service and interaction features. In service ecosystems the service choreographies are based on the business networks and business models relevant in the corresponding service ecosystem. The service engineers can utilize the published service choreography descriptions for developing services that correspond closely to the needs of the underlying business models. Service development based on published service choreography descriptions provides well advised motivations for service engineering in different domains. Moreover, utilization of domain-specific service description languages decreases the semantic gap between service engineering and business engineering domains.

Secondly, the conceptual model provide means for extracting, making explicit and sharing best practices in service ecosystems. Especially service kinds can be considered as representations of best practice for service development in service ecosystems. Service kinds can be derived from business models and networks (represented as service choreographies in the conceptual model) or based on technological premises. In mature service ecosystems the service kinds and their definitions can become valuable, reusable engineering assets for service engineers. Such decoupling between business engineering and service engineering domains would increase the sustainability of the service ecosystems. Decoupling the business and service engineering artefacts enables loosely coupled, iterative development of business models and enabling services while still preserving well advised service engineering.

For facilitating efficient service production in service ecosystems, this thesis provides means for creating coherent families of domain-specific (i.e. ecosystem-specific) modelling languages. The uniformity of the resulting language family is guaranteed by model transformations for creating ecosystem-specific knowledge management infrastructure. The model transformations provide a set of

intensional meta-models defining the abstract syntaxes of the DSLs, and more importantly, a knowledge management model which integrates the individual intensional meta-models to a coherent whole.

These model transformations, and the knowledge management meta-model providing means for implementing the knowledge management infrastructure, were described as part of this thesis. The utilization of domain-specific languages enables efficient, model-driven service engineering practices (e.g. generation of mock services for faster prototyping, code generation of service stubs and wrappers, model-based testing and deployment of services etc.).

Moreover, service engineers can utilize the service ecosystem knowledge base (including definitions for service choreographies, service kinds and different feature definitions) to direct their development efforts. Service choreography definitions available in the service ecosystem can be utilized in conjunction with re-usable model transformations to decrease the service development efforts; this becomes especially efficient in top-down approaches to service development (see Section 2.2).

Without appropriate service engineering tools, service production becomes hindered by technological complexities and longer development times. Complexity of service engineering may dispel potential ecosystem members while longer development times may result in missed business opportunities. Thus, the availability and quality of service engineering tools in a service ecosystem has a considerable effect on the sustainability of the ecosystem. Service engineering tool providers implement design and development tools which enable rapid and efficient service production methods. Three primary concerns were identified in Chapter 2 for service engineering tool providers: *1)* identification of engineering capabilities required for component service production (shared with service engineers and discussed above); *2)* efficient tool production and specialization, and *3)* language and tool interoperability.

This thesis provides means for efficient tool production. As demonstrated in Chapter 5 the ecosystem models can be utilized for generating the abstract syntax definitions (i.e. meta-models) for a modelling language family. The abstract syntax definitions already provided means for developing service ecosystem-specific modelling tools manually. For example, the Eclipse Modelling Framework [39] can be utilized for implementing model editors with graphical or textual concrete syntaxes for the generated meta-models. During service ecosystem operation, the architecture models and generated DSL families provide means for tool and language integration with model-driven techniques. Especially the technique known as *model weaving* can be used for such model integration [73].

Infrastructure providers deliver the infrastructure services required for enabling operation of service ecosystems. Delivering infrastructure services could

be business on its own, or it might be affordable due to supporting the other aspects of infrastructure provider business models (e.g. bootstrapping the use of business services in the service ecosystem). Three concerns were identified in Chapter 2 for infrastructure providers: *1)* identification and design of required infrastructure services, *2)* efficient production of infrastructure services, and *3)* infrastructure service interoperability.

Identification and design of required infrastructure services is enabled by SEAF and utilization of explicit service ecosystem architectures during analysis and design phases. These artefacts enable analysis and design of service ecosystems, including their infrastructure services, as well as facilitate communication between infrastructure providers and other stakeholders. Moreover, formal service ecosystem architecture models enable architecture analysis techniques.

Efficient production of infrastructure services based on service ecosystem models was demonstrated in Chapter 5. Software engineering artefacts were generated for implementing knowledge management infrastructures for different service ecosystems. Platform-specific models were automatically generated (SCA [109] and OSGi [120] models) from the service ecosystem models; these models enable efficient service production and deployment based on state-of-the-art SOA practices. Moreover, Java interfaces and Eclipse projects were generated for each knowledge repository required by a service ecosystem.

Infrastructure services interoperability was also demonstrated in Chapter 5. The service ecosystem models were used as means for model-driven integration of infrastructure services: the platform-specific models and other service engineering artefacts (e.g. Java interfaces) generated from ecosystem models reflect the inter-dependencies between knowledge artefacts and repositories declared in the knowledge management model.

Ecosystem providers represent entities such as organizations or consortia whose business models require utilization and instrumentation of a service ecosystem. Ecosystem providers are likely to also act as providers of (some of the) infrastructure services, but this is not necessary: the delivery of infrastructure services can be out-sourced, if applicable in the business model. Ecosystem providers are responsible, typically also trusted parties, which maintain the sustainability of service ecosystems by regulating their behaviour and providing means for other stakeholders to join and operate in the ecosystems. For ecosystem providers two main concerns were identified in Chapter 2: *1)* identification of required ecosystem capabilities and elements supporting their delivery, *2)* formalization of service ecosystem architecture descriptions, and *3)* supporting service ecosystem governance.

Ecosystem providers initiate establishment of service ecosystems by first defining their required capabilities and means for delivering those capabilities. For

this purpose the conceptual model developed in this thesis provides concepts of service ecosystem capabilities, life cycles and infrastructure services. The conceptual model already provides means for holistic analysis of service ecosystems. The conceptual model was formalized as a service ecosystem meta-model. Moreover, the conceptual model and SEAF provide means for defining the kinds of knowledge elements required for supporting the operation of the service ecosystem. The models resulting in the definition and formalization process enacted by ecosystem providers can then be applied for establishing business analysis and prototyping by different ecosystem stakeholders, for example.

Supporting sustainability in service ecosystems necessitates a framework for *service ecosystem governance*. Service ecosystem governance is here defined as a collaborative activity taking place between ecosystem members, which extends service governance (such as described in the Open Group SOA Governance Framework [117], for example) of the individual members in the ecosystem [143]. Service governance comprises activities and structures for directing, monitoring and managing the capabilities enacted in a service ecosystem. The conceptual model and the modelling framework for service ecosystems enables such collaborative service ecosystem governance by providing means for defining the ecosystem capabilities and their supporting life cycles and infrastructure services. The responsibilities of the ecosystem members with respect to service ecosystem governance activities become unambiguously defined in the ecosystem models.

As demonstrated above, the artefacts developed in this thesis enable facilitation of sustainable service ecosystems. The concerns identified for primary stakeholders are supported by this work. Most importantly, the requirements stemming from service ecosystem sustainability and the vision of open service ecosystems are met in a coherent and unified approach which provides means for rigorous service ecosystem engineering.

6.2.3 Demarcation of the thesis

The primary motive for this thesis is to develop means for facilitation of sustainable service ecosystems. Such means have been provided, and their completeness, feasibility, applicability and utility have been demonstrated. There are a few demarcations that were made during this work that are especially worth mentioning. The demarcations were made to keep the amount of conceptual and implementation work in the bounds of an academic thesis.

First of all, integration of domain-specific semantics and rules in the ecosystem models was not addressed. Domain-specific semantics and rules are now supposed to be implemented in the knowledge management infrastructure of the service ecosystems. However, integrating semantics and rules in the ecosystem models would clearly provide stronger and more explicit ecosystem definitions.

Moreover, the integrated rules could accelerate development of infrastructure services and modelling tools, since the implementation artefacts could be generated automatically from the formalized rules.

Secondly, the generation of complete domain-specific languages was not addressed in this thesis. The potential for generation of ecosystem-specific DSL:s was demonstrated by model transformations generating meta-models corresponding to ecosystem concept intensions; these meta-models correspond to definitions of the abstract syntaxes of domain-specific languages. Generation of complete DSL:s would require generation of technology-specific artefacts, such as UML profiles, OCL declarations and tool-specific extension elements (e.g. user-interface elements). In general, generation of domain-specific languages automatically in any domain would be infeasible. However, with more restricted domains such as service ecosystems and the domain-specific meta-modelling approach utilized in this work, it is likely that DSL generation could be automated quite far.

Thirdly, the model-transformations developed for this thesis exclude several technology-specific transformations that are valuable for instrumentation production-grade service ecosystems. Such model transformations were excluded, since they do not contribute to the evaluation and validation of the research results. However, for establishing sustainable service ecosystems, such engineering artefacts are required. For example, for service consumers and service engineers model transformations generating technology-specific service descriptions (i.e. WSDL or WS-BPEL) are valuable for bridging different technological domains. More over, model transformations consuming and producing SoaML-based [116] descriptions of the business networks could be provided, since efficient top-down, model-driven service development tools such as the ModelPro-tool[2] already exist for this modelling language. In cloud computing service ecosystems model transformations for generating and consuming infrastructure service deployment descriptors, such as based on the Open Cloud Computing Interface (OCCI) Infrastructure Specification, would become valuable.

Finally, no complete implementations for knowledge repositories were given by this work. A selection of the knowledge repositories have been previously implemented in the Pilarcos middleware platform [85, 87]. The technical implementability of knowledge repositories, and assessing their properties does not provide any further contributions or insight to the evaluation and validation of this work. Performance evaluation and usability concerns related to the knowledge repositories are interesting aspects, but are not related to the conceptual model and the corresponding modelling framework. Moreover, implementation of a complete knowledge management infrastructure requires considerable research and implementation work which has been addressed in EU projects involving several

[2] http://portal.modeldriven.org/project/modelpro

academic and industrial partners, such as the ModelBus [21] project. Instead, the applicability of the conceptual model and the modelling framework with respect to implementing the knowledge management framework was demonstrated by providing model transformations generating corresponding technology-specific models and other engineering artefacts (e.g. Java interfaces and Eclipse project structures). The generated products can be utilized for implementing the corresponding knowledge management infrastructure comprising inter-related repositories.

6.3 Impacts

We can analyze the impacts of this work by considering different actors in service ecosystems and what level of support is provided for their activities. First of all, the solution provided in this dissertation enables efficient development of domain-specific service ecosystems. The domain-specific meta-modeling language behind this framework is used for modeling the service ecosystem. Service ecosystem modeling can be utilized by different stakeholders during requirements gathering and design processes in cooperation. After an appropriate service ecosystem model has been designed, the resulting model is utilizable for producing ecosystem-specific meta-models, corresponding DSLs and knowledge repositories. Model-driven engineering principles are exploited for efficient generation of these artefacts.

Secondly, the framework provides means for individual service providers to join selected service ecosystems in a more flexible manner. The collection of tools, methods and modeling languages are typically specific for individual service providers based on their expertise, experience and practice. When joining a new service ecosystem, a service provider must possibly adopt new kinds of methods, tools or languages to provide services in conformance with the ecosystem. Such an intrusive adoption of new practices and expertise makes joining new service ecosystems an expensive process. However, explicit service ecosystem models, such as provided by this framework, can provide more efficient means for such adaptation by conceptual unification: organization-specific languages (and tools) can be mapped to the ones used by the ecosystem. Such mappings can be formalized as weaving models [36] and further utilized for efficient implementation of model integration [74].

Thirdly, the framework presented in this thesis can be exploited by modeling and software engineering tool providers. The domain-specific meta-modeling language for service ecosystems provides means for developing coherent families of domain-specific languages, or DSLs. Traditionally DSLs are developed one language at a time. However, in service ecosystems several languages need to be used in conjunction to describe the different viewpoints (e.g. legal enti-

ties vs. functional entities) in the service ecosystem. In the single-language-at-a-time model the correspondences between languages and consistency between viewpoints may become hard to handle due to complex dependencies between features. In this framework these complexities can be handled more efficiently, since the correspondences are formalized in the service ecosystem model. The model can be used for generating the abstract syntaxes of the individual DSLs in the corresponding language family, and especially, for creating explicit correspondence descriptions between the elements of the DSLs. Correspondences between individual viewpoint languages can be formalized with use of QVT, for example [129].

Finally, the framework developed in this thesis enables model-based management and governance of open service ecosystems. During operation phase of an open service ecosystem, the explicit and formal ecosystem architecture model is utilizable for establishing a shared, unambiguous understanding about the characteristics of the ecosystem. The characteristics include especially the behavioural and structural properties of the ecosystem, and behaviour and responsibilities expected from its members. In addition, the shared ecosystem architecture model enables potential members to assess risks and costs associated with joining the ecosystem. Such risk and cost assessment can be implemented by analysing the conformance and compatibility between local enterprise architectures and service ecosystem architectures, for example. More over, the ecosystem architecture models enable model-based interoperability approaches (see e.g. [13, 89]) as well as collaborative service ecosystem governance practices [143].

This dissertation provides facilities for enhancing interoperability management and software engineering support in service ecosystems. For enhancing interoperability management in service ecosystems, this work formalizes a top-level ontology for declaring service ecosystem-specific features. Such interoperability knowledge is utilized in service ecosystem life cycles for guaranteeing interoperable operation of service-based collaborations. Interoperability knowledge includes information about features and their mutual dependencies, and their applicability with respect to different models of collaboration, for example.

From the software engineering support perspective this work provides a comprehensive definition of the entities and features identifiable in service ecosystems. Thus, a unifying framework for defining vocabularies enabling engineering knowledge exchange about service artefacts is provided. Knowledge repositories based on a unified ecosystem model and maintaining corresponding feature information can then be utilized by developers for sharing information and enabling global software engineering practices. Especially, formalization of service ecosystem concepts as models and meta-models makes it possible for enabling development tool interoperability by integration of software engineering processes and

domain-specific languages through the ecosystem models and knowledge repositories.

Successful engagement in the envisioned open service ecosystems necessitate investments in terms of knowledge, expertise and practices from enterprises. First of all, enterprises must possess explicit knowledge about their own business and organization to enable appropriate exposure of their capabilities in service ecosystems. Analysis methods for assessing the compatibility between enterprise architectures and service ecosystem architectures should be developed to enable efficient business decision making. Secondly, expertise on model-driven engineering [144] is needed for utilizing service ecosystem architecture models fully in enterprises. For example, model-driven configuration of service-oriented architecture infrastructure and deployment of services [4] based on service ecosystem architecture models would enable efficient delivery of business services. Such configurable, model-driven SOAs would be especially valuable in scenarios where an organization operates in several service ecosystems simultaneously. Finally, enterprise governance (see e.g. [64]) practices should be extended to cater for business models based on operation in one or more service ecosystems. Moreover, service ecosystem governance [143] is a whole new field of collaborative governance, which should be aligned with enterprise governance to enable efficient service-based business. More research on service ecosystem governance must be conducted before it can be aligned with other enterprise governance practices.

The preceding investments are aligned with current trends in systems engineering, enterprise governance and service-oriented computing. In systems engineering explicit architecture models and model-driven engineering [144] are increasingly utilized for handling the complexity and enhancing respective engineering processes. Enterprise governance has been addressed for several years both in the academia [64] and industry, such as in form of SOA governance standardization [159]. Service-oriented computing [122, 151] is recognized as the paradigm for design and implementation of modern loosely-coupled distributed systems. Especially in enterprise computing environments service-oriented computing is realized with enterprise service buses [145].

6.4 Further prospects

This work contributes to the domain-specific language engineering research by providing an approach for developing families of inter-related, coherent DSLs. A domain ontology model declared for a service ecosystem can be considered as a specification of a family of domain-specific languages. For each concept intension a domain-specific language should be provided. Some parts of such DSLs could

be even generated automatically: a concept including behavioural features could be provided with a UML profile extending activity diagrams, for example.

Moreover, the ontological relationships between concepts, such as the ontological instantiation relationships or part-whole relationships, declare conformance and consistency dependencies between corresponding DSLs. Finally, the notion of knowledge repository is actually designed in such a way that different representations of the same knowledge item can be accepted to the same repository, if the repository maintains a knowledge item container associated with the corresponding representation format. From these premises we envision that our solution can be utilized for enabling DSL and modelling tool interoperability within service ecosystems.

This work provides means for creation of service engineering platforms, which facilitate agile service engineering and co-creation in service ecosystems. A framework for establishing domain-specific, technology independent service declaration languages for designing services, service composites and service bundles has been provided. The conceptual framework and the corresponding service ecosystem meta-model will be extended with constructs addressing agility of service engineering and co-creation. In this context, agility is addressed especially by reusable service engineering assets (e.g. models for service channel declarations, service composite / bundle definitions, integration patterns, engineering method fragments, and technology-specific adapters), and model-driven engineering principles (e.g. model transformations and code generation / interpretation). Co-creation of services is enabled by platform services that facilitate engineering tool and knowledge integration; for this purpose, knowledge repositories maintaining reusable service engineering assets are required in the service ecosystem. Efficient implementability of such knowledge repositories has been demonstrated as part of this thesis.

For enabling and managing open innovation [29] in future service ecosystems, the corresponding methodologies and business models need to be grounded inherently on the properties of the surrounding service ecosystem. From the methodological perspective, service providers need to be provided with facilities that enable them to collect feedback information and knowledge about user reactions and needs with respect to new services. Such mechanisms need to be grounded on a solid basis and they need to be ubiquitous in the sense that their existence and use does not hinder the actual functionality of the system, or do not propose too strict limitations on the usability and applicability of services.

From the business perspective, a solid foundation needs to be established for enabling the utilization of service usage information in such a way that satisfies all the participants within a service mashup business-wise. That is, a the future service ecosystem needs to be equipped with an intrinsic business model that pro-

vides added value when compared to the current situation, and enables claiming the profits within the corresponding value chain involving basic service providers, mashup providers and service end-users.

The future perspective of open service markets where mashup services (business mashup services) are created rapidly from basic service components requires mechanisms for managing the plethora of available services and service kinds. In this scenario, new services are rapidly developed by users and user communities via agile mashup technologies and methods. The basic services provided by service providers are extended and exploited by users that tailor those services to suit new kinds of usage scenarios possibly not even identified by the original provider. In such a dynamic and agile service ecosystem, the user wants to know which kind of services are available for the task she wants to execute. On the other hand, service providers want knowledge about current and forthcoming usage scenarios and needs for managing and directing their service evolution, development processes. Service and mashup providers also seek continuously for innovative uses of their services to further aid their future service development processes.

References

[1] ALDAZABAL, A., BAILY, T., NANCLARES, F., SADOVYKH, A., HEIN, C., AND RITTER, T. Automated Model Driven Development Processes. In *Proceedings of the ECMDA workshop on Model Driven Tool and Process Integration* (2008).

[2] ALMEIDA, J., DIJKMAN, R., VAN SINDEREN, M., AND PIRES, L. On the Notion of Abstract Platform in MDA Development. In *Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International* (2004), IEEE, pages 253–263.

[3] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I., AND ZAHARIA, M. A View of Cloud Computing. *Commun. ACM 53* (April 2010), pages 50–58.

[4] ARNOLD, W., EILAM, T., KALANTAR, M., KONSTANTINOU, A. V., AND TOTOK, A. A. Pattern Based SOA Deployment. In *Service-Oriented Computing – ICSOC 2007* (2007), vol. 4749 of *Lecture Notes in Computer Science*, Springer, pages 1–12.

[5] ASSMANN, U., ZSCHALER, S., AND WAGNER, G. Ontologies, Metamodels, and the Model-Driven Paradigm. In *Ontologies for Software Engineering and Software Technology*, C. Calero, F. Ruiz, and M. Piattini, Eds. Springer Berlin / Heidelberg, 2006, pages 249–273.

[6] ASUNCION, C., AND VAN SINDEREN, M. Pragmatic Interoperability: A Systematic Review of Published Definitions. In *Enterprise Architecture, Integration and Interoperability*, P. Bernus, G. Doumeingts, and M. Fox, Eds., vol. 326 of *IFIP Advances in Information and Communication Technology*. Springer Boston, 2010, pages 164–175.

[7] ATKINSON, C., AND KÜHNE, T. The Essence of Multilevel Metamodeling. In *UML 2001 - The Unified Modeling Language. Modeling Languages,*

Concepts, and Tools, M. Gogolla and C. Kobryn, Eds., vol. 2185 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2001, pages 19–33.

[8] ATKINSON, C., AND KÜHNE, T. Model-Driven Development: A Metamodeling Foundation. *IEEE Softw. 20*, 5 (2003), pages 36–41.

[9] BAIDA, Z. S. *Software-aided Service Bundling: Intelligent Methods & Tools for Graphical Service Modeling*. PhD thesis, Vrije Universiteit, 2006. SIKS Dissertation Series No. 2006-06.

[10] BANERJEE, P., FRIEDRICH, R., BASH, C., GOLDSACK, P., HUBERMAN, B., MANLEY, J., PATEL, C., RANGANATHAN, P., AND VEITCH, A. Everything as a Service: Powering the New Information Economy. *Computer 44*, 3 (march 2011), pages 36–43.

[11] BARBERO, M., JOUAULT, F., GRAY, J., AND BÉZIVIN, J. A Practical Approach to Model Extension. In *ECMDA-FA 2007* (2007), vol. 4530 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 32–42.

[12] BENGUARIA, G., LARRUCEA, X., ELVESAETER, B., NEPLE, T., BEARDSMORE, A., AND FRIESS, M. A Platform Independent Model for Service Oriented Architectures. In *Enterprise Interoperability: New Challenges and Approaches* (Apr. 2007), G. Doumeingts, J. Müller, G. Morel, and B. Vallespir, Eds., Springer, pages 23–32.

[13] BERRE, A., ELVESÆTER, B., FIGAY, N., GUGLIELMINA, C., JOHNSEN, S., KARLSEN, D., KNOTHE, T., AND LIPPE, S. The ATHENA Interoperability Framework. *Enterprise Interoperability II* (2007), pages 569–580.

[14] BERRE, A. J. UPMS - UML Profile and Metamodel for Services - an Emerging Standard. In *EDOC '08: Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference* (Washington, DC, USA, 2008), IEEE Computer Society.

[15] BEUCHE, D., PAPAJEWSKI, H., AND SCHRÖDER-PREIKSCHAT, W. Variability Management with Feature Models. *Science of Computer Programming 53*, 3 (2004), pages 333–352.

[16] BÉZIVIN, J., JOUAULT, F., ROSENTHAL, P., AND VALDURIEZ, P. Modeling in the Large and Modeling in the Small. In *Model Driven Architecture* (2005), vol. 3599 of *Lecture Notes in Computer Science*, Springer, pages 33–46.

[17] BÉZIVIN, J., JOUAULT, F., AND TOUZET, D. Principles, Standards and Tools for Model Engineering. In *ICECCS '05: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)* (Washington, DC, USA, 2005), IEEE Computer Society, pages 28–29.

[18] BÉZIVIN, J., JOUAULT, F., AND VALDURIEZ, P. On the Need for Megamodels. In *OOPSLA workshop on Best Practices for Model-Driven Software Development* (2004).

[19] BÉZIVIN, J., SOLEY, R. M., AND VALLECILLO, A. Editorial to the Proceedings of the First International Workshop on Model-Driven Interoperability. In *Proceedings of the First International Workshop on Model-Driven Interoperability (MDI'10)* (New York, NY, USA, 2010), ACM.

[20] BLANC, X., GERVAIS, M.-P., AND SRIPLAKICH, P. Model Bus: Towards the Interoperability of Modelling Tools. In *Model Driven Architecture* (2005), vol. 3599 of *Lecture Notes in Computer Science*, Springer, pages 17–32.

[21] BLANC, X., GERVAIS, M.-P., AND SRIPLAKICH, P. Modeling Services and Web Services: Application of ModelBus. In *International Conference on Software Engineering Research and Practice (SERP'05)* (2005).

[22] BOSCH, J., AND BOSCH-SIJTSEMA, P. From Integration to Composition: On the Impact of Software Product Lines, Global Development and Ecosystems. *J. Syst. Softw. 83*, 1 (2010), pages 67–76.

[23] BROGI, A., CANAL, C., AND PIMENTEL, E. Behavioural Types and Component Adaptation. In *10th International Conference on Algebraic Methodology and Software Technology (AMAST 2004)* (2004), vol. 3116 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 42–56.

[24] BRUNELIÉRE, H., ALLILAIRE, F., BÉZIVIN, J., AND JOUAULT, F. Global Model Management in Eclipse GMT/AM3. In *Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference* (2006).

[25] BRUNELIERE, H., CABOT, J., CLASEN, C., JOUAULT, F., AND BÉZIVIN, J. Towards Model Driven Tool Interoperability: Bridging Eclipse and Microsoft Modeling Tools. *Modelling Foundations and Applications* (2010), pages 32–47.

[26] BUCKL, S., MATTHES, F., ROTH, S., SCHULZ, C., AND SCHWEDA, C. M. A Conceptual Framework for Enterprise Architecture Design. In

Trends in Enterprise Architecture Research, E. Proper, M. M. Lankhorst, M. Schönherr, J. Barjis, S. Overbeek, W. Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, and C. Szyperski, Eds., vol. 70 of *Lecture Notes in Business Information Processing*. Springer Berlin Heidelberg, 2010, pages 44–56.

[27] CAMARINHA-MATOS, L. M., AND AFSARMANESH, H. Collaborative Networks: Value Creation in Knowledge Society. In *PROLAMAT 2006, Knowledge Enterprise: Intelligent Strategies in Product Design, Manufacturing, and Management* (2006), vol. 207, pages 26–40.

[28] CAMARINHA-MATOS, L. M., AND AFSARMANESH, H. A Framework for Virtual Organization Creation in a Breeding Environment. *Annual Reviews in Control 31*, 1 (2007), pages 119 – 135.

[29] CHESBROUGH, H. *Open Innovation: The New Imperative for Creating and Profiting from Technology*. Harvard Business School Press, 2003.

[30] CHESBROUGH, H. *Open Services Innovation: Rethinking Your Business to Grow and Compete in a New Era*. Jossey-Bass, 2011.

[31] CHRISTENSEN, E., CURBERA, F., MEREDITH, G., AND WEERAWARANA, S. *Web Services Description Language (WSDL) 1.1*, 1.1 ed. W3C, Mar. 2001.

[32] COLOMBO, M., DI NITTO, E., DI PENTA, M., DISTANTE, D., AND ZUCCALA, M. Speaking a Common Language: A Conceptual Model for Describing Service-oriented Systems. In *ICSOC 2005* (2005), vol. 3826 of *Lecture Notes in Computer Science*, Springer, pages 48–60.

[33] CZARNECKI, K., ANTKIEWICZ, M., KIM, C. H. P., LAU, S., AND PIETROSZEK, K. Model-driven Software Product Lines. In *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (New York, NY, USA, 2005), OOPSLA '05, ACM, pages 126–127.

[34] DEL FABRO, M., BÉZIVIN, J., AND VALDURIEZ, P. Model-Driven Tool Interoperability: An Application in Bug Tracking. *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE* (2006), pages 863–881.

[35] DEMIREZEN, Z., SUN, Y., GRAY, J., AND JOUAULT, F. Enabling Tool Reuse and Interoperability Through Model-driven Engineering. *Journal of Computational Methods in Science and Engineering 10*, Supplement 2 (2010), pages 187–202.

[36] DIDONET DEL FABRO, M., AND JOUAULT, F. Model Transformation and Weaving in the AMMA Platform. In *Workshop on Generative and Transformational Techniques in Software Engineering (GTTSE)* (2005), pages 71–77.

[37] DIKAIAKOS, M., KATSAROS, D., MEHRA, P., PALLIS, G., AND VAKALI, A. Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *Internet Computing, IEEE 13*, 5 (sept.-oct. 2009), pages 10–13.

[38] Eclipse - An Open Development Platform. http://www.eclipse.org/, 2008.

[39] Eclipse Modeling Framework website. http://www.eclipse.org/modeling/emf/, 2008.

[40] ELVESAETER, B., TAGLINO, F., GROSSO, E., BENGURIA, G., AND CAPELLINI, A. Towards Enterprise Interoperability Service Utilities. In *Enterprise Distributed Object Computing Conference Workshops, 2008 12th* (sept. 2008), pages 224 –229.

[41] European Commission – CORDIS – Seventh Framework Programme (FP7) – Information and Communication Technologies. http://cordis.europa.eu/fp7/ict/programme/home_en.html, June 2012.

[42] European Commission : CORDIS : FP7 : ICT : Service & Software Architectures and Infrastructures (SSAI) : Home. http://cordis.europa.eu/fp7/ict/ssai/home_en.html, June 2012.

[43] Internet of Things Europe. http://www.internet-of-things.eu/, June 2012.

[44] FAVRE, J. Towards a Basic Theory to Model Model Driven Engineering. In *3rd Workshop in Software Model Engineering in conjunction with UML2004, WiSME* (2004).

[45] FAVRE, J.-M. Foundations of Model (Driven) (Reverse) Engineering : Models - Episode I: Stories of The Fidus Papyrus and of The Solarus. In *Language Engineering for Model-Driven Software Development* (2004).

[46] FISCHER, G., AND OSTWALD, J. Knowledge Management: Problems, Promises, Realities, and Challenges. *IEEE Intelligent Systems 16*, 1 (2001), pages 60–72.

[47] CHOREOS – Large Scale Choreographies for the Future Internet – Research Project Website. http://www.choreos.eu/, June 2012.

[48] The Cloud4SOA Research Project web site. http://www.cloud4soa.eu/, June 2012.

[49] The INDENICA Research Project web site. http://www.indenica.eu/, June 2012.

[50] SOA4All – Web Principles web page. http://www.soa4all.eu/webprinciples.html, June 2012.

[51] SOA4All web page. http://www.soa4all.eu/home.html, June 2012.

[52] European Commission: FP7 ICT Work Programme 2011-2012. Software & Service Architectures and Infrastructure – Towards the Internet of Services. http://cordis.europa.eu/fp7/ict/ssai/home_en.html, Apr. 2012.

[53] FRANKEL, D. S. *Model Driven Architecture: Applying MDA to Enterprise Computing*. OMG Press, 2003.

[54] GASEVIC, D., KAVIANI, N., AND HATALA, M. On Metamodeling in Megamodels. In *Model Driven Engineering Languages and Systems* (2007), vol. 4735 of *Lecture Notes in Computer Science*, Springer, pages 91–105.

[55] GAY, S., AND HOLE, M. Types and Subtypes for Client-Server Interactions. *Lecture Notes in Computer Science 1576* (1999), pages 74–90.

[56] GOTH, G. Critics Say Web Services Need a REST. *IEEE Distributed Systems Online 5*, 12 (2004), 1.

[57] GREEFHORST, D., PROPER, E., GREEFHORST, D., AND PROPER, E. Introduction. In *Architecture Principles*, vol. 4 of *The Enterprise Engineering Series*. Springer Berlin Heidelberg, 2011, pages 1–6.

[58] GREEFHORST, D., PROPER, E., GREEFHORST, D., AND PROPER, E. The Role of Enterprise Architecture. In *Architecture Principles*, vol. 4 of *The Enterprise Engineering Series*. Springer Berlin Heidelberg, 2011, pages 7–29.

[59] GUIZZARDI, G. On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models. In *Databases and Information Systems IV - Selected Papers from the Seventh International Baltic Conference DB&IS'2006* (2007), vol. 155 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pages 18–39.

[60] HEIN, C., RITTER, T., AND WAGNER, M. Model-Driven Tool Integration with ModelBus. In *Workshop Future Trends of Model-Driven Development* (2009).

[61] HERBSLEB, J. D. Global Software Engineering: The Future of Sociotechnical Coordination. In *FOSE '07: 2007 Future of Software Engineering* (Washington, DC, USA, 2007), IEEE Computer Society, pages 188–198.

[62] HEVNER, A., MARCH, S., PARK, J., AND RAM, S. Design Science in Information Systems Research. *Mis Quarterly 28*, 1 (2004), pages 75–105.

[63] HONDA, K., VASCONCELOS, V. T., AND KUBO, M. Language Primitives and Type Discipline for Structured Communication-based Programming. In *Proceedings of the 7th European Symposium on Programming* (1998), Springer-Verlag, pages 122–138.

[64] HOOGERVORST, J. A. P. *Enterprise Governance and Enterprise Engineering (The Enterprise Engineering Series)*, 1 ed. Springer, Feb. 2009.

[65] HOOGERVORST, J. A. P. Introduction Enterprise Governance and Enterprise Engineering. In *Enterprise Governance and Enterprise Engineering*, The Enterprise Engineering Series. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, ch. 1, pages 3–24.

[66] ISO/IEC JTC1. *Information Technology – Open Distributed Processing – Reference Model – Enterprise Language*, 2002.

[67] ISO/IEC JTC1/SC7. *ISO/IEC 10746-2: Information Technology – Open Distributed Processing – Reference model: Foundations*, Sept. 1996.

[68] ISO/IEC JTC1/SC7. *ISO/IEC 10746-3: Information Technology – Open Distributed Processing – Reference model: Architecture*, Sept. 1996.

[69] ISO/IEC JTC1/SC7. *ISO/IEC 10746-1: Information Technology – Open Distributed Processing – Reference model: Overview*, Dec. 1998.

[70] ISO/IEC JTC1/SC7. *ISO/IEC 10746-4: Information technology – Open Distributed Processing – Reference model: Architectural Semantics*, Dec. 1998.

[71] ISO/IEC JTC1/SC7. *ISO/IEC 19793: Information technology – Open Distributed Processing –Use of UML for ODP system specifications*, Oct. 2009. ITU-T Recommendation X.906 | ISO/IEC 19793. Version 02.01.

[72] ISO/IEC JTC1/SC7/WG42. *ISO/IEC FDIS 42010 – Architecture Description*, final (v30d) ed., Mar. 2011.

[73] JOSSIC, A., DEL FABRO, M., LERAT, J.-P., BÉZIVIN, J., AND JOUAULT, F. Model Integration with Model Weaving: a Case Study in System Architecture. In *Systems Engineering and Modeling, 2007. ICSEM '07. International Conference on* (march 2007), pages 79–84.

[74] JOSSIC, A., DIDONET DEL FABRO, M., LERAT, J., BÉZIVIN, J., AND JOUAULT, F. Model Integration with Model Weaving: a Case Study in System Architecture. In *International Conference on Systems Engineering and Modeling (ICSEM'07)* (Mar. 2007), IEEE, pages 79–84.

[75] JOUAULT, F., AND BÉZIVIN, J. KM3: A DSL for Metamodel Specification. In *Formal Methods for Open Object-Based Distributed Systems* (2006), vol. 4037 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 171–185.

[76] KARTSEVA, V., GORDIJN, J., AND TAN, Y. Inter-Organisational Controls as Value Objects in Network Organisations. In *Advanced Information Systems Engineering* (2006), Springer, pages 336–350.

[77] KAZMAN, R., KLEIN, M., AND CLEMENTS, P. ATAM: Method for Architecture Evaluation: ATAM - Architecture Trade-off Analysis Method report. Tech. Rep. CMU/SEI-2000-TR-004, Carnegie Mellon Software Engineering Institute, 2002.

[78] KERRIGAN, M., MOCAN, A., SIMPERL, E., AND FENSEL, D. Modeling Semantic Web Services with the Web Service Modeling Toolkit. *J. Netw. Syst. Manage. 17*, 3 (2009), pages 326–342.

[79] KOPECKY, J., VITVAR, T., BOURNEZ, C., AND FARRELL, J. SAWSDL: Semantic Annotations for WSDL and XML Schema. *Internet Computing, IEEE 11*, 6 (nov.-dec. 2007), pages 60–67.

[80] KOTOK, A., AND WEBBER, D. R. R. *ebXML: The New Global Standard for Doing Business Over the Internet*. New Riders, Boston, 2001.

[81] KRUMMENACHER, R., NORTON, B., SIMPERL, E., AND PEDRINACI, C. SOA4All: Enabling Web-scale Service Economies. In *Semantic Computing, 2009. ICSC '09. IEEE International Conference on* (sept. 2009), pages 535–542.

[82] KÜHNE, T. Matters of (Meta-) Modeling. *Software and Systems Modeling (SoSyM) 5*, 17 (Dec. 2006), pages 369–385.

[83] KURTEV, I., BÉZIVIN, J., AND AKSIT, M. Technological Spaces: An Initial Appraisal. *CoopIS, DOA'2002 Federated Conferences, Industrial track* (2002).

[84] KUTVONEN, L. Challenges of Collaborative and Interoperable Computing – A Working Version. Research strategy of the CINCO group: http://cinco.cs.helsinki.fi, Jan. 2007.

[85] KUTVONEN, L., METSO, J., AND RUOHOMAA, S. From Trading to eCommunity Management: Responding to Social and Contractual Challenges. *Information Systems Frontiers (ISF) - Special Issue on Enterprise Services Computing: Evolution and Challenges 9*, 2–3 (July 2007), pages 181–194.

[86] KUTVONEN, L., RUOKOLAINEN, T., AND METSO, J. Interoperability Middleware for Federated Business Services in web-Pilarcos. *International Journal of Enterprise Information Systems, Special issue on Interoperability of Enterprise Systems and Applications 3*, 1 (Jan. 2007), pages 1–21.

[87] KUTVONEN, L., RUOKOLAINEN, T., RUOHOMAA, S., AND METSO, J. Service-oriented Middleware for Managing Inter-enterprise Collaborations. In *Global Implications of Modern Enterprise Information Systems: Technologies and Applications* (Dec. 2008), Advances in Enterprise Information Systems (AEIS), IGI Global, pages 209–241.

[88] LANKHORST, M., AND LANKHORST, M. Introduction to Enterprise Architecture. In *Enterprise Architecture at Work*, The Enterprise Engineering Series. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, ch. 1, pages 1–11.

[89] LEMRABET, Y., BIGAND, M., CLIN, D., BENKELTOUM, N., AND BOUREY, J.-P. Model Driven Interoperability in Practice: Preliminary Evidences and Issues from an Industrial Project. In *Proceedings of the First International Workshop on Model-Driven Interoperability* (New York, NY, USA, 2010), MDI '10, ACM, pages 3–9.

[90] LININGTON, P. F., MILOSEVIC, Z., TANAKA, A., AND VALLECILLO, A. *Building Enterprise Systems with ODP - An Introduction to Open Distributed Processing*. Innovation in Software Engineering and Software Development. Chapman & Hall/CRC Press, Sept. 2011. ISBN: 978-1-4398-6625-2.

[91] LISKOV, B. H., AND WING, J. M. A Behavioral Notion of Subtyping. *ACM Trans. Program. Lang. Syst. 16*, 6 (1994), pages 1811–1841.

[92] LIZCANO, D., JIMÉNEZ, M., SORIANO, J., CANTERA, J. M., REYES, M., HIERRO, J. J., GARIJO, F., AND TSOUROULAS, N. Leveraging the Upcoming Internet of Services through an Open User-Service Front-End Framework. In *ServiceWave '08: Proceedings of the 1st European Conference on Towards a Service-Based Internet* (Berlin, Heidelberg, 2008), Springer-Verlag, pages 147–158.

[93] LOMUSCIO, A. R., WOOLDRIDGE, M., AND JENNINGS, N. R. A Classification Scheme for Negotiation in Electronic Commerce. *Lecture Notes in Computer Science 1991* (Jan. 2001).

[94] LOPEZ-SANZ, M., ACUNA, C., CUESTA, C., AND MARCOS, E. Defining Service-Oriented Software Architecture Models for a MDA-based Development Process at the PIM level. In *Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference on* (18-21 2008), pages 309–312.

[95] LOUTAS, N., PERISTERAS, V., BOURAS, T., KAMATERI, E., ZEGINIS, D., AND TARABANIS, K. Towards a Reference Architecture for Semantically Interoperable Clouds. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on* (2010), pages 143 –150.

[96] LUDWIG, H., KELLER, A., DAN, A., KING, R., AND FRANCK, R. A Service Level Agreement Language for Dynamic Electronic Services. *Electronic Commerce Research 3*, 1-2 (2003), pages 43–59.

[97] MARGARIA, T., AND STEFFEN, B. Service Engineering: Linking Business and IT. *Computer 39*, 10 (2006), pages 45–55.

[98] MARKS, E. A. *Service-oriented architecture governance for the services driven enterprise*. John Wiley & Sons, Inc., 2008.

[99] MARTIN, D., AND DOMINGUE, J. Semantic Web Services, part 1. *Intelligent Systems, IEEE 22*, 5 (sept.-oct. 2007), pages 12–17.

[100] McCarthy, D., and Dayal, U. The Architecture of an Active Database Management System. *SIGMOD Rec. 18*, 2 (June 1989), pages 215–224.

[101] Mejía, R., López, A., and Molina, A. Experiences in Developing Collaborative Engineering Environments: An Action Research Approach. *Comput. Ind. 58*, 4 (2007), pages 329–346.

[102] Milosevic, Z., Josang, A., Dimitrakos, T., and Patton, M. A. Discretionary Enforcement of Electronic Contracts. In *EDOC '02: Proceedings of the Sixth International Enterprise Distributed Object Computing Conference (EDOC'02)* (Washington, DC, USA, 2002), IEEE Computer Society, pages 39–50.

[103] Mirbel, I., and Ralyté, J. Situational Method Engineering: Combining Assembly-based and Roadmap-driven Approaches. *Requirements Engineering 11*, 1 (2006), pages 58–78.

[104] MODELling Solution for SoftWARE Systems (MODELWARE). http://www.modelware-ist.org, 2006. IST Project 511731 (completed).

[105] Murugesan, S. Understanding Web 2.0. *IT Professional 9*, 4 (july-aug. 2007), pages 34 –41.

[106] NAICS. *North American Industry Classification System (NAICS)*. http://www.naics.com.

[107] The NESSI website. http://www.nessi-europe.com, Nov. 2009.

[108] NESSI Consortium. *NEXOF Reference Architecture Model V2.0*, Mar. 2009.

[109] OASIS. *Service Component Architecture (SCA)*, Nov. 2011. http://www.oasis-opencsa.org/sca.

[110] Object Management Group. *UML 2.0 OCL Specification*, 2.0 ed., Oct. 2003. OMG Final Adopted Specification – ptc/03-10-14.

[111] Object Management Group. *Unified Modeling Language: Superstructure*, 2 ed., Aug. 2005.

[112] Object Management Group. *Meta Object Facility (MOF) Core Specification*, 2.0 ed., Jan. 2006. OMG Available Specification – formal/06-01-01.

[113] Object Management Group. *Unified Modeling Language: Infrastructure*, 2 ed., Mar. 2006.

[114] OBJECT MANAGEMENT GROUP. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*, 2.0 ed., July 2007. Final Adopted Specification – ptc/07-07-07.

[115] OBJECT MANAGEMENT GROUP. *MOF Model to Text Transformation Language*, 1.0 ed., Jan. 2008. OMG Available Specification.

[116] OBJECT MANAGEMENT GROUP. *Service oriented architecture Modeling Language (SoaML)*, Dec. 2009. http://www.omg.org/spec/SoaML/.

[117] THE OPEN GROUP. *SOA Governance Framework – Draft Technical Standard*, 2009.

[118] THE OPEN GROUP. *The Open Group Architecture Framework*, Oct. 2011.

[119] THE OPEN GROUP. *ArchiMate 2.0 Specification*, Jan. 2021. Technical Standard.

[120] OSGI ALLIANCE. *OSGi Alliance Specifications*, Nov. 2011. http://www.osgi.org/Specifications/HomePage.

[121] OWL-S COALITION. *OWL-S 1.1 Release*, Nov. 2004.

[122] PAPAZOGLOU, M. P., AND GEORGAKOPOULOS, D. Service-oriented Computing. *Commun. ACM 46*, 10 (2003), pages 24–28.

[123] PAPAZOGLOU, M. P., AND HEUVEL, W.-J. Service Oriented Architectures: Approaches, Technologies and Research Issues. *The VLDB Journal 16*, 3 (2007), pages 389–415.

[124] PAPAZOGLOU, M. P., AND VAN DEN HEUVEL, W.-J. Business Process Development Life Cycle Methodology. *Commun. ACM 50*, 10 (2007), pages 79–85.

[125] PARREIRAS, F. S., STAAB, S., AND WINTER, A. On Marrying Ontological and Metamodeling Technical Spaces. In *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering* (New York, NY, USA, 2007), ACM, pages 439–448.

[126] QUARTEL, D. A., STEEN, M. W., POKRAEV, S., AND SINDEREN, M. J. COSMO: A Conceptual Framework for Service Modelling and Refinement. *Information Systems Frontiers 9*, 2-3 (2007), pages 225–244.

[127] RALYTÉ, J., DENECKÈRE, R., AND ROLLAND, C. Towards a Generic Model for Situational Method Engineering. In *Proceedings of the 15th international conference on Advanced information systems engineering*, J. Eder and M. Missikoff, Eds., vol. 2681 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pages 95–110.

[128] ROMAN, D., KELLER, U., LAUSEN, H., DE BRUIJN, J., LARA, R., STOLLBERG, M., POLLERES, A., FEIER, C., BUSSLER, C., AND FENSEL, D. Web Service Modeling Ontology. *Appl. Ontol. 1*, 1 (2005), pages 77–106.

[129] ROMERO, J. R., JAÉN, J. I., AND VALLECILLO, A. Realizing Correspondences in Multi-Viewpoint Specifications. In *IEEE International Enterprise Distributed Object Computing Conference* (2009), IEEE, pages 163–172.

[130] ROSETTANET CONSORTIUM. Rosettanet Implementation Framework: Core Specification v02.00.00, 2004. http://www.rosettanet.org/.

[131] RUOHOMAA, S., AND KUTVONEN, L. Trust Management Survey. In *Proceedings of the iTrust 3rd International Conference on Trust Management, 23–26, May, 2005, Rocquencourt, France* (May 2005), Springer-Verlag, LNCS 3477/2005, pages 77–92.

[132] RUOHOMAA, S., AND KUTVONEN, L. Trust and Distrust in Adaptive Inter-enterprise Collaboration Management. *Journal of Theoretical and Applied Electronic Commerce Research, Special Issue on Trust and Trust Management 5*, 2 (Aug. 2010), pages 118–136.

[133] RUOHOMAA, S., KAUR, P., AND KUTVONEN, L. From Subjective Reputation to Verifiable Experiences - Augmenting Peer-control Mechanisms for Open Service Ecosystems. In *Trust Management VI* (2012), Springer, pages 142–157.

[134] RUOKOLAINEN, T. Type-based Validation and Management of Business Service Interoperability. In *IBM Ph.D. Symposium at ICSOC 2007* (Vienna, Austria, Sept. 2007), pages 13–18.

[135] RUOKOLAINEN, T. Modelling Framework for Interoperability Management in Collaborative Computing Environments. Tech. Rep. C-2009-9, Department of Computer Science, University of Helsinki, June 2009. Licentiate's thesis.

[136] RUOKOLAINEN, T. A Model-Driven Approach to Service Ecosystem Engineering (web page). http://www.cs.helsinki.fi/u/thruokol/see/, Feb. 2013. Web page providing the complete service ecosystem meta-model representations, tool implementations and case studies for model-driven service ecosystem engineering.

[137] RUOKOLAINEN, T., AND KUTVONEN, L. Addressing Autonomy and Interoperability in Breeding Environments. In *Network-Centric Collaboration and Supporting Frameworks* (Helsinki, Finland, Sept. 2006), L. Camarinha-Matos, H. Afsarmanesh, and M. Ollus, Eds., vol. 224 of *IFIP International Federation for Information Processing*, Springer, pages 481–488.

[138] RUOKOLAINEN, T., AND KUTVONEN, L. Interoperability in Service-Based Communities. In *Business Process Management Workshops: BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS* (2006), C. Bussler and A. Haller, Eds., vol. 3812 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 317–328.

[139] RUOKOLAINEN, T., AND KUTVONEN, L. Service Typing in Collaborative Systems. In *Enterprise Interoperability: New Challenges and Approaches* (Apr. 2007), G. Doumeingts, J. Müller, G. Morel, and B. Vallespir, Eds., Springer, pages 343–354.

[140] RUOKOLAINEN, T., AND KUTVONEN, L. Managing Interoperability Knowledge in Open Service Ecosystems. In *Enterprise Distributed Object Computing Conference Workshops* (2009), V. Tosic, Ed., pages 203–211.

[141] RUOKOLAINEN, T., AND KUTVONEN, L. Framework for Managing Features of Open Service Ecosystems. In *Handbook of Research on Non-Functional Properties for Service-Oriented Systems: Future Directions*, S. Reiff-Marganiec and M. Tilly, Eds. IGI Global, Dec. 2011.

[142] RUOKOLAINEN, T., NAUDET, Y., AND LATOUR, T. An Ontology of Interoperability in Inter-enterprise Communities. In *Enterprise Interoperability II — New Challenges and Approaches* (Funchal, Portugal, Mar. 2007), Springer, pages 159–170.

[143] RUOKOLAINEN, T., RUOHOMAA, S., AND KUTVONEN, L. Solving Service Ecosystem Governance. In *Proceedings of the 15th IEEE International EDOC Conference Workshops* (Aug. 2011), IEEE Computer Society, pages 18–25.

[144] SCHMIDT, D. C. Model-Driven Engineering. *Computer 39*, 2 (Feb. 2006), pages 25–31.

[145] SCHMIDT, M.-T., HUTCHISON, B., LAMBROS, P., AND PHIPPEN, R. The Enterprise Service Bus: Making Service-oriented Architecture Real. *IBM Syst. J. 44*, 4 (2005), pages 781–797.

[146] SCHROTH, C., AND JANNER, T. Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services. *IT Pro* (May 2007), pages 36–41.

[147] The SeCSE website. http://www.secse-project.eu/, Nov. 2009.

[148] SHADBOLT, N., HALL, W., AND BERNERS-LEE, T. The Semantic Web Revisited. *Intelligent Systems, IEEE 21*, 3 (jan.-feb. 2006), pages 96–101.

[149] SHEN, Y., MIETTINEN, M., MOEN, P., AND KUTVONEN, L. Privacy Preservation Approach in Service Ecosystems. In *Proceedings of the 15th IEEE International EDOC Conference Workshops* (Helsinki, Finland, Aug. 2011), IEEE Computer Society, pages 283–292.

[150] SINGH, M. P. An Ontology for Commitments in Multiagent Systems. *Artificial Intelligence and Law 7*, 1 (1999), pages 97–113.

[151] SINGH, M. P., AND HUHNS, M. N. *Service-Oriented Computing: Semantic, Processes, Agents*. John Wiley & Sons, Ltd., West Sussex, England, 2005.

[152] SKENE, J., LAMANNA, D. D., AND EMMERICH, W. Precise Service Level Agreements. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering* (Washington, DC, USA, 2004), IEEE Computer Society, pages 179–188.

[153] SKENE, J., SKENE, A., CRAMPTON, J., AND EMMERICH, W. The Monitorability of Service-level Agreements for Application-service Provision. In *WOSP '07: Proceedings of the 6th international workshop on Software and performance* (New York, NY, USA, 2007), ACM Press, pages 3–14.

[154] OASIS Reference Model for Service Oriented Architecture 1.0. www.oasis-open.org/committees/soa-rm/, Oct. 2006.

[155] SRINIVASAN, S., AND GETOV, V. Navigating the Cloud Computing Landscape: Technologies, Services, and Adopters. *Computer 44*, 3 (march 2011), pages 22–23.

[156] STOJANOVIC, Z., AND DAHANAYAKE, A., Eds. *Service-Oriented Software System Engineering: Challenges and Practices*. Idea Group Publishing, 2005.

[157] TAGG, R. Workflow in Different Styles of Virtual Enterprise. In *ITVE '01: Proceedings of the workshop on Information technology for virtual enterprises* (Washington, DC, USA, 2001), IEEE Computer Society, pages 21–28.

[158] THIRIOUX, X., COMBEMALE, B., CRÉGUT, X., AND GAROCHE, P.-L. A Framework to Formalize the MDE Foundations. In *Workshop on Towers of Models co-located with TOOLS Europe* (June 2007), R. F. Paige and J. Bézivin, Eds., pages 14–30.

[159] SOA Governance – The Open Group. http://www.opengroup.org/projects/soa-governance, June 2012.

[160] TSAI, W.-T., WEI, X., ADN JEN-YAO CHUNG, R. P., HUANG, Q., AND CHEN, Y. Service-oriented System Engineering (SOSE) and its Applications to Embedded System Development. *Service Oriented Computing and Applications 1*, 1 (Apr. 2007), pages 3–17.

[161] U.S DEPARTMENT OF DEFENCE. *The DoDAF Architecture Framework Version 2.02*, 2.02 ed., Aug. 2010.

[162] VALLECILLO, A., VASCONCELOS, V. T., AND RAVARA, A. Typing the Behavior of Objects and Components using Session Types. *Electronic Notes in Theoretical Computer Science 68*, 3 (2003).

[163] VAN GLABBEEK, R., AND GOLTZ, U. Refinement of Actions and Equivalence Notions for Concurrent Systems. *Acta Inf. 37*, 4-5 (2000), pages 229–327.

[164] VINCENT, H., ISSARNY, V., GEORGANTAS, N., FRANCESQUINI, E., GOLDMAN, A., AND KON, F. CHOReOS: Scaling Choreographies for the Internet of the Future. In *Middleware '10 Posters and Demos Track* (New York, NY, USA, 2010), Middleware Posters '10, ACM, pages 8:1–8:3.

[165] VITVAR, T., ZAREMBA, M., AND MORAN, M. SESA: Emerging Technology for Service-centric Environments. *Software, IEEE 24*, 6 (nov.-dec. 2007), pages 56 –67.

[166] VOORSLUYS, W., BROBERG, J., AND BUYYA, R. Introduction to Cloud Computing. In *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg, and A. Goscinski, Eds. John Wiley & Sons, 2011.

[167] W3C. *OWL Web Ontology Language Guide*, Feb. 2004. W3C Recommendation 10 February 2004.

[168] W3C. *RDF Primer*, Feb. 2004. W3C Recommendation 10 February 2004.

[169] W3C. *Web Services Architecture*, Feb. 2004. W3C Working Group Note 11.

[170] W3C. *XML Schema Documentation; Part 1:Structures, Part 2: Datatypes*, 2nd ed., Oct. 2004. W3C Recommendation, http://www.w3.org/XML/Schema.

[171] W3C. *Semantic Annotations for WSDL and XML Schema*, 28 august 2007 ed., Aug. 2007. W3C Recommendation, http://www.w3.org/TR/sawsdl.

[172] W3C WORKING GROUPS. *Extensible Markup Language (XML)*. W3C, 2005. http://www.w3.org/XML/, valid 27th January 2013.

[173] World Wide Web. http://en.wikipedia.org/wiki/World_Wide_Web, valid 27th January 2013.

[174] WINSKEL, G., AND NIELSEN, M. Models for Concurrency. In *Semantic Modelling*, S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, Eds., vol. 4 of *Handbook of Logic in Computer Science*. Clarendon Press, 1995.

[175] WOLAK, R., KALAFATIS, S., AND HARRIS, P. An Investigation into Four Characteristics of Services. *Journal of Empirical Generalisations in Marketing Science 3* (1998), pages 22–41.

[176] WONG, J., AND HONG, J. What Do We "Mashup" When We Make Mashups? In *WEUSE '08: Proceedings of the 4th international workshop on End-user software engineering* (New York, NY, USA, 2008), ACM, pages 35–39.

[177] WORLD HEALTH ORGANIZATION. What is "active ageing"? http://www.who.int/ageing/active_ageing/en/index.html, valid 27th January 2013.

[178] Web Service Modeling Ontology homepage. http://www.wsmod.org/, June 2010.

[179] YU, X., ZHANG, Y., ZHANG, T., WANG, L., HU, J., ZHAO, J., AND LI, X. A Model-driven Development Framework for Enterprise Web Services. *Information Systems Frontiers 9*, 4 (2007), pages 391–409.

[180] ZDUN, U., HENTRICH, C., AND DUSTDAR, S. Modeling Process-driven and Service-oriented Architectures using Patterns and Pattern Primitives. *ACM Trans. Web 1*, 3 (2007), 14.

TIETOJENKÄSITTELYTIETEEN LAITOS
PL 68 (Gustaf Hällströmin katu 2 b)
00014 Helsingin yliopisto

DEPARTMENT OF COMPUTER SCIENCE
P.O. Box 68 (Gustaf Hällströmin katu 2 b)
FIN-00014 University of Helsinki, FINLAND

JULKAISUSARJA **A**

SERIES OF PUBLICATIONS **A**

Reports may be ordered from: Kumpula Science Library, P.O. Box 64, FIN-00014 University of Helsinki, FINLAND.

A-2007-1 P. Sarolahti: TCP Performance in Heterogeneous Wireless Networks. 171 pp. (Ph.D. Thesis)

A-2007-2 M. Raento: Exploring privacy for ubiquitous computing: Tools, methods and experiments. 61+150 pp. (Ph.D. Thesis)

A-2007-3 L. Aunimo: Methods for Answer Extraction in Textual Question Answering. 127+18 pp. (Ph.D. Thesis)

A-2007-4 T. Roos: Statistical and Information-Theoretic Methods for Data Analysis. 82+75 pp. (Ph.D. Thesis)

A-2007-5 S. Leggio: A Decentralized Session Management Framework for Heterogeneous Ad-Hoc and Fixed Networks. 230 pp. (Ph.D. Thesis)

A-2007-6 O. Riva: Middleware for Mobile Sensing Applications in Urban Environments. 195 pp. (Ph.D. Thesis)

A-2007-7 K. Palin: Computational Methods for Locating and Analyzing Conserved Gene Regulatory DNA Elements. 130 pp. (Ph.D. Thesis)

A-2008-1 I. Autio: Modeling Efficient Classification as a Process of Confidence Assessment and Delegation. 212 pp. (Ph.D. Thesis)

A-2008-2 J. Kangasharju: XML Messaging for Mobile Devices. 24+255 pp. (Ph.D. Thesis).

A-2008-3 N. Haiminen: Mining Sequential Data – in Search of Segmental Structures. 60+78 pp. (Ph.D. Thesis)

A-2008-4 J. Korhonen: IP Mobility in Wireless Operator Networks. 186 pp. (Ph.D. Thesis)

A-2008-5 J.T. Lindgren: Learning nonlinear visual processing from natural images. 100+64 pp. (Ph.D. Thesis)

A-2009-1 K. Hätönen: Data mining for telecommunications network log analysis. 153 pp. (Ph.D. Thesis)

A-2009-2 T. Silander: The Most Probable Bayesian Network and Beyond. 50+59 pp. (Ph.D. Thesis)

A-2009-3 K. Laasonen: Mining Cell Transition Data. 148 pp. (Ph.D. Thesis)

A-2009-4 P. Miettinen: Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms. 164+6 pp. (Ph.D. Thesis)

A-2009-5 J. Suomela: Optimisation Problems in Wireless Sensor Networks: Local Algorithms and Local Graphs. 106+96 pp. (Ph.D. Thesis)

A-2009-6 U. Köster: A Probabilistic Approach to the Primary Visual Cortex. 168 pp. (Ph.D. Thesis)

A-2009-7 P. Nurmi: Identifying Meaningful Places. 83 pp. (Ph.D. Thesis)

A-2009-8 J. Makkonen: Semantic Classes in Topic Detection and Tracking. 155 pp. (Ph.D. Thesis)

A-2009-9 P. Rastas: Computational Techniques for Haplotype Inference and for Local Alignment Significance. 64+50 pp. (Ph.D. Thesis)

A-2009-10 T. Mononen: Computing the Stochastic Complexity of Simple Probabilistic Graphical Models. 60+46 pp. (Ph.D. Thesis)

A-2009-11 P. Kontkanen: Computationally Effcient Methods for MDL-Optimal Density Estimation and Data Clustering. 75+64 pp. (Ph.D. Thesis)

A-2010-1 M. Lukk: Construction of a global map of human gene expression - the process, tools and analysis. 120 pp. (Ph.D. Thesis)

A-2010-2 W. Hämäläinen: Efficient search for statistically significant dependency rules in binary data. 163 pp. (Ph.D. Thesis)

A-2010-3 J. Kollin: Computational Methods for Detecting Large-Scale Chromosome Rearrangements in SNP Data. 197 pp. (Ph.D. Thesis)

A-2010-4 E. Pitkänen: Computational Methods for Reconstruction and Analysis of Genome-Scale Metabolic Networks. 115+88 pp. (Ph.D. Thesis)

A-2010-5 A. Lukyanenko: Multi-User Resource-Sharing Problem for the Internet. 168 pp. (Ph.D. Thesis)

A-2010-6 L. Daniel: Cross-layer Assisted TCP Algorithms for Vertical Handoff. 84+72 pp. (Ph.D. Thesis)

A-2011-1 A. Tripathi: Data Fusion and Matching by Maximizing Statistical Dependencies. 89+109 pp. (Ph.D. Thesis)

A-2011-2 E. Junttila: Patterns in Permuted Binary Matrices. 155 pp. (Ph.D. Thesis)

A-2011-3 P. Hintsanen: Simulation and Graph Mining Tools for Improving Gene Mapping Efficiency. 136 pp. (Ph.D. Thesis)

A-2011-4 M. Ikonen: Lean Thinking in Software Development: Impacts of Kanban on Projects. 104+90 pp. (Ph.D. Thesis)

A-2012-1 P. Parviainen: Algorithms for Exact Structure Discovery in Bayesian Networks. 132 pp. (Ph.D. Thesis)

A-2012-2 J. Wessman: Mixture Model Clustering in the Analysis of Complex Diseases. 119 pp. (Ph.D. Thesis)

A-2012-3 P. Pöyhönen: Access Selection Methods in Cooperative Multi-operator Environments to Improve End-user and Operator Satisfaction. 211 pp. (Ph.D. Thesis)

A-2012-4 S. Ruohomaa: The Effect of Reputation on Trust Decisions in Inter-enterprise Collaborations. 214+44 pp. (Ph.D. Thesis)

A-2012-5 J. Sirén: Compressed Full-Text Indexes for Highly Repetitive Collections. 97+63 pp. (Ph.D. Thesis)

A-2012-6 F. Zhou: Methods for Network Abstraction. 48+71 pp. (Ph.D. Thesis)

A-2012-7 N. Välimäki: Applications of Compressed Data Structures on Sequences and Structured Data. 73+94 pp. (Ph.D. Thesis)

A-2012-8 S. Varjonen: Secure Connectivity With Persistent Identities. 139 pp. (Ph.D. Thesis)

A-2012-9 M. Heinonen: Computational Methods for Small Molecules. 110+68 pp. (Ph.D. Thesis)

A-2013-1 M. Timonen: Term Weighting in Short Documents for Document Categorization, Keyword Extraction and Query Expansion. 53+62 pp. (PhD thesis)

A-2013-2 H. Wettig, Hannes: Probabilistic, Information-Theoretic Models for Etymological Alignment. 130+62 pp. (PhD thesis)