Framework for managing features of open service ecosystems

Toni Ruokolainen
Department of Computer Science, University of Helsinki, FINLAND

Lea Kutvonen
Department of Computer Science, University of Helsinki, FINLAND

ABSTRACT

The recent increased use of Internet, social media, and networked business mark a development trend where software-based services flow to the open market for enabling service-oriented networked business. Our vision is that in future, organizations and individuals collaborate within open service ecosystems. 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. For facilitating collaboration establishment in open service ecosystems features of services and cooperation facilities, and feature inter-dependencies need to be governed rigorously. Towards this purpose we have established a framework for unambiguous description of service ecosystem features. The framework comprises a conceptual model which provides especially a categorization of features, and a formalization of the conceptual model as a meta-model for service ecosystems. We show that the corresponding feature categories have their specific roles and semantics as part of different ecosystem elements and in different phases of service ecosystem processes.

1. INTRODUCTION

The recent increased use of Internet, social media, and networked business mark a development trend where software-based services flow to the open market. Technological approaches like SaaS, SOA, and Web Services present tools and architectures for this: they provide protocols for accessing remote functionality encapsulated to a business-relevant units, declared available though service registries and manifests of service functionality, requirements for messaging platform support, information representation and semantics, and choreography (protocol) for exchanges in utilising the service.

However, this situation is uncontrolled and uncontrollable in several ways. First, the trustworthiness of the services marketed is unknown, as there is no guaranteed knowledge (facts) about their properties. The clientele is left to rely on declarations by the service providers. The declarations carry several risk aspects. The semantic of the declaration may be obscure due to the lack of shared vocabulary for describing service behaviour in functional and nonfunctional aspects exists. Furthermore, the declarations can be biased, as the cost of inaccurate declarations is not sufficient as an incentive.

Second, the interoperability between independently developed services is immature, especially in terms of nonfunctional properties. There is no commonly accepted framework for functionality and selectable properties or property management for those functions. Middleware platforms have built-in support for various transparency properties (e.g., location and access, data representation, transactionality) and various security technologies (e.g., encryption, non-repudiation), but as the groupings of properties differ, the interworking challenges still exist. Furthermore, the concepts of nonfunctional properties commonly refer to platform services, but in modern social networking and inter-enterprise collaboration scenarios, business

and user oriented properties (such as policies for governing joint behaviour, pricing schemes, privacy preservation declarations) are relevant requirements.

Third, the current platforms weakly support collaboration management or concepts required for it. Concepts of contracts, parties, authority and ownership, policies and breaches of contracts causing sanctions are necessary for the different kinds of networked collaborations.

As a partial solution to these challenges, 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 (Bosch & Bosch-Sijtsema, 2010). Software ecosystem strategy is utilized by companies such as Amazon [i] or Nokia [ii] for establishing communities of developers and clientele over their own corresponding platforms.

While the software ecosystem approach emphasises the software production challenges, the open use of services from the open marketplace is stressed by service ecosystem approaches. A service ecosystem is an environment for creating and managing service-based collaborations, such as virtual organizations or service mash-ups, from services provided by a community of service providers. Service ecosystems exist currently especially in form of platform provider specific Software-as-a-Service (SaaS) -environments. The typical service ecosystems available currently are closed, meaning that the methods and technologies used for providing new services are pre-determined by the hosting environment, and service compositions and collaboration networks are determined statically during service development. Such closed ecosystems can not be applied in domains where services are to be provided and managed by autonomous entities, or when service collaboration networks are to be established dynamically on demand.

The challenge still remains to provide an environment where several service-oriented software engineering (SOSE) methodologies and distributed teams could produce services that easily can be organised into collaborations managed by dynamic contracts, because linkage between these two sides is missing.

The main architecture design must address a more complex situation where the clientele and the ecosystem itself have potential conflicts of interest in details, but still, the members of the ecosystem have incentives for collaboration both at business network level and ecosystem introduction, control and evolution levels.

Instead of considering nonfunctional features as a single, uniformly directed domain of research, there is definite need to address separately three levels: i) business collaboration aspects, ii) service properties, and iii) collaborative management of communication technology usage in specific cases. For the business collaboration aspects, the declaration of vocabulary and behaviour requirements is directed by companies and consortia focusing on issues including business processes, legal systems, business models. For service properties, the aspects to be managed will be implemented mainly as software – and thus design choices depend on the software engineering practices used – but the management of quality, business issues and user interaction aspects need to be aligned across the service ecosystem with the needs of the business collaboration level. Further, the communication technology needs to be governable though shared vocabulary with the level above, although the facilities are provided and designed by platform service producers.

Our vision in the CINCO group (Collaborative and Interoperable Computing research group, http://cinco.cs.helsinki.fi) is that 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 best their own needs. A set of global infrastructure services are then used for service publication and discovery, as well as dynamic establishment of service-based collaboration networks (Kutvonen, Ruokolainen, Ruohomaa, & Metso, 2008). An open service ecosystem is based on the service-oriented architecture (SOA) architectural style with service brokering and dynamic binding

facilities, but requires more sophisticated infrastructure services for enabling interoperable service collaboration.

The contribution of this Chapter is built up in steps. First we introduce the service ecosystem framework that merges the concepts for service-oriented software engineering and eContracting in such a way that ecosystem level evolution is facilitated. This provides the environment in which, as the second step, the enhanced concepts of ecosystem features can be connected to the relevant primary targets, i.e. service, collaboration and communication. Once the concepts are introduced, the third step provides the processes necessary for evolving the open service ecosystem understanding and usage of the features. This text provides insight for the Pilarcos ecosystem work, and as new contribution, introduces the feature management method within the Pilarcos ecosystem frame. Thus the text reflects the CINCO group vision that in future, individual users, enterprises or public organizations can easily compose new services from open service markets, or establish temporary collaborations with complex peer relationships. Furthermore, these contract-governed collaborations can be managed by all involved parties. All this is supported by a global infrastructure with facilities for interoperability control and contract-based community management (establishment, control and breach recovery) among autonomous organization; this infrastructure also takes responsibility of governing trust and privacy-preservation issues. The support environment is complemented with service-oriented software engineering practices that enable semantic and pragmatic interoperability management.

Section 2 introduces the open service ecosystem, while Section 3 elaborates on the ecosystem spanning from technical platforms to business-oriented needs of managing inter-enterprise collaborations. Section 3 demonstrates the conceptual model and its formalization as a meta-model hierarchy, and how the necessary concepts can be organised to achieve the goals of synchronous management of the three levels of issues described above. Section 4 proceeds then to discuss management of features in several service ecosystem processes: service ecosystem life-cycles, eContracting, and ecosystem evolution. Section 5 discusses the benefits of the framework, while conclusions are given in Section 6.

2. OPEN SERVICE ECOSYSTEM

We propose open service ecosystems as a coherent solution for the challenges of interoperability and collaboration management, met in the phases of service production and utilisation through collaborations, and innovation of new types of collaborations and services. Most of the challenges arise from the inherent and necessary independence of actors involved, including collaboration partners, parties engineering services, clients, and platform providers.

In terms of operational time composability of services and facilities of inter-enterprise collaborations, a key point is sufficient, automated support for interoperability. Interoperability concept covers technical, semantic and pragmatic interoperability aspects. Technical interoperability is concerned with connectivity between the computational services, allowing messages to be transported from one application to another. Semantic interoperability means that the message content becomes understood in the same way by the senders and the receivers. This concerns both information representation and messaging sequences. Pragmatic interoperability captures the willingness of partners to perform the actions needed for the collaboration. This willingness to participate refers both to the capability of performing a requested action, and to policies dictating whether it is preferable for the enterprise to allow that action to take place.

At the design and engineering time concerns include efficient production and maintenance of service software with clear, published interfaces and behaviour descriptions. While the direct production concern is how to extend model-based methodologies to distributed team environment, an even more pressing is the need of producing exploitable, composable services that can be managed in such a way that interoperability can be achieved at operational time. Behind these two problem areas remains the problem of changing business models and changing computing and communication platforms: both the production and operational systems should be tolerant for changes.

Against this background we define a service ecosystem as an environment for creating and managing service-based collaborations, such as virtual organizations or service mash-ups, from services provided by a community of autonomous entities. In future service-oriented networked business, organizations and individuals collaborate within service ecosystems. Currently there are several emerging service ecosystems in the domains of business and technology. Software ecosystems (Bosch & Bosch-Sijtsema, 2010) are utilized by enterprises for producing software applications more efficiently for heterogeneous clientele with varying requirements. Software ecosystems can be characterized as product centric service ecosystems. Different kinds of electronic business networking environments, such as eCommerce platforms, supply chains, and virtual organizations, can be considered as collaboration and process centric service ecosystems. Cloud computing platforms, such as provided by Amazon, Google, or Salesforce, are resource centric service ecosystems. Finally, community and individual centric service ecosystems are realized by social networking platforms such as Facebook, LinkedIn or MySpace. The maturity of these ecosystems vary from closed software systems to open collaboration systems with ad hoc collaboration models.

For us, the open service ecosystem is characterized especially by its evolution with respect to available services and collaboration types, and dynamic establishment of collaborations. Indeed, the purpose of the ecosystem is to provide infrastructure, tools and vocabulary for independent entities (people, organisations, collaborations) to create new collaborations, utilising the already existing services from the ecosystem. This is in contrast to a common goal in related work (e.g., ECOLEAD (Rabelo, Gusmeroli, Arana, & Nagellen, 2006), CrossWork (Mehandjiev & Grefen, 2010)) for creating a shared space with a shared incentives of the members. In open service ecosystems the initiators of collaborations each have their private incentives and conflicts of interest are to be expected, and resolution of such conflicts need to be supported.

The open service ecosystems are defined by a conceptual framework and a set of life cycles:

- The conceptual framework provides a vocabulary and an ontology for defining the properties of services, collaborations and entities in the ecosystem.
- A service ecosystem life cycle declares which kinds of activities are expected from the ecosystem participants to support the operation of the ecosystem. Foundational ecosystem life cycles, which exist in every service ecosystem, include a *service life cycle* and a *collaboration establishment life cycle*. Additional life cycles can be associated with a service ecosystem depending on the requirements of the domain of interest, such as *ecosystem evolution life-cycle*.

The service life cycle addresses steps of service innovation, modeling, production and utilisation. Therefore, the service-oriented software engineering methodologies and instruments must produce modules that exploit the facilities of the ecosystem infrastructure for efficient and dependable service delivery (i.e. service publication, discovery, selection, location and binding). The collaboration establishment life cycle relies on artifacts defining the content of contracts, and infrastructure mechanisms for dynamic establishment of safe, interoperable service collaborations taking qualitative requirements into consideration. The ecosystem evolution life-cycle allows the concept base and infrastructure service base to be enhanced and modified while the ecosystem is utilised. This evolution then allows existing SOSE methods and eContracting methods to facilitate the new innovations.

The Pilarcos ecosystem illustrated in Figure 1 can be considered as a concretization of the ecosystem framework. The Pilarcos architecture views inter-enterprise collaboration as a loosely-coupled, dynamic constellation of business services. The constellation is governed by an eContract that captures the business network model describing the roles and interactions of the collaboration, the member services, and policies governing the joint behavior (Kutvonen, Metso, & Ruohomaa, 2007, Kutvonen, Ruokolainen, & Metso, 2007).

The Pilarcos architecture for the open service ecosystem comprises of

1. the participating enterprises, with their public business service portfolios exported (Kutvonen, Ruokolainen, Ruohomaa, & Metso, 2008);
2. business-domain governing consortia, with their public models of business scenarios and business models expressed as exported business network models (comprising a set of business

process descriptions and compulsory associations between roles in them, and governing policies about acceptable behavior)(Kutvonen, 2002);

3. a joint ontology about vocabulary to be used for contract negotiation, commitment and control (Metso & Kutvonen, 2005, Ruokolainen & Kutvonen, 2006, 2007b);
4. legislative rules to define acceptable contracts (Metso & Kutvonen, 2005);
5. technical rules to define conformance rules over all categories of meta-information held as collaboration and interoperability knowledge (Ruokolainen, 2009, Ruokolainen & Kutvonen, 2007a);
6. infrastructure services to support partner discovery and selection, contract negotiation and commitment to new collaborations, monitoring of contracted behavior of partners, and breach detection and recovery services; these services especially include trust aspects in decision-making on commitment and breaches (Kutvonen, Metso, & Ruohomaa, 2007, Kutvonen, Ruokolainen, & Metso, 2007);
7. reputation information flow, collected from past collaborations (Ruohomaa & Kutvonen, 2008, Ruohomaa, Viljanen, & Kutvonen, 2006).

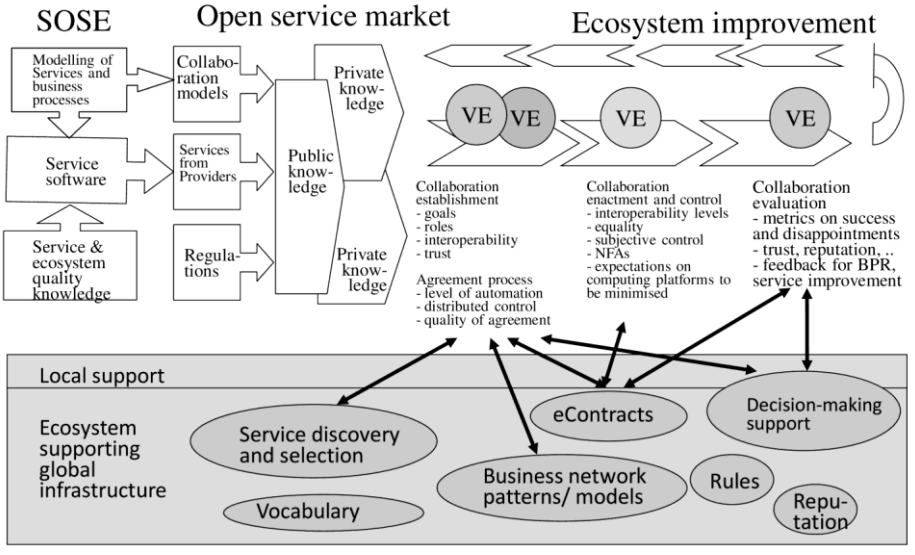

Figure 1: A schematic view of the Pilarcos service ecosystem life-cycles.

Figure 1 illustrates the ecosystem life-cycles. On the left, meta-information repositories and flows are shown to be created by the publishing and exporting processes denoted above as items 1 and 2. The repositories in particular contain public information about the available business network models, available services and reputation information about the available services. This information is stored in globally federated repositories, applying strictly specified structuring and conformance rule (Kutvonen, 2004) created by the processes listed above as items 3, 4 and 5. The information is in turn utilized by the ecosystem infrastructure functions listed as item 6, e.g. service discovery and selection, eContracting functions,

monitoring of business services and reporting of experience on the services when a collaboration terminates. These functions are further described below.

On the right, the life-cycle of independent collaborations is shown to flow from establishment to evaluation at the dissolution phase. The infrastructure functions provide support for the four phases of the collaboration: establishment, agreement, enactment and control, and evaluation. In the Pilarcos framework the collaboration establishment is a multi-lateral process involving a collaboration initiator and one or more service providers that have published their services in the service ecosystem.

Service discovery and selection supports the collaboration establishment phase. It is based on public business network models describing the collaborations, and public service offers made by service providers (Kutvonen, Metso, & Ruohomaa, 2007, Ruokolainen & Kutvonen, 2007b). The business network models capture the best practices of a given field, and they are built from formally defined service types. The task of producing these models and types naturally falls to consortia and standardization bodies.

Service selection includes automated static interoperability checking, which ensures that the service offers fit the model of the collaboration, and have terms that are compatible with other offers being selected into the proposed business network. As service discovery and selection is separate from contract negotiations, it can be done without access to sensitive information; this makes it possible to have this task implemented as a third-party service (Kutvonen, Metso, & Ruohomaa, 2007).

Automated eContract establishment supports the agreement phase of the collaboration (Kutvonen, Metso, & Ruohomaa, 2007). The business network model and the proposed service offers to populate the roles in it are processed by an automated contract negotiation infrastructure, which is controlled locally by each collaboration partner. Contracts are based on templates specific to the collaboration model, and the terms of service provision given in service offers form the basis of negotiations. The negotiated eContract includes a model of the business process of the collaboration, as well as the finalized terms of service in the form of accepted service offers.

Monitoring supports the enactment and control phase of the collaboration in particular (Kutvonen, Metso, & Ruokolainen, 2005). It is done by each collaborator to protect local resources, keep track of the progress of the collaboration, and to ensure that partners follow the collaboration model. The business network model and service provision terms set by the negotiated eContract form the specification of correct behavior in the collaboration, which becomes relatively straightforward to monitor.

Experience reporting supports the evaluation phase of the collaboration, and connects to the monitoring service during the enactment of the collaboration (Ruohomaa & Kutvonen, 2008, Ruohomaa et al., 2006). Experience reporting forms the core of social control in the open service ecosystem. As contract violations are detected by monitors, they are published to other actors as well: it is important to create a direct reputation impact to privacy and data security violations in order to limit the damage that misbehaving actors can achieve in other collaborations.

For evaluating the ecosystem approach, a set of prototype infrastructure services has been implemented in the Pilarcos interoperability middleware (Kutvonen et al., 2008). Systematic performance testing of the framework has been conducted in the context of the Pilarcos interoperability middleware. The results show that the approach is feasible performance-wise (Metso & Kutvonen, 2005). The conceptual framework and the corresponding metamodels has been scrutinized during their development with architectural analysis. For example, a threat analysis has been conducted with respect to privacy issues (Moen et al, 2010).

As can be detected experimenting with the Pilarcos type of ecosystem, the ecosystem features have several roles within service ecosystems. They are used as qualitative service features, for example during service discovery and selection. During collaboration establishment they are considered as contractual artifacts that are negotiated between entities willing to establish service-based collaborations. Finally, they are deployable products, that is artifacts created by someone using a specific process, which are put into use during collaboration enactment. Furthermore, while the set of possible service ecosystem features is open and can not be predetermined or enumerated due to their context dependency and evolution of an open service ecosystem, their usage can be disciplined by deliberate management facilities. These facilities involve design and deployment of the features, as well the operational time facilities for governing their utilization.

3. FEATURES IN SERVICE ECOSYSTEMS

Service ecosystem is a socio-technical complex systems where autonomic entities collaborate with each other over a service-oriented computing environment. The service-oriented computing environment provides infrastructure services for establishing interoperable collaborations. Collaborations are enabled by cooperation facilities, such as communication channels, that are set up during collaboration establishment for arbitrating activities and knowledge between ecosystem members. In service ecosystems the different features of entities and cooperation facilities affect the structure, behaviour and qualities of service-based collaborations.

For facilitating collaboration establishment processes and other service ecosystem processes the features and their inter-dependencies need to be governed rigorously. Towards this purpose we have established a framework for unambiguous description of service ecosystem features. The framework comprises a conceptual model which provides especially a categorization of features, and a formalization of the conceptual model as a meta-model for service ecosystems. In the following we describe the categorization of service ecosystem features. As we will see, the categorization does not actually contain a notion of non-functional features. Different features in service ecosystems govern activities taken in different phases of ecosystem life-cycles: so-called cooperative features defining intensions of legal entities are used for decision making in the preparatory phases of collaboration establishment, whereas so-called extra-functional features declare qualitative features of interaction and communication. All service ecosystem features have active roles during collaboration establishment processes (e.g. virtual organization establishment).

In this Section we discuss the characteristics of service ecosystem features and their management. We first define what we mean with feature management and discuss related problems, and activities. We then provide a categorization of service ecosystem entities and cooperation facilities that are utilized for enabling service-based collaborations. After that we define a categorization of ecosystem features. The categorization is derived from the identification of ecosystem entities and cooperation facilities, and requirements stemming from foundational ecosystem life-cycles, such as collaboration establishment life-cycles. The categorizations and the conceptual framework describing relationships between the kinds of ecosystem entities, cooperation facilities, and their features are formalized in a service ecosystem meta-model. UML class diagrams (Object Management Group, 2005) described in this Chapter illustrate parts of a larger modelling framework (Ruokolainen, 2009) for managing service ecosystem knowledge. Finally, we provide a discussion of the semantics for service ecosystem features. We perceive that each feature category is associated with a distinguishing semantic framework.

Managing service ecosystem features

Managing features of software systems is problematic: features may have complex dependencies with each other, they can be defined at different abstraction levels, and interpretation of their meaning or importance can be subjective. In addition to these generic problems, more specific challenges are introduced for feature management in open service ecosystems. In open service ecosystems new kinds of features emerge following the demands of the individual members and the domain of operation. This dynamism of the knowledge landscape must be addressed by mechanisms that allow extension of the feature ontologies. Due to the autonomy of ecosystem members feature management can not be centrally controlled. Instead, features should be managed using a federated approach where feature descriptions can be shared between ecosystem members and utilized efficiently in the local systems. Finally, for guaranteeing interoperability during dynamic collaboration establishment processes, features should be provided with rigorous and unambiguous semantics.

Features can have several kinds of inter-dependencies. When addressing features at the same level of abstraction, features may have horizontal interactions with each other. Behavioural features of business

services can be affected by security features requiring key-exchange protocols, for example. Another example of horizontal feature interaction is the potential conflict between performance and security features: introduction of communication encryption may increase the response time of business service. Features at the same level of abstraction may require other features to function correctly: introducing message encryption feature on a communication endpoint is typically not valid without introducing an decryption feature on the other communication endpoint. Finally, features may have direct conflicts with each other (e.g. monitoring of communication vs. privacy preservation), be mutually exclusive (e.g. different message encryption schemes), or have some domain specific dependencies with each other.

Features are defined in different levels of abstraction for decreasing the complexity of their description and for achieving loose coupling between business and technology. In this setting features defined at a higher level of abstraction are instantiated at a lower level of abstraction by a collection of more specific features. For example, at the business level a feature requirement for communication confidentiality is declared. This high-level feature can be implemented at a lower level of abstraction by an appropriate combination of features representing strong encryption and privacy. Further down the abstraction chain, the feature of strong encryption can be implemented by providing a feature that represents usage of RSA algorithm with 1024 bit key length, for example.

Especially in open service ecosystems the subjectivity of interpretation of feature intensions has to be addressed carefully. Without a shared understanding about the meaning of ecosystem features, identification and selection of eligible features, and analysis of feature interoperability are impossible. For this purpose, means for categorizing service ecosystem features must be provided. The categorization is utilized for classifying available features such that features providing required characteristics of business services and communication can be efficiently identified. Such feature categorization provides a basis for an ontology of service ecosystem features. In addition to prescribing feature categories, such ontology should also provide means for defining horizontal and vertical feature dependencies.

For increasing the elasticity and sustainability of the service ecosystem, an ontology describing service ecosystem features must be dynamically extensible. Especially, it should be possible to introduce new kinds of feature categories on demand. A modelling framework that is based on the powertype pattern (see e.g. (Gonzalez-Perez & Henderson-Sellers, 2006)) or some other means for dynamic type definition should be used. While it may be sufficient in other contexts to use ontologies for simply describing features, such as has been done for example in (Kabilan et al., 2007) or (Kassab, Ormandjieva, & Daneva, 2009), this is not sufficient in open service ecosystems. Instead, feature definitions need to be provided also with prescriptive definitions for reducing ambiguity in feature interpretation, and for enabling efficient feature implementation in ecosystem member organizations. Feature intensions in service ecosystem domain ontologies can be formalized as meta-models and models (Ruokolainen, 2009).

However, even defining both the descriptive and prescriptive characteristics of features is not sufficient in open service ecosystems. While such an ontology provides some guidance especially for the selection and development of required features, from the interoperability management perspective these descriptions are incomplete. What is still missing from this setting is rigorous semantics providing unambiguous interpretation of feature intensions and dependencies. This deficiency can be approached with abstract platform thinking and by providing proper semantics for the different categories of features.

An *abstract platform* represents the support that is assumed by platform-independent models of a distributed application (Almeida, Dijkman, Sinderen, & Pires, 2004). In open service ecosystems abstract platforms are made explicit by models that prescribe the characteristics of interaction and communication. From feature management perspective, these models provide a mechanism for prescribing the effects that certain features have on interaction and communication. More over, feature dependencies can be characterized with respect to abstract platform models, such as descriptions of communication channels. For example, it can be described that two specific features can not be bound simultaneously to a communication channel.

Semantics for features in service ecosystems should be formalized by using proper, category specific semantics. Behavioural features of services can be formalized using Petri-nets, process algebra, or finite state automata, for example. Structural features of communicated information can be formalized based on

different type systems. Policies and business rules, which are kinds of non-functional features, can be formalized as temporal or deontic constraints over service behaviour. Different alternatives for formalizing feature semantics are discussed below, after introducing the feature categories of service ecosystems.

Service ecosystem entities and cooperation facilities

Features in service ecosystems specify the characteristics of ecosystem entities and abstract platform components. An entity has its own existence and has an identity (e.g. a unique identifier, address, name, or URI) which can be used for referring to and identifying the corresponding entity. Examples of service ecosystem entities are organizations, individuals, business services, and service endpoints. We consider entities and features as the primary artifacts in service ecosystems, and meaning of an entity is prescribed by the features it possesses. Abstract platform components are called in this framework as *cooperation facilities*, since they provide elements that are needed for realizing interaction and communication in service ecosystems. Denotation of a cooperation facility comprises a selection of features.

A diagram illustrating the relationships between entity kinds and features, as well as the top-level categorization to functional and legal entity kinds is given in Figure 2. The notions of *Concept* and *Intension* are foundational parts of the service ecosystem meta-model which enable ontological and linguistic meta-modelling practices (Ruokolainen, 2009).

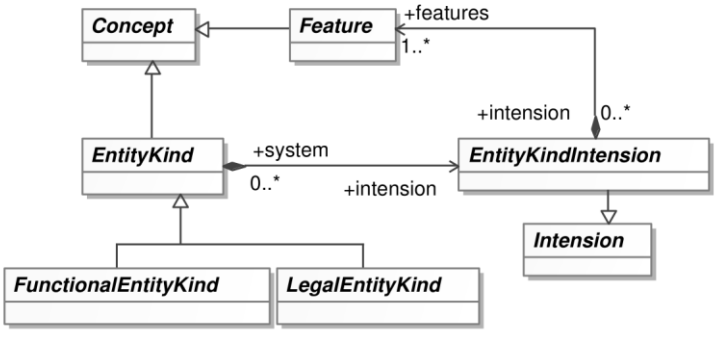

Figure 2: Ecosystem entity kinds.

Functional entities provide the essential activities, behaviour, and interactions for realizing collaboration in service ecosystems. Functional entities comprise endpoint, information, behavioural and service entity kinds, as illustrated in Figure 3. The intensions of functional entities are prescribed by *functional features*.

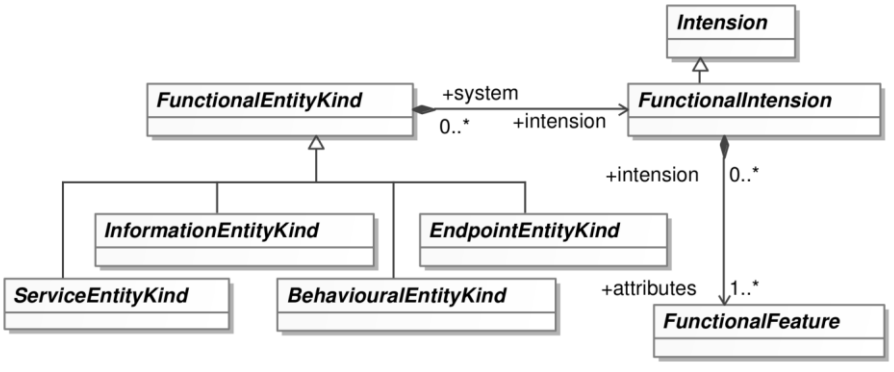

Figure 3: Functional entity kinds.

Endpoint entities represent interaction endpoints in the system. Their features declare what kind of interaction semantics is to be used, e.g. remote procedure calls or publish-subscribe. Information entities represent information contents available in the system with the corresponding features describing business document structures, for example. Behavioural entities' intensions compose behavioural patterns that are supported for realizing collaborations in the ecosystem. We may have behavioural entities that define simple message exchange patterns (MEPs) of the web services architecture (Web Services Architecture Working Group, 2004) or behavioural entities that declare more complex business protocols, as in the case of the Pilarcos service ecosystem (Kutvonen, Ruokolainen, et al., 2008, Ruokolainen & Kutvonen, 2007b). Finally, service entities represent the actual services available in the system. Service entities are further classified to two distinct categories: business services and infrastructure services. Business services are used in the business networks for realizing collaboration activities. Infrastructure services, such as business service discovery services or populators (Kutvonen, Ruokolainen, et al., 2008), are used for realizing service ecosystem life-cycle activities.

Service providers, consumers, clientele and other real-life actors are represented in service ecosystems by the concept of legal entity. Legal entities are categorized to individuals and organizations, as illustrated in Figure 4. The intension of a legal entity kind is defined by a collection of *cooperative features*. A cooperative feature can prescribe policies a legal entity must conform to, or declare a reputation mechanism for evaluating the trustworthiness of a legal entity.

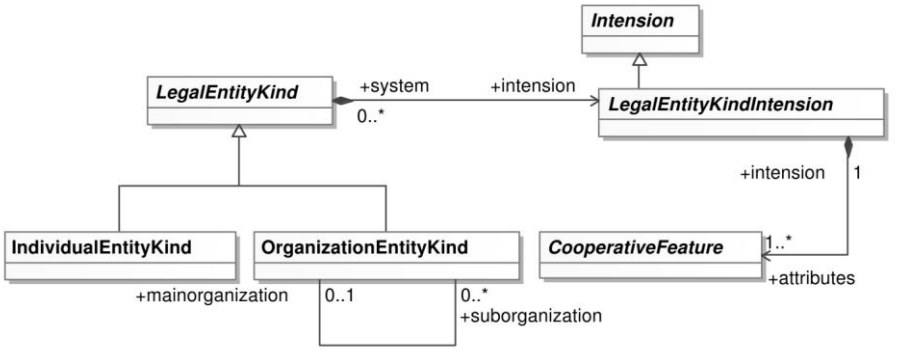

Figure 4: Legal entity kinds.

Cooperation facilities provide elements for describing the abstract platform of a service ecosystem. These abstractions provide 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 5. Intensions of cooperation facilities are defined by a set of *facility features*.

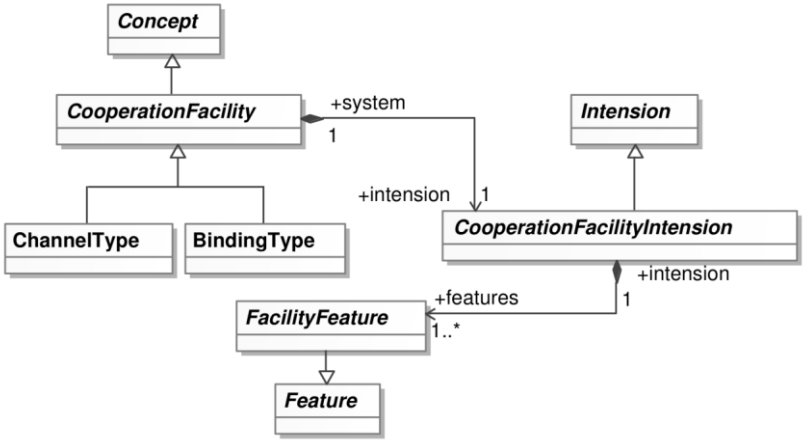

Figure 5: Cooperation facilities.

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 and their features. A channel type comprises an ordered set of channel phases. Each phase represents an individual activity that must be taken for propagating the communication payload from one interaction endpoint to another.

Categorization of service ecosystem features

To facilitate interoperability management in service ecosystems it becomes essential to unambiguously specify ecosystem features. As was discussed above, a specification of ecosystem features must declare both descriptive (ontological) and prescriptive (engineering) characteristics of the features. In the following, we introduce a descriptive categorization of service ecosystem features. The categorization is based on definition of ecosystem entities and cooperation facilities. Especially, most of the feature categories are declared for providing intensions for the entities and cooperation facilities. In addition, qualitative features affecting the functionality of entities and cooperation facilities are provided with appropriate categories.

The categorization of service ecosystem features identifies five different categories, namely *1)* functional features, *2)* facility features, *3)* cooperative features, *4)* contractual features, and *5)* extra-functional features. The categorization is illustrated in Figure 6. Functional features declare intensions of functional entity kinds, while cooperative features are associated with legal entity kinds. Facility features provide meaning for the cooperation facilities. Cooperative, contractual and extra-functional features are qualitative features of legal entity kinds, business services and operations, and cooperation facilities, correspondingly. In the following we discuss the non-functional part of this categorization; features associated with semantics of functional entities such as business services, information or service endpoints are not discussed further in this Chapter.

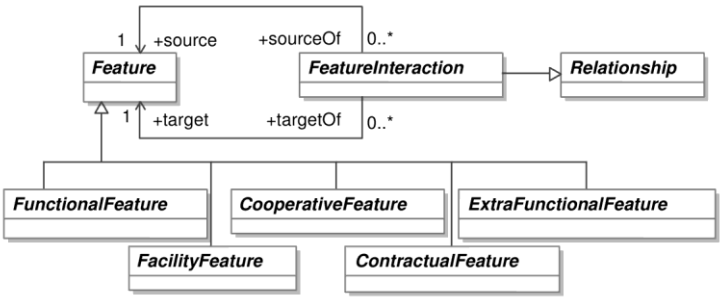

Figure 6: Service ecosystem feature categories.

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 phases, as illustrated in Figure 7. 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 phases*. The ordering is provided by the *predecessor*-association inherited from the concept of *Event*. Each channel phase declared in a channel type is associated with a binding port type defined in a binding type. This effectively makes the set of channel phases a bipartite collection, each phase now belonging to a set associated with one of the two binding port types defined in a binding type.

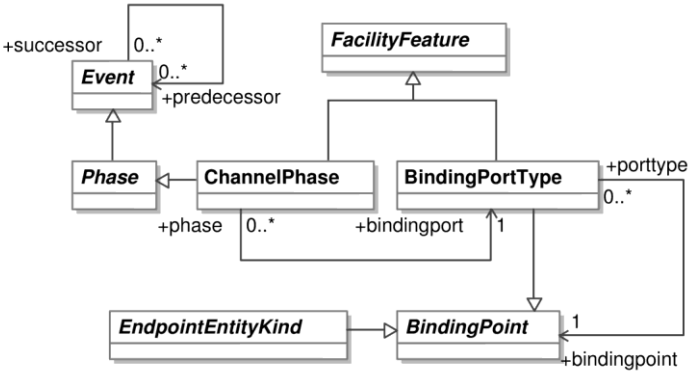

Figure 7: Facility features.

Cooperative features represent a category of service ecosystem features that define the intensions of legal entities. Legal entities are characterized by the rules they must conform to, and means for judging their trustworthiness in a service ecosystem community. As illustrated in Figure 8, the characteristics are represented by the concepts of *Policy* and *ReputationKind*. 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 (Kutvonen, Metso, & Ruohomaa, 2007).

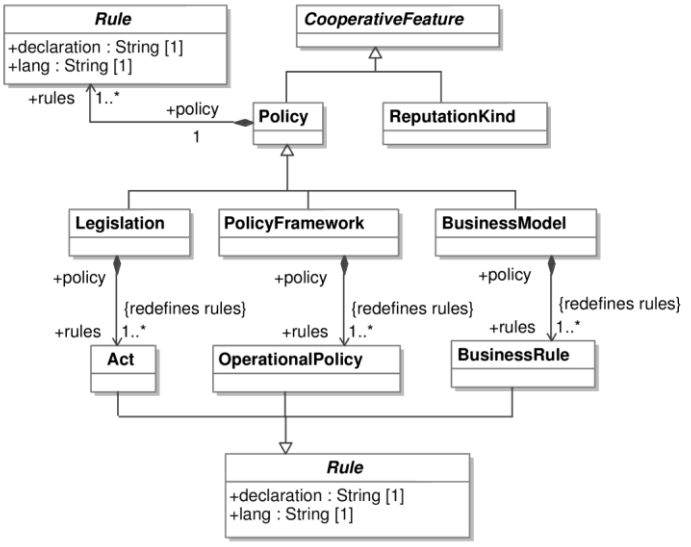

Figure 8: Cooperative features.

Policies are further classified to legislation, policy frameworks and business models. Legislation comprises legal acts that must be obeyed by the corresponding kind of legal entities. 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. Business models are collections of business rules, which are declarative statements defining or constraining some aspect of a business. Different kinds of reputation models or criteria, such as recommendations or ratings, can be categorized under the concept of *ReputationKind*.

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. Collaborative properties especially are subject to business service owners' autonomic intentions. For collaborative 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.

Contractual features represent qualitative characteristics of business services and their operations. Contractual features comprise availability constraints and different charging styles, in addition to different models for settling about the service usage, as illustrated in Figure 9. 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 constraint *"response time must be less than 200 ms"*.

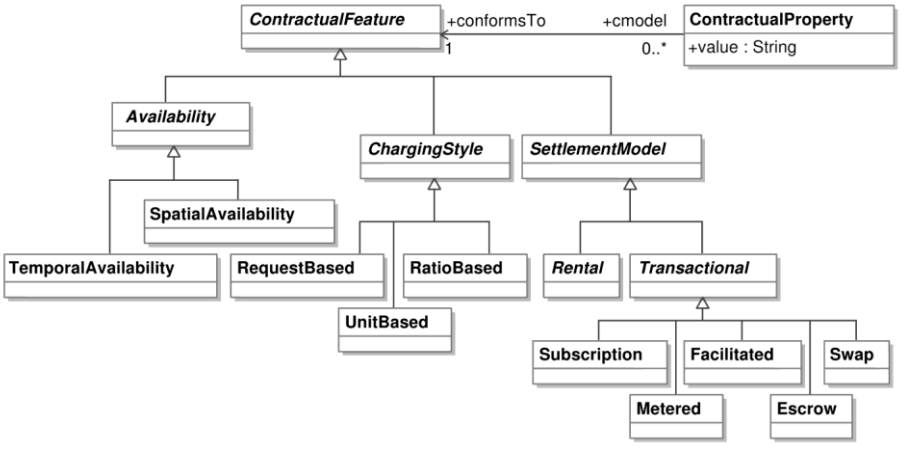

Figure 9: Contractual features.

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. More over, for enabling loosely coupled and dynamic business collaborations, contractual features should be dynamically configurable in the local systems.

Extra-functional features represent qualitative characteristics of cooperation facilities. We identify two categories of extra-functional features: interaction features and communication features, as illustrated in Figure 10. Interaction features are bound to binding types and they represent interaction characteristics, such as functionality related to messaging and encoding. Communication features are bound to channel types and represent functionality such as encryption, decryption or monitoring of behaviour. Communication features must be introduced in certain order to be feasible, that is they can have mutual ordering dependencies: information monitoring must be executed before encryption, for example.

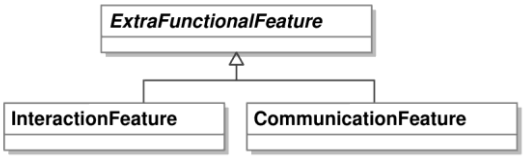

Figure 10: Extra-functional features.

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.

Characteristics of service ecosystem features

The feature categories presented above represent characteristics of distinctive ecosystem elements. From the set of categories we can identify two groups of categories: *1)* intensional features and *2)* qualitative features. Intensional features specify intensions of ecosystem entities and cooperation facilities. That is, the group of intensional features includes functional, facility and cooperative features. Rest of the feature categories, namely contractual and extra-functional features, can be characterized as qualitative features, since they are used for specifying qualitative features of business services and cooperation facilities.

Especially, there is a difference in the usage of intensional features and qualitative features. Intensional features are declared statically over the corresponding subjects, that is entity kinds and cooperation facilities. By contrast, qualitative features are bound dynamically to their targets, such as business services or communication channels. Qualitative features are bound with a mechanism of *property binding*. A property binding is a relationship between a property subject (e.g. a business service) and a property declaration, as illustrated in Figure 11.

A property object can be either a set-based constraint, such as *PropSomeOf* or *PropNoneOf*, a contractual property (applicable over business services or service operations) or an extra-functional feature (applicable over cooperation facilities). The set based constraints give means for declaring different property variations, such as different service pricing policies, for example. During the population phase of the eContracting life-cycle, the properties required by a business network are matched against those declared by service providers. The *PropOneOf* constraint means that any single one of the given properties must be same and supported by a provided services. Constraint *PropSomeOf* means that a number of the given values must be the same but not necessarily all. *PropExactly* means that all properties must be the same. *PropNoneOf* is an exclusive range and means that none of the given values are suitable.

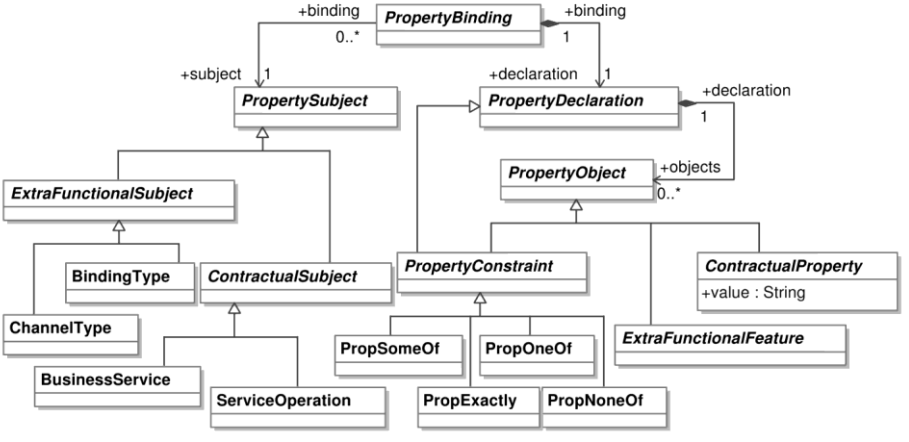

Figure 11: Property binding model.

The categorization also constitutes a family of semantic domains: each feature category is associated with distinctive semantic frameworks. Functional features can be formalized by using a selection of semantic frameworks. For formalizing behavioural features, such as service conversations, formal methods based on Petri-nets (Hamadi & Benatallah, 2003), process algebras (Salan, Bordeaux, & Schaerf, 2004), or finite-state machines (Berardi, Calvanese, Giacomo, Lenzerini, & Mecella, 2003) can be used. Structural features, such as business document typing, can be formalized with appropriate typing schemes addressing XML (Simeon & Wadler, 2003, Hosoya, Vouillon, & Pierce, 2005), for example.

Facility features are used for specifying the semantics of binding types and channel types. In each service ecosystem there are some principles how communication channels can be constructed, for example. These rules can be provided with axiomatic semantics which "*involves rules for deducing assertions about the correctness or equivalence of programs and corresponding parts*" (Zhang & Xu, 2004). Axiomatic semantics is a kind of semantic framework which is used especially for formalizing programming languages. In the context of service ecosystems the cooperation facilities are associated with domain-specific axiomatic semantics. The corresponding rules constrain the construction of channel types and binding types, and provide criteria for their correctness.

Cooperative features are utilized for establishing feasible service provisioning relationships, and for governing the usage and operation of business services. Declarative business rules can be formalized with conceptual graphs (Valatkaite & Vasilecas, 2003) or defeasible logic (Antoniou & Arief, 2002), for example. Operational policies (e.g. privacy preservation) or other normative rules can be formalized, at least to some extent, with different modal logics. Modal logics, such as temporal, deontic or epistemic logics, are utilizable for declaring operational policies over business services, for specifying obligations and permissions over legal entities, and for defining privacy policies, for example (see e.g. (Lupu & Sloman, 1999, Luo, Tan, & Dong, 2009, Benbernou, Meziane, & Hacid, 2007)).

Contractual features are bound to business services and service operations for characterizing their business capabilities. This category includes features such as service availability (e.g. declarations that a service is available during business hours or within a geographical location), charging style (e.g. per business operation or intensity of use), and different models for settling about service use (e.g. rental or subscription). Contractual features are negotiated during eContract establishment; the negotiations are typically bilateral. The properties accepted in negotiations are put in service-level agreements.

The distinguishing characteristics of contractual features is that they are instantiated to concrete values. These values are called *contractual properties*. A contractual feature is considered as a type definition which defines the acceptable value range for the corresponding kinds of properties. An simplified example of contractual feature instantiation is given in Figure 12. The example is illustrated as UML class diagram (Object Management Group, 2005) with instance specifications of the classes presented previously.

In this example, *AvailabilityInCountries* is defined as a kind of a *SpatialAvailability*; this is declared with a *conformsTo* relationship. Contractual features define especially the acceptable value ranges for the corresponding properties. In this case, the acceptable values are lists of ISO standardized country codes [iii] (this declaration is provided only as an informal comment in the example). The contractual property named *MyServiceAvailability* declares that a contractual subject, i.e. a business service or operation, is available in Finland, United Kingdom, Japan and United States. Finally, the contractual subject is bound with the *PropertyBinding* concept to a business service with the name of *MyService*.

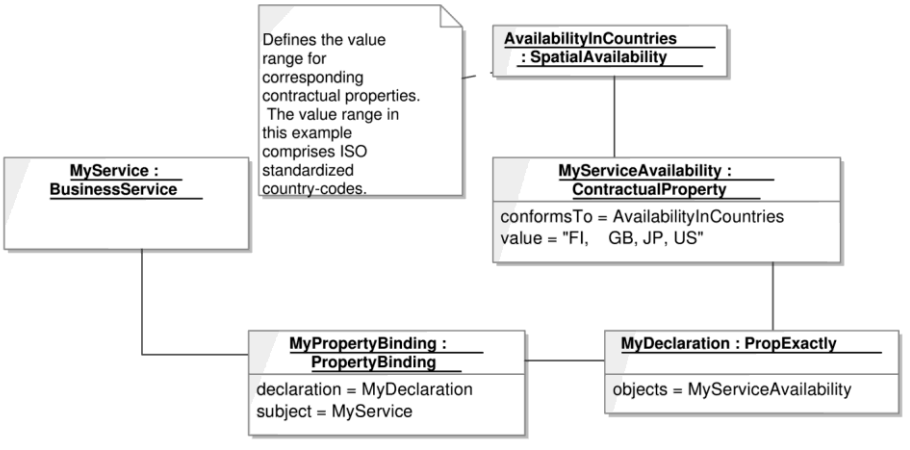

Figure 12: Instantiating a contractual feature.

In addition to contractual features, the extra-functional features are a category of qualitative features that can be bound dynamically. Extra-functional features are bound with the property binding mechanism to cooperation facilities, that is binding types and channel types. In distinction to contractual features that were instantiatable to contractual properties, extra-functional features do not have such a direct typing relationship. Instead, extra-functional features are made concrete by transformations between abstraction levels, e.g. from business level requirements to technology level artifacts.

In this framework, the semantics of extra-functional features are given as model transformations. The model transformations take as an input a cooperation facility and produce a cooperation facility with the required feature implemented by appropriate channel phases, for example. We clarify the characteristics of extra-functional features with a simple example. In this example an extra-functional feature for secure communication is addressed. Within the knowledge base of the service ecosystem exists a declaration for an extra-functional feature named *SecureCommunication*; this is illustrated in Figure 13. More over, a model transformation has been published, named *SCTrans*, which is declared as a *representation of* (Favre, 2004) the *SecureCommunication* feature.

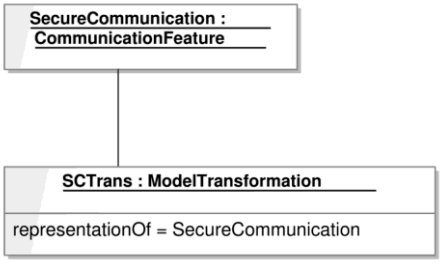

Figure 13: Example of representing an extra-functional feature with model transformation.

In this example we assume that the intensions of cooperation facilities are modeled using a meta-model described in Figure 14. The meta-model is a simplified and streamlined version of the meta-models defined in (Ruokolainen, 2009). The meta-model is an Ecore meta-model of the Eclipse Modeling Framework (The Eclipse Foundation, 2010a) declared in XText-based (The Eclipse Foundation, 2010b) concrete textual syntax. The metamodel defines seven classes with appropriate properties for describing cooperation facilities.

```
import ecore : 'http://www.eclipse.org/emf/2002/Ecore#/';

package facilities : facilities = 'http://cinco.org/cooperationfacilities'
{
  class NamedElement
  {
    attribute name : String[1];
  }
  class ChannelType extends NamedElement
  {
    property features : ChannelPhase[+] { composes };
  }
  class ChannelPhase extends NamedElement
  {
    property predecessor : ChannelPhase[*];
    property bindingport : BindingPortType[1];
  }
  class BindingType extends NamedElement
  {
    property features : BindingPortType[+] { composes };
  }
  class BindingPortType extends NamedElement, BindingPoint
  {
    property bindingpoint : BindingPoint[1];
  }
  class BindingPoint;
  class EndpointEntityKind extends NamedElement, BindingPoint;
}
```

Figure 14: A simplified Eclipse Ecore meta-model prescribing intensions of cooperation facilities.

The *SCTrans* model transformation can be defined using the QVT model transformation language (*Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*, 2005), for example. Such description of the model transformation is given in Figure 15. The model transformation effectively adds encryption and decryption phases to the channel phase sequences contained in any channel type conforming to the meta-model defined in Figure 14. Encryption phases are introduced before every initial phase of channel sequences induced by the *predecessor* reference. Decryption phases are introduced after each final phase of channel sequences.

```
modeltype FACIL "strict" uses facilities('http://cinco.org/cooperationfacilities');

transformation SCTrans(in input : FACIL, out output : FACIL);

main() {
  input.rootObjects()[FACIL::ChannelType]->map channel2SecureChannel();
}

-- Returns the channel phases that do not have any predecessors.
query FACIL::ChannelType::firstPhase() : OrderedSet(FACIL::ChannelPhase) {
  return self.features->select(e | e.predecessor->isEmpty());
}

-- Returns the channel phases that do not have any successors.
query FACIL::ChannelType::lastPhase() : OrderedSet(FACIL::ChannelPhase) {
  return self.features->select(e | -- Select channel phase 'e' from the result
    not self.features->exists(p |  -- if there is no 'p' such that
      p.predecessor->includes(e) -- 'e' is a predecessor of 'p'
    )
  );
}

-- Maps a ChannelPhase to a ChannelPhase (simple copy)
mapping FACIL::ChannelPhase::mapChannelPhase() : FACIL::ChannelPhase {
  name := self.name;
  bindingport := self.bindingport;
  predecessor := self.predecessor;
}

-- Maps a ChannelType to another ChannelType with encryption and decryption
-- phases in phase sequence beginnings and ends, correspondigly.
mapping FACIL::ChannelType::channel2SecureChannel() : FACIL::ChannelType {
  init { var feats := self.features; }

  name := self.name.concat('WithSecurity');

 self.features->forEach(a | true) {
    -- First copy the channel phase to the new channel type.
    features += a->map mapChannelPhase();

    if(self.firstPhase()->includes(a)) then {
      -- This is first phase in sequence
      var enc := object FACIL::ChannelPhase { -- Create the encryption phase.
        name := a.name.concat('Encryption');
        bindingport := a.bindingport;
      };
      -- Set encryption phase a a predecessor of the first channel phase.
      a.resolveoneIn(FACIL::ChannelPhase::mapChannelPhase, ChannelPhase).
        predecessor := enc;
      features += enc;
    } else {
      if(self.lastPhase()->includes(a)) then {
        -- This is last phase in sequence
        var dec := object FACIL::ChannelPhase { -- Create the decryption phase
          name := a.name.concat('Decryption');
          bindingport := a.bindingport;
          -- Set the last phase as the predecessor of the decryption phase
          predecessor := a;
        };
        features += dec;
      } endif;
    } endif;
  }
}
```

Figure 15: SCTrans model transformation defined in QVT language.

Extra-functional features may induce a series of model transformations, or transformation chains. In this setting, the application order of the transformation is essential, since the corresponding features can have mutual dependencies that have to be respected, or there are several abstraction levels in use.

4. MANAGING FEATURES IN SERVICE ECOSYSTEM PROCESSES

Service ecosystems involve several processes where feature management activities take place. The ecosystem processes include those of service-oriented software engineering processes, ecosystem evolution, service ecosystem life-cycles. Service-oriented software engineering processes utilize

domain-specific methodologies suitable for producing service artifacts. The artifacts include service implementation components and models defining different features of services, service collaborations and cooperation facilities. By ecosystem evolution we mean the "meta-life-cycle" of service ecosystems from their design and initiation to operation, and their progressive development, especially with respect to available features, during their operation. Finally, ecosystem life-cycles are processes which prescribe especially processes for collaboration establishment. Service delivery or product life-cycles, among others, could be prescribed in as service ecosystem life-cycles depending on the domain and objectives of the corresponding ecosystem.

Feature management activities in the preceding processes can be characterized as comprising of *a)* feature identification and selection, *b)* feature concretization, *c)* feature introduction, and *d)* feature coordination. These activities are enacted in different phases of the ecosystem processes and have their distinguishing interpretations. Manifestations of feature management activities in service-oriented software engineering, eContracting and ecosystem evolution processes are illustrated in Table 1.

	SOSE	**eContracting**	**Evolution**
IDENTIFICATION	Requirements engineering	Population	Domain analysis
CONCRETIZATION	Feature specification	Negotiation	Ecosystem modeling
INTRODUCTION	Feature implementation	Binding	Feature publication
COORDINATION	Deployment & configuration	Monitoring	Knowledge management

Table 1: Feature management activities in service ecosystem processes.

The actors and the visibility of produced artifacts are different in each of the processes illustrated in Table 1. In service-oriented software engineering processes ecosystem members act typically as individuals for producing local, private artifacts such as implementation components. In service ecosystem life-cycles, such as the eContracting process, a collection of ecosystem members constitute a community which shares knowledge about the characteristics, i.e. features, of the collaboration. Finally, in ecosystem evolution the members of the ecosystem introduce new, public and globally available knowledge into the service ecosystem; this knowledge includes especially features and their categories.

In service-oriented software engineering processes feature identification is provided by requirements engineering activities. Identified features are made concrete by feature specifications which define the descriptive (i.e. ontological) and prescriptive (i.e. engineering) characteristics of the features. The set of identified features can then be formalized with a service ecosystem modeling language (Ruokolainen, 2009). New ecosystem features are introduced locally by implementing them in platform specific technologies. Finally, ecosystem features are coordinated by deployment and configuration activities which weave feature implementations with provided business services, communication components, or other feature implementation components.

In ecosystem evolution the fundamental features and their categories are identified by a domain analysis. The domain analysis is executed during the initial design of the service ecosystem. Domain analysis is *"process by which information used in developing software systems is identified, captured and organized with the purpose of making it reusable when creating new systems"* (Prieto-Díaz, 1990). When this definition of domain analysis is put into the context of service ecosystems, "software systems" are

considered as service collaborations, and "creation of new systems" means establishment of new service collaborations.

During the design of a new service ecosystem the foundational features, the abstract platform, and their inter-dependencies are identified during a domain analysis process. The results of the domain analysis are used for modeling the features of the service ecosystem. Feature concretization is implemented thus during ecosystem modeling. During the operation of service ecosystem new features can be introduced by ecosystem members by publishing feature models. Infrastructure services providing knowledge management functionality are used for such model publication. Ecosystem specific knowledge base, which includes especially the feature models, is coordinated by knowledge management activities enacted by infrastructure services. These activities maintain the knowledge base consistency needs for enabling establishment of interoperable service collaborations.

In the following, we describe more thoroughly the role of feature management activities in collaboration establishment life-cycles, taking the eContracting process of the Pilarcos service ecosystem (Kutvonen, Ruokolainen, et al., 2008, Kutvonen, Metso, & Ruohomaa, 2007) as an example.

eContracting

Service ecosystems are provided with a collaboration establishment life-cycle. A collaboration establishment life-cycle defines a process for preparing necessary agreements and facilities required for service-based cooperation between community members. In the context of the Pilarcos framework (Kutvonen, Ruokolainen, et al., 2008, Kutvonen, Metso, & Ruohomaa, 2007) this process is known as *eContracting*. During eContracting processes features are managed during population, negotiation, configuration, operation and dissolution phases, as illustrated in Figure 16. Phase specific activities, such as service discovery in the population phase or monitoring in the operation phase, are taken for managing business network and service features; the activities are enacted in cooperation by legal entities and infrastructure services. Each phase is also associated with a collection of business services which is refined or utilized in the corresponding eContracting phase.

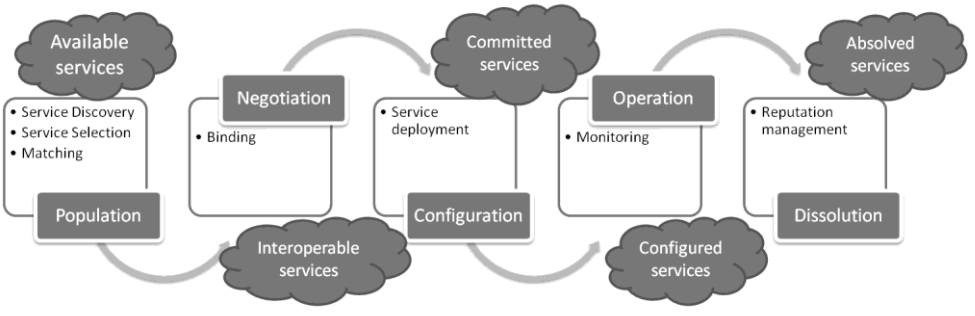

Figure 16: Phases in an eContracting life-cycle.

An eContracting life-cycle starts with a population phase where a business network model is filled with services matching the criteria of the selected business network and those set by the initiator of the population phase (Kutvonen, Metso, & Ruohomaa, 2007). Population phase utilizes infrastructure services available in a service ecosystem for realizing necessary activities; the population activities themselves are enacted by a infrastructure services known as a populator (Kutvonen, Ruokolainen, et al., 2008). Service discovery mechanisms provided by the infrastructure services are first used for identifying services that can

be potentially accepted for a specific business network. The primary criteria for service discovery is the functional features associated with services, e.g. behaviour and structure.

Service discovery activity provides a set of services that are technically compatible with the corresponding business network model. The primary purpose of the service selection activity is to guarantee technical and semantic interoperability. Each service passing the service selection criteria should be at least technologically and behaviourally compatible with the given form of collaboration. In addition to interoperability criteria, both collaboration itself and its initiator may require certain level of initial trust and reputation from corresponding service providers. Infrastructure service providing trust and reputation management mechanisms (Kutvonen, Metso, & Ruohomaa, 2007), are utilized for this purpose.

In the population phase a set of collaboration proposals are established from a selection of services and and a business network model that characterizes the structure and requirements of the collaboration (Kutvonen, Ruokolainen, & Metso, 2007a, Kutvonen, Metso, & Ruohomaa, 2007). Semantic interoperability is addressed further by the population phase especially with respect to the non-functional features of the services and requirements set by the collaboration. Compatibility between different features are matched; for this purpose, constraint satisfaction algorithms can be used (Kutvonen, Metso, & Ruohomaa, 2007). As a final outcome of the population phase, a set of collaboration contract proposals is provided. The services included in the proposals are guaranteed to be interoperable with each other.

While the population phase addresses technical and semantic concerns of interoperability, negotiation phase is utilized especially for addressing the pragmatic interoperability aspects. As an example of pragmatic interoperability aspects, expression of the entities' willingness to collaborate, are considered during the negotiation phase. Cooperative features are utilized especially for such decision making. First of all, the policies associated with the kinds of legal entities are used as the principal criteria for selecting members for business networks to be established. Secondly, reputation of legal entities is used for further judging the eligibility of an entity as a member in the business network.

The negotiation phase enables autonomic ecosystem members to resolve and bargain about the contractual features of the collaboration. The negotiations result in formulation of a collaboration contract which states the responsibilities for each participating entity, the structure of the collaboration, and features expected from the corresponding cooperation facilities, such as communication channels. The collaboration contract is then used for managing the operation of the collaboration (Kutvonen, Ruohomaa, & Metso, 2008, Metso & Kutvonen, 2005).

In service binding the features agreed upon during the preceding negotiations are introduced as more concrete, usually technology specific, declarations. Especially, cooperation facilities are refined with the required extra-functional features; more over, features in higher abstraction levels are instantiated to lower abstraction levels using model transformations, for example.

After a successful binding service providers are equipped with declarations that can be used locally for configuring the technological platforms. Models of the cooperation facilities declared during service binding can be utilized for configuring systems in local administration domains. Models of extra-functional features can be used for generating appropriate implementation components, such as communication interceptors or adapters. More over, models representing cooperative features can be utilized for feeding the local business rule engines with appropriate rules.

During the operation phase the use of features is coordinated with monitoring mechanisms. Especially, contractual properties are used for service-level monitoring of both external (e.g. detecting contract breaches) and internal services. Finally, in the dissolution phase especially the reputation features of legal entities are coordinated. The reputation of entities are updated in accordance to the corresponding kind of reputation system.

5. DISCUSSING THE FRAMEWORK

Due to the dynamism of the environment and autonomy of entities special emphasis must be imposed on controlling and maintaining interoperability knowledge in open service ecosystems. During collaboration establishment processes information is required especially about the features of ecosystem members,

provided services, and available cooperation facilities. In the previous Sections we have described a framework for enabling management of service ecosystem features. In this Section, we discuss impacts of this work on the management of non-functional features in open service ecosystems. After that we introduce a selection of related work with comparison to our framework and a brief analysis on future research directions in the area of model-driven management of service ecosystem features.

Impacts on the management of service ecosystem features

The framework presented in this Chapter provides a well-defined classification of service ecosystem features. We have shown that the corresponding feature categories have their specific roles, as part of different ecosystem elements and in different phases of service ecosystem processes. We have intentionally avoided the use of term "non-functional feature". First of all, the meaning of a non-functional feature or property is ambigous. It's definition as "any other feature than functional" does not get us too far in their management. In this framework, we have first analyzed the components that act in service ecosystems and then defined the features in accordance to the categorization of entities and cooperation facilities (Ruokolainen, 2009). Secondly, features in service ecosystem are all functional in a sense that they are used for decision making, negotiation, or supporting service interactions in the different phases of service ecosystem processes.

Based on the categorization of the foundational entities, a feature categorization has been defined. The characteristics of the corresponding categories are summarized in Table 2. In this characterization two groups of features are distinguished. The three feature categories on the top of the table (functional, facility and cooperative features) can be considered as intensional feature categories, since they are used for specifying the intensions of service ecosystem entities and cooperation facilities. Two remaining categories, contractual and extra-functional, specify qualities of provided business services and the abstract platform. Categories are first characterized with respect to the target of the feature definitions in the corresponding category. Secondly, the kind of semantics utilizable for formalizing the features is given. Finally, some characteristic examples of concrete feature definitions are provided for each of the categories.

	TARGET	SEMANTICS	EXAMPLES
FUNCTIONAL FEATURES	Functional entity intension	Various frameworks (e.g. for operational or structural features)	Service behaviour; business document structures
FACILITY FEATURES	Cooperation facility intension	Axiomatic	One-to-one interaction; communication monitoring
COOPERATIVE FEATURES	Legal entity intension	Various logics (e.g. temporal, deontic, epistemic logics)	Corporate form definitions; domain specific business rules; information privacy laws
CONTRACTUAL FEATURES	Business service and operation qualities	Denotational	Availability of business services; price per operation call
EXTRA-FUNCTIONAL FEATURES	Interaction and communication qualities	Translational (transformations over cooperation facilities)	WS-* / REST –style messaging; communication security

Table 2: Overview of the service ecosystem features.

The framework discussed in this chapter is based on a conceptualization of service ecosystem elements and formalization of the corresponding concepts with a formal meta-model (Ruokolainen, 2009). The meta-model can be considered as a *domain specific meta-modeling language* (Zschaler, Kolovos, Drivalos, Paige, & Rashid, 2010) for service ecosystems. The corresponding meta-modeling language is used for defining the fundamental elements of a service ecosystem prescribing life-cycles, entities and features. Additional domain specific concepts are also included in the resulting service ecosystem models. A service ecosystem model is then utilized for generating service ecosystem specific engineering artifacts. The set of artifacts includes meta-models for describing concept intentions; one such meta-model describing cooperation facilities was illustrated in context of the example given in Section 4. The set of meta-models generated from a service ecosystem model actually constitutes a *family of domain specific languages (DSLs)*. In addition to DSLs, skeletons for ecosystem specific infrastructure services can be generated from the ecosystem model; these include especially model repositories for maintaining information about entities, cooperation facilities and their features.

This work 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 from service ecosystems. Thus, a unifying framework for defining vocabularies enabling engineering knowledge exchange about service artifacts 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.

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 framework discussed in this Chapter 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 information system providers in requirements gathering and design processes in cooperation with their clients. After an appropriate service ecosystem model has been designed, the resulting model is utilizable for producing ecosystem specific meta-models, corresponding DSLs and model repositories. Model-driven engineering principles are exploited for efficient generation of these artifacts.

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 (Bézivin et al., 2005) and further utilized for efficient implementation of model integration (Jossic et al., 2007).

Finally, the framework presented in this Chapter 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 entities 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 (Romero, Jaén, & Vallecillo, 2009).

Research issues in model-driven management of service ecosystem features

The framework presented in this paper utilizes model-driven engineering principles for modeling and managing features in open services ecosystems. Similar approaches have been introduced before for example in (Jonkers et al., 2005). The authors introduce a method for integrating functional models with non-functional ones in the context of model-based service development processes. In their work, the authors make a distinction between two modeling spaces for non-functional features, namely design and analysis space. Design space comprises modeling languages and tools for describing non-functional features. Analysis space consists of specification languages and notations which are applicable for formalizing the semantics of the non-functional features of interest. Horizontal transformations are then used for propating information between a design space and a corresponding analysis phase. Vertical model transformations are used for model refinement within the modelling spaces in the traditional model-driven engineering sense. Similarly, (Köllmann et al., 2007) presents an approach for managing several Quality of Service (QoS) dependability dimensions. This approach applies model-driven development and aspect-oriented techniques for detaching the QoS aspects from software specifications. Graph transformations are then utilized for weaving the QoS aspects to QoS independent models. The approach of (Jonkers et al., 2005) for attaching domain-specific semantics for non-functional features can be utilized in the framework presented in this Chapter. Also, the approach of (Köllmann et al., 2007) for providing translational semantics for non-functional features can be used. Our approach is more specific in a sense that it is targeted for open service ecosystems. Especially, our approach formalizes the inter-dependencies and roles between different "non-functional" and functional features in service ecosystems. We have, to a certain degree, fixed the semantic for different features in service ecosystems. We see that such constraints over the feature categories, their definitions and usage are needed for enabling feature management in various service ecosystems.

In (Ameller, Cabot & Franch, 2010) the authors report current state of model-driven engineering approaches for managing non-functional requirements in software engineering processes. They make a remark based on a literature survey that in general non-functional requirements are not addressed in model-driven engineering methods. Towards enhancing the situation they envision a general framework that integrates non-functional requirements management into model-driven software engineering process, and identify research issues related to their framework. In their framework proposal the authors utilize a platform-independent model (PIM) for representing the functionality and non-functional requirements of a system. This PIM is then analysed against a knowledge base containing information about available non-functional, architectural and technological features and solutions. Based on this analysis a model transformation is created which transforms the PIM to an architectural model. The architectural model describes an architecture that implements all the functionality of the system in a way that satisfies the non-functional requirements whose satisfaction depends on the decisions made at the architectural level (Amellers, Cabot & Franch, 2010). The architectural model is then analyzed againts the knowledge base and a second model transformation is applied. The model transformations takes the architectural model as an input and produces a platform-specific model (PSM). The PSM follows the architectural guidelines expressed in the architectural model but also takes into account non-functional requirements depending on technological choises. Finally, a model to text transformation can be applied for generating technology specific code from the PSM. The approach proposed by Amellers, Cabot & Franch (Amellers, Cabot &

Franch, 2010) can be aligned with our approach. In our framework the knowledge about service ecosystem features is available in specific knowledge repositories. This knowledge is based on the categorization of the features and the corresponding metamodels presented in this Chapter. The metamodels introduced in this Chapter are part of a larger metamodel which formalizes the foundational elements of service ecosystems (Ruokolainen, 2009). This service ecosystem metamodel can be considered as a model for architectural models in the sense of (Amellers, Cabot & Franch, 2010). It is used as a basis for defining platform-independent models that specify all requirements of service-based collaborations and their elements. In addition to architectural issues, the service ecosystem metamodel includes technology-oriented knowledge in form of cooperation facility features and extra-functional features, as discussed in the previous sections.

More generally the framework behind the work presented in this Chapter is related to research conducted in the areas of large-scale SOA systems, their modeling and corresponding service-based middleware platforms. There are a few European research initiatives and projects that have similar goals with this respect. NESSI (*Networked European Software & Services Initiative)* is a European Technology Platform dedicated to software and services (Lizcano et al., 2010). 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" (NESSI Consortium, 2009). In comparison to the reference architecture developed as part of the NEXOF, our service ecosystem framework is more focused on the knowledge management side of service ecosystems, and explicitly provides means for extending domain models of specific ecosystems with new concepts.

The SeCSE (*Service Centric System Engineering*) is an EU Integrated Project of the 6th Framework Program that aims for developing processes, methods and tools to develop service-oriented systems (Colombo et al., 2005). The SeCSE project provides a conceptual model for service oriented systems describing actors, entities and activities relevant to the service domain, and relationships between them. While the conceptual model of SeCSE 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 (Colombo et al., 2005). The primary purpose of our framework and the corresponding conceptual model is to facilitate the infrastructure services and tools needed for instrumenting service ecosystems.

The framework presented in this Chapter includes several topics for further research; a selection of these are discussed below. The framework implicitly proposes an approach for modeling service-oriented systems with a family of feature-specific languages. Each of the are used for defining different aspects of the system, and the overall model defining the service ecosystem is utilized for guaranteeing coherency of the language family and consistency between different languages. Utilizing such a language family for operating in a service ecosystem necessitates appropriate modeling tools and methodologies. From modeling tools perspective, several notations have been developed for describing different kinds of features in software systems (e.g. Object Management Group, 2005; Amellers, Cabot & Franch, 2010). However, simply providing a notation for feature modeling is not sufficient. What is still lacking from most of the modeling tools is the capability analyze feature interactions and effects of introducing cross-cutting concerns in system models. Towards this purpose, research should be conducted especially in the areas of analysing viewpoint correspondences and consistency based on semantics frameworks defined for corresponding domain-specific languages. Feature interactions are likely to introduce interactions between the different semantics frameworks (e.g. between operational semantics of business processes and declarative semantics of business rules); this is a research topic that should be investigated more in the future. From the methodological viewpoint, engineering processes and methols should be developed that are applicable for distributed development taking place in open service ecosystems and that utilize multi-viewpoint modeling practices. Knowledge sharing facilities provided by shared, global knowledge repositories should be also integrated to corresponding software engineering tools.

Modeling of service ecosystem features is only the first phase in their application. Especially in open service ecosystems models are utilized for configuring, adapting and governing the operation of the system. Such a models-at-runtime –approach (e.g. Blair, Bencomo, & France, 2009) involves in itself several new

research challenges, the most foundational one being that of maintaining the causal relationship between the model and the running system. Maintaining the causal relationship becomes problematic especially in open service ecosystems, where individual services are maintained by autonomous service providers and access to the underlying technological systems are restricted due to security related and competitive reasons. In the Pilarcos framework (Kutvonen et al., 2008) we have especially covered issues related to maintaining a coherent view over the service collaborations between autonomous partners.

6. CONCLUSION

Open service ecosystems present means to solve many collaboration management and interoperability control problems. Indeed, the ecosystem concept reveals that there is notably different interest domains within the feature concept family: business control needs, service control needs, and configuration needs on communication channels between services. Although the presently arising software and service ecosystems forward the business domain significantly, there are still severe problems to be solved:

- trustworthiness of service offers,
- interoperability control automation, and
- collaborative, systematic methods for dynamic collaboration management.

These issues cannot be resolved unless the open service ecosystems are able to bind together

- the service-oriented software engineering methodologies that are responsible of providing business network models serving as eContract templates and thus providing evaluated rules for detecting illegal, unwanted, or low quality services;
- operational time collaboration management, including service selection advised with trustworthiness predictions, eContract forming, monitoring of contract breaches, and feedback on the experiences gained.

This Chapter has shown how a consistent knowledge base for maintaining features in open service ecosystems can be provided, thus creating a life link between the engineering and operational environments. Furthermore, the knowledge base structure must allow for declaration of new concepts and new relationships between concepts, thus facilitating further evolution of the ecosystem without disturbance in the already existing collaborations. The framework allows multiple different kind of ecosystems to be established, and controlled either as isolated, or federated, for cases of competing or collaborating ecosystems.

REFERENCES

Almeida, J. P., Dijkman, R., Sinderen, M. v., & Pires, L. F. (2004). On the Notion of Abstract Platform in MDA Development. In *EDOC '04: Proceedings of the Eighth IEEE International Enterprise Distributed Object Computing conference* (pp. 253–263). Washington, DC, USA: IEEE Computer Society.

Ameller, D., Cabot, J. & Franch, X. (2010). Dealing with Non-Functional Requirements in Model-Driven Development. In *Requirements Engineering Conference (RE), 2010 18th IEEE International* (pp. 189-198). Washington, DC, USA: IEEE Computer Society.

Antoniou, G., & Arief, M. (2002). Executable declarative business rules and their use in electronic commerce. In *SAC '02: ACM Symposium on Applied Computing* (pp. 6–10). New York, NY, USA: ACM Press.

Benbernou, S., Meziane, H., & Hacid, M. S. (2007). Run-Time Monitoring for Privacy-Agreement Compliance. In *ICSOC '07: Proceedings of the 5th International Conference on Service-Oriented Computing* (pp. 353–364). Berlin / Heidelberg, Germany: Springer-Verlag.

Berardi, D., Calvanese, D., Giacomo, G. D., Lenzerini, M., & Mecella, M. (2003). Automatic Composition of E-services That Export Their Behavior. In *ICSOC '03: Proceedings of the First International Conference on Service-Oriented Computing* (Vol. 2910, pp. 43–58). Berlin / Heidelbert, Germany: Springer.

Bézivin, J., Jouault, F., Rosenthal, P. & Valduriez, P. (2005). Modeling in the Large and Modeling in the Small. In *Model Driven Architecture.* (Vol. 3599, pp. 33-46). Berlin / Heidelberg, Germany: Springer.

Blair, G., Bencomo, N., France, R.B. (October, 2009). Models@run.time. *Computer* (vol. 42, no. 10, pp. 22-27). Washington, DC, USA: IEEE Computer Society.

Bosch, J., & Bosch-Sijtsema, P. (2010). From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software*, *83*(1), 67–76.

Colombo, M., Di Nitto, E., Di Penta, M., distante, D., & Zuccala, M. (2005). Speaking a common language: A conceptual model for describing service-oriented systems. In Benatallah, B., Casati F. & Traverso, P. (Ed.), *Service-Oriented Computing – ICSOC 2005,* (Vol. 3820, pp. 48-60). Berlin / Heidelberg, Germany: Springer.

Favre, J.-M. (2004). Foundations of Model (Driven) (Reverse) Engineering : Models - Episode I: Stories of The Fidus Papyrus and of The Solarus. In Bézivin, J. & Heckel, R. (Ed.), *Language engineering for model-driven software development,* (Vol. 04101). Schloss Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI).

Gonzalez-Perez, C., & Henderson-Sellers, B. (2006). A powertype-based metamodelling framework. *Software and Systems Modeling*, (Vol. 5, pp. 72-90).

Hamadi, R., & Benatallah, B. (2003). A Petri net-based model for web service composition. In *CRPITS'17: Proceedings of the Fourteenth Australasian database conference on database technologies 2003* (pp. 191–200). Darlinghurst, Australia: Australian Computer Society, Inc.

Hosoya, H., Vouillon, J., & Pierce, B. C. (2005). Regular expression types for XML. *ACM Transactions on Programming Languages and Systems*, (Vol. 27, no. 1, pp. 46–90).

Jonkers, H, Iacob, M.E., Lankhorts, M.M. &Strating, P. (2005). Integration and Analysis of Functional and Non-Functional Aspects in Model-Driven E-Service Development. In *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference.* Washington, DC, USA: IEEE Computer Society.

Jossic, A., Didonet Del Fabro, M., Lerat, J., Bézivin, J., & Jouault, F. (2007). Model Integration with Model Weaving: a Case Study in System Architecture. In *Proceedings of the 2007 International Conference on Systems Engineering and Modeling (ICSEM'07)* (pp. 79–84). IEEE.

Kabilan, V., Johannesson, P., Ruohomaa, S., Moen, P., Herrmann, A., Åhlfeldt, R.-M. (2007). Introducing the common non-functional ontology. In *Enterprise Interoperability II — New Challenges and Approaches* (pp. 633–646). London, United Kingdom: Springer.

Kassab, M., Ormandjieva, O., & Daneva, M. (2009). An ontology based approach to non-functional requirements conceptualization. In *International Conference on Software Engineering Advances (ICSEA)* (pp. 299–308). Washington, DC, USA: IEEE Computer Society.

Kutvonen, L. (2002). Automated management of interorganisational applications. In *Proceedings of the Sixth International Enterprise Distributed Object Computing Conference (EDOC'02).* Washington, DC, USA: IEEE Computer Society.

Kutvonen, L. (2004). Challenges for ODP-based infrastructure for managing dynamic B2B networks. In A. Vallecillo, P. Linington, & B. Wood (Eds.), *Workshop on ODP for Enterprise Computing (WODPEC 2004)* (pp. 57–64).

Kutvonen, L., Metso, J., & Ruohomaa, S. (2007, July). From trading to eCommunity management: Responding to social and contractual challenges. *Information Systems Frontiers (ISF) - Special Issue on Enterprise Services Computing: Evolution and Challenges*, (Vol. 9, no. 2-3, pp. 181-194).

Kutvonen, L., Metso, J., & Ruokolainen, T. (2005, November). Inter-enterprise collaboration management in dynamic business networks. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE*. (Vol. 3760, pp. 593-611). Berlin / Heidelberg, Germany: Springer.

Kutvonen, L., Ruohomaa, S., & Metso, J. (2008, September 8–10). Automating decisions for inter-enterprise collaboration management. In *Pervasive collaborative networks. IFIP TC 5 WG 5.5 Ninth working conference on virtual enterprises, september 8–10, 2008, poznan, poland* (pp. 127–134). Springer.

Kutvonen, L., Ruokolainen, T., & Metso, J. (2007, January). 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), 1–21.

Kutvonen, L., Ruokolainen, T., Ruohomaa, S., & Metso, J. (2008, October). Service-Oriented Middleware for Managing Inter-Enterprise Collaborations. In A. Gunasekaran (Ed.), *Global Implications of Modern Enterprise Information Systems: Technologies and Applications* (pp. 208–241). IGI Global.

Köllmann, C., Kutvonen, L., Linington, P., & Solberg, A. (2007). An Aspect-Oriented Approach to Manage QoS Dependability Dimensions in Model Driven Development. In L. Ferreira Pires and S. Hammoudi (Ed.), *The 3rd International Workshop on Model-Driven Enterprise Information Systems (MDEIS 2007)* (pp. 85-94). INSTICC Press.

Lizcano, D., Jiménez, M., Soriano, J., Cantera, J. M., Reyes, M., Hierro, J. J., Garijo, F., & Tsouroulas, N. (2008). 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* (pp. 147-158). Berlin / Heidelberg, Germany: Springer-Verlag.

Luo, X., Tan, Z., & Dong, R. (2009). Automatic verification of composite web services based on temporal and epistemic logic. In *WGEC '09: Proceedings of the 2009 Third International Conference on Genetic and Evolutionary Computing* (pp. 693–696). Washington, DC, USA: IEEE Computer Society.

Lupu, E. C., & Sloman, M. (1999). Conflicts in Policy-Based Distributed Systems Management. *IEEE Transactions on Software Engineering*, *25*(6), 852–869.

Mehandjiev, N., & Grefen, P. (Eds.). (2010). *Dynamic Business Process Formation for Instant Virtual Enterprises*. Springer.

Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. (2005, 11November). (Final Adopted Specification – ptc/05-11-01)

Metso, J., & Kutvonen, L. (2005, September). Managing Virtual Organizations with Contracts. In *Workshop on Contract Architectures and Languages (CoALa2005).*

Moen, P., Ruohomaa, S., Viljanen, L. & Kutvonen, L. (2010). *Safeguarding against new privacy threats in inter-enterprise collaboration environments*. (Report No. C-2010-56). Helsinki, Finland: University of Helsinki, Department of Computer Science.

NESSI Consortium. *NESSI Open Framework - Reference Architecture – RA Model V2.0* (April, 2009). Retrieved December 17, 2010, from http://www.nexof-ra.eu/sites/default/files/D6.2_v1.0.pdf

Prieto-Díaz, R. (1990). Domain Analysis: an Introduction. *SIGSOFT Software Engineering Notes*, *15*(2), 47–54.

Rabelo, R. J., Gusmeroli, S., Arana, C., & Nagellen, T. (2006). The ECOLEAD ICT Infrastructure for Collaborative Networked Organizations. In *Network-centric collaboration and supporting frameworks* (Vol. 224, pp. 451–460). Springer.

Romero, J. R., Jaén, J. I., & Vallecillo, A. (2009). Realizing Correspondences in Multi-Viewpoint Specifications. In *EDOC '09: IEEE International Enterprise Distributed Object Computing Conference* (pp. 163–172). IEEE.

Ruohomaa, S., & Kutvonen, L. (2008, March). Making multi-dimensional trust decisions on inter-enterprise collaborations. In *Proceedings of the Third International Conference on Availability, Security and Reliability (ARES 2008)* (pp. 873–880). IEEE Computer Society.

Ruohomaa, S., Viljanen, L., & Kutvonen, L. (2006, March). Guarding enterprise collaborations with trust decisions — the TuBE approach. In *Interoperability for enterprise software and applications. Proceedings of the workshops and the doctoral symposium of the second IFAC/IFIP I-ESA International Conference: EI2N, WSI, IS-TSPQ 2006* (pp. 237–248). ISTE Ltd.

Ruokolainen, T. (2009, June). *Modelling framework for interoperability management in collaborative computing environments* (Tech. Rep. No. C-2009-9). Department of Computer Science, University of Helsinki. (Licentiate's thesis)

Ruokolainen, T., & Kutvonen, L. (2006, September). Addressing Autonomy and Interoperability in Breeding Environments. In L. Camarinha-Matos, H. Afsarmanesh, & M. Ollus (Eds.), *Network-Centric Collaboration and Supporting Frameworks* (Vol. 224, pp. 481–488). Helsinki, Finland: Springer.

Ruokolainen, T., & Kutvonen, L. (2007a, September). Managing non-functional properties of inter-enterprise business service delivery. In *Non functional properties and service level agreements in service oriented computing workshop (NFPSLA-SOC) (co-located with the 5th international conference on service oriented computing, ICSOC 2007).*

Ruokolainen, T., & Kutvonen, L. (2007b, April). Service Typing in Collaborative Systems. In G. Doumeingts, J. Müller, G. Morel, & B. Vallespir (Eds.), *Enterprise Interoperability: New Challenges and Approaches* (pp. 343–354). Springer.

Salan, G., Bordeaux, L., & Schaerf, M. (2004). Describing and Reasoning on Web Services using Process Algebra. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services* (p. 43). Washington, DC, USA: IEEE Computer Society.

Simeon, J., & Wadler, P. (2003). The essence of XML. In *POPL '03: Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 1–13). New York, NY, USA: ACM Press.

Object Management Group. (2005, August) *Unified Modeling Language: Superstructure.*

The Eclipse Foundation. (2010a). *Eclipse Modeling Framework website.* Retrieved December 17, 2010, from http://www.eclipse.org/modeling/emf/.

The Eclipse Foundation. (2010b) *Xtext - Language Development Framework.* Retrieved December 17, 2010, from http://www.eclipse.org/Xtext/.

Valatkaite, I., & Vasilecas, O. (2003, September). A Conceptual Graphs Approach for Business Rules Modeling. In *Advances in databases and information systems* (Vol. 2798).

Web Services Architecture Working Group. (2004, February) *Web Services Architecture.* W3C Working Group Note 11 February 2004. http://www.w3.org/TR/ws-arch/

Zhang, Y., & Xu, B. (2004). A survey of semantic description frameworks for programming languages. *SIGPLAN Notifications*, *39*(3), 14–30.

Zschaler, S., Kolovos, D., Drivalos, N., Paige, R., & Rashid, A. (2010). Domain-Specific Metamodelling Languages for Software Language Engineering. In M. van den Brand, D. Gasevic, & J. Gray (Eds.), *SLE'10: Software Language Engineering* (Vol. 5969, p. 334-353). Springer Berlin / Heidelberg.

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

[ii] Nokia Ovi: http://www.ovi.com/

[iii] http://www.iso.org/iso/country_codes/iso_3166_code_lists.htm