Requirements Engineering in Complex Domains

Matthias Jarke[1], Ralf Klamma[1], Klaus Pohl[2], and Ernst Sikora[2]

[1] RWTH Aachen University & Fraunhofer FIT
{jarke,klamma}@informatik.rwth-aachen.de
[2] University of Duisburg-Essen
{klaus.pohl,ernst.sikora}@paluno.uni-due.de

Abstract. Complexity in the application domains of software-intensive systems is continuously growing due to at least two reasons. Firstly, technical complexity grows as hardware and software have to interact in individual or even communicating embedded systems. Secondly, social complexity grows as the process organizations of the 1990's are gradually being replaced by loosely coupled networks of actors, often organized around community platforms. In this chapter, we discuss recent solution attempts for these two issues individually, and end with speculating about their possible future interaction.

Keywords: abstraction layers, goals, scenarios, architecture, social networks, reflective architectures, web communities.

1 Introduction

As an essential bridge between user and developer concepts, requirements engineering (RE) has increasingly captured the attention of researchers, practitioners, and sometimes even executives and politicians for the past thirty years. Already in the early 1990's, RE was defined as the process of "establishing a vision in context", i.e. the critical role of the context in which a system is developed, operated, and evolved, has been recognized [1]. In two European Basic Research Projects as well as in the German SFB IMPROVE [2], the relations between requirements and the context of the system have been elaborated focusing on aspects such as organization of the context knowledge in different "worlds" of engineering or the interplay of goals and contextual scenarios.

In the 21st century, the basic idea of establishing visions in context remains valid, but the context to be considered and the stakeholders have changed significantly. Firstly, "greenfield" development of completely new systems hardly exists any more. In contrast, changing and new requirements must typically be embedded in large-scale corporate and technical system architectures. Secondly, the pressures on cost and the globalization necessitate internationally distributed "development worlds" with increasing interactions and much stronger needs for precise communication. Third, much of the innovations are no longer coming from the well-structured process organizations (like in the 1990's), but from the edges of shifting cross-organizational networks and flexible service-orientation of which the Web 2.0 movement is probably the best-known example.

G. Engels et al. (Eds.): Nagl Festschrift, LNCS 5765, pp. 602–620, 2010.

In this paper, we illustrate these new trends and their consequences for requirements engineering researched in two recent projects, one focusing on the co-development of requirements and architectures in complex software-intensive systems (Section 2), the other focusing on requirements engineering in internet communities where the boundaries between users and developers tend to blur as quickly as the membership of the communities themselves (Section 3). Section 4 concludes with some speculation about future confluence of these challenges.

2 Requirements Engineering for Complex Software-Intensive Systems

As Brooks stated in 1987: „The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later." [3]

Today, in many domains such as automotive, avionics, automation, energy, or medicine, innovative product features are realized increasingly by means of software, more precisely through networks of software-intensive, embedded systems. The number of (software-based) system functions and the number of dependencies between these system functions increase rapidly. Hence the complexity of these systems and their software increases as well. The need to develop software-based, innovative functions makes requirements engineering for software-intensive systems more challenging than ever: The requirements engineering process must satisfy strict quality demands which are often enforced even by laws and standards. At the same time, the schedules for developing new systems or, respectively, new system versions are tight, and the development costs must constantly be reduced. Moreover, the proper integration of requirements engineering into the overall development process, especially the intertwining of requirements engineering and architectural design, is becoming an essential factor for successful system development.

In the light of these challenges we have developed the COSMOD-RE method (sCenario and gOal based System development MethOD; see e.g. [4]). COSMOD-RE supports the co-development of requirements and architectural artifacts for software-intensive, embedded systems at multiple layers of abstraction. In the following, we sketch out the cornerstones of COSMOD-RE. In Section 2.1, we provide a brief overview of the key building-blocks of COSMOD-RE. In Section 2.2, we introduce the four abstraction layers which define the backbone of COSMOD-RE. In Section 2.3, we briefly describe the co-development of requirements and architectural artifacts which we support by means of goals and scenarios. Moreover, we illustrate the use of COSMOD-RE using a simplified ACC (adaptive cruise-control system). Section 2.4 provides a brief outlook on the further development of COSMOD-RE.

2.1 COSMOD-RE: Brief Overview

Fig. 1 depicts an overview of the three main building blocks of COSMOD-RE that are described in the next sections: the use of a hierarchy of four abstraction layers, the

support of the intertwined and partly concurrent development of requirements and architectural artifacts by means of co-design-processes, and the use of goals and scenarios to support the refinement of requirements across the abstraction layers as well as the co-development of requirements and architectural artifacts.

Fig. 1. Main building blocks of the COSMOD-RE method

2.2 COSMOD-RE Abstraction Layers

The use of abstraction layers is a well-proven means for problem-solving and, in particular, for dealing with and managing the complexity of software-intensive systems (see e.g. [5]). As Weber and Weisbrod state: "Engineers cannot develop a complex system – such as an up-to-date telematics unit – solely by talking about and dealing with the system requirements that make up the low-level, detailed component specification. On the contrary, developing complex systems from the top down in several layers of granularity is inevitable. When it comes to RE, this observation still holds. Unfortunately, today systematic processing and documentation of higher level requirements and design decisions are insufficient." [6]

In order to support and guide the specification of requirements at different levels of granularity, COSMOD-RE is based on a generalized (or essential) hierarchy of abstraction layers that can be applied to a wide range of domains and systems (see Fig. 2).

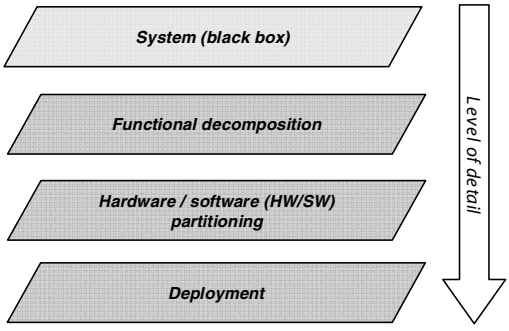

Fig. 2. Hierarchy of four essential abstraction layers defining the backbone of COSMOD-RE

The abstraction layers used in COSMOD-RE are:

- *System layer*: The system layer (highest abstraction layer) defines requirements and architectural artifacts fairly independently of the implementation technology and internal structure (decomposition) of the system. The system is regarded as a black box. This layer thus focuses on the embedding of the system in its environment. The requirements at the system layer are defined mainly from a system usage perspective. At each successive layer, the requirements are refined and additional design concerns are taken into account.
- *Functional decomposition layer*: At the functional decomposition layer, the system is decomposed into coarse-grained, logical building blocks. The requirements for each building block as well as the interrelations and interactions between the building blocks are defined at this layer. The functional decomposition layer aims at supporting problem understanding and problem decomposition.
- *Hardware/software (HW/SW) partitioning layer*: At the hardware/software partitioning layer, the system is decomposed into coarse-grained, technical building blocks, i.e. hardware and software building blocks. The requirements for each building block as well as the interrelations and interactions between the hardware and software building blocks are defined. At this layer, the decision is made which system properties are realised through hardware components and which ones are realised through software components.
- *Deployment layer*: At the deployment layer, the deployment of the system (i.e. the application software and hardware) into a hardware and software platform is defined. Such a platform typically consists of a set of physical units which are connected by physical networks and are equipped with basic system software. Hence, at this layer, the decision is made which application software or hardware component is deployed to which physical unit. This entails refining the requirements with regard to the specifics of each physical unit (such as processing speed and memory, input and output facilities, operating system, communication software, middleware etc. of the respective physical unit).

This hierarchy can be refined or simplified, if needed. The specific hierarchy of abstraction layers used in a project should take into account characteristic properties

of the domain, the organization, or even the project itself (such as system complexity, desired reusability etc.). Using such a hierarchy of abstraction layers offers the following advantages for the development of complex, software-intensive systems (see [7] for more details):

- *Hierarchical (problem) decomposition*: When using abstraction layers, the problem of defining detailed requirements for a software-intensive system is decomposed into a set of smaller problems of defining detailed requirements for individual components and their interactions. Clearly, specifying, for instance, the behavioral requirements for a component (e.g. the requirements for the brake control software) is a simpler task than specifying the behavioral requirements for the entire vehicle at the same level of detail simultaneously.
- *Requirements stability*: Requirements and, partly, architectural solutions at higher abstraction layers are defined independently of their technical realizations e.g. by hardware and software components. Since requirements defined at the system layer are fairly independent of the technical solution, they are typically not affected by changes in the technical realization, such as changes at the hardware/software or the deployment layer. The effects of the abstraction layers are similar to the positive effects of the differentiation between the "essence of a system" and the "incarnation of a system" (technology-independent and technology-dependent requirements) in Essential Systems Analysis [8].
- *Traceability and rationale*: Requirements defined at lower abstraction layers can be related e.g. by means of "refines" relationships to requirements defined at higher abstraction layers. The requirements defined at the lower abstraction layers can be traced back to higher-level requirements. The requirements defined at higher abstraction layers thus provide rationale for the requirements defined at the lower abstraction layers. Among other things, this improves the traceability of the requirements and contributes to their comprehensibility.

When developing requirements at multiple abstraction layers, it becomes obvious that requirements engineering and architectural design are inevitably intertwined (see e.g. [9] [10]) as discussed in the next section.

2.3 Co-development of Requirements and Architectural Artifacts

When developing a complex system, the stakeholders have to accomplish two different but closely related tasks (see e.g. [11]):

- *Refinement of the requirements*: The stakeholders have to refine high-level requirements into detailed requirements which contain enough details to facilitate the implementation and quality assurance (e.g. testing) of the system.
- *Decomposition of the system*: The stakeholders have to decompose the overall system into a set of interacting parts (sub-systems or components), i.e. the stakeholders have to define a detailed architecture which satisfies the defined (detailed) requirements.

Clearly, the requirements influence and partly even determine the architecture. However, design decisions (e.g. the choice of a specific architectural solution for the system) strongly influence the refinement of high-level requirements into detailed

(implementable and testable) requirements. In other words, the refinement of the requirements depends (partly) on the design choices taken. In addition, an innovative architectural solution can even lead to entirely new (high-level) requirements.

As suggested, for example, by Nagl and his co-authors (see e.g. [12]), development methods and tools should support a tight integration between requirements engineering and architectural design. Still, existing development methods do not foster the intertwined development of requirements and architecture. Refining requirements without a systematic exploration of possible design options bears the danger that important design decisions are made implicitly. In other words, stakeholders often hide design decisions taken (explicitly or implicitly) in the detailed requirements. Such implicit design decisions often rule out other (and perhaps better) solutions. In order to avoid such problems (implicit design decisions), the COSMOD-RE method fosters and systematically supports the intertwined development of requirements and architectural artifacts.

2.3.1 A Co-design Process

COSMOD-RE supports the intertwined and partly concurrent development of requirements and architectural artifacts by means of so-called co-design processes. Fig. 3 depicts the structure of the co-design processes defined by COSMOD-RE. A co-design process in COSMOD-RE supports the development of requirements artifacts at an abstraction layer L_i and the alignment of these requirements artifacts with the architectural solution at the abstraction layer L_{i+1}.

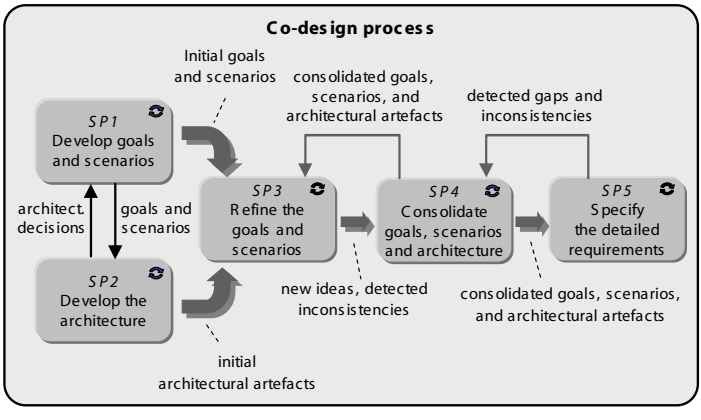

Fig. 3. Overview of the sub-processes of a co-design process

The goals of the five sub-processes depicted in Fig. 3 can be characterized as follows:

- Sub-process SP_1 supports the development of initial goals and scenarios at the layer L_i (see Section 2.3.2).
- Sub-process SP_2 supports the development of initial architectural artifacts at the layer L_{i+1}. Sub-processes SP_1 and SP_2 can be performed in an intertwined manner or, partly, in parallel.

- Sub-process SP_3 supports refinement of the goals and scenarios defined at layer L_i based on the architecture defined at layer L_{i+1}. The refinement results in goals and scenarios at layer L_{i+1}. In addition, the responsibilities for satisfying the goals and scenarios are assigned to the elements of the architecture. Thereby, the detection of mismatches between the requirements defined at layer L_i and the architecture at layer L_{i+1} is supported (such as goals that the architecture cannot satisfy or architectural elements not justified by system goals).
- Sub-process SP_4 is responsible for reconciling the requirements and the architecture at the two neighbouring layers L_i and L_{i+1}. It results in corrections and changes applied to the requirements artefacts and the architectural artefacts at both layers.
- In sub-process SP_5, the detailed (solution-oriented) requirements at layer L_i are specified based on the results obtained from the other sub-processes, i.e. consolidated goals, scenarios, and (coarse-grained) architectural artifacts.

In the following, we motivate the use of goals and scenarios for supporting the co-development of requirements and architectural artifacts as well as the refinement of requirements across multiple abstraction layers. The use of goals and scenarios for these purposes is illustrated by a brief example (Section 2.3.3).

2.3.2 Goals and Scenarios

COSMOD-RE uses goals and scenarios to support the co-development of requirements and architectural artifacts as well as the refinement of system requirements into component requirements. Goals document stakeholder intentions and thereby refine the system vision into verifiable system objectives. Goals are typically solution-neutral. For example, they do not prescribe or assume the use of a specific technology. In addition to goals, scenarios are used to document concrete examples of system usage which lead to the fulfilment or unfulfilment of a defined goal. Typically, a scenario is defined as a sequence of interaction steps. Goals and scenarios prevent the stakeholders from making premature design decisions or focusing on solution details too early. Moreover, goals and scenarios define a sound basis for identifying and exploring possible solutions in the downstream development activities:

- *Definition of an initial, coarse-grained architecture*: Typically, as soon as the system goals and scenarios are defined and agreed, an initial, coarse system architecture can be developed. Therefore, goals and scenarios are well suited to support the co-development of requirements and architectural artifacts.
- *Definition of solution-oriented system requirements*: The defined, agreed, and verified system goals and scenarios (along with the initial, coarse architecture) provide a sound basis for defining detailed, solution-oriented requirements for the system. The resulting, detailed system requirements are typically more stable if agreed and consolidated goals and scenarios are used as input.
- *Definition of component requirements*: Typically, it is significantly easier to refine goals and scenarios than refining, for instance, a data model, a functional model, or a behavioral model. Hence, in COSMOD-RE, system goals and system scenarios are refined into component goals and components scenarios prior to defining (solution-oriented) component requirements. In this way, component goals and component scenarios are provided to support the definition of (solution-oriented) component requirements.

2.3.3 An Example

We illustrate the development and refinement of goals and scenarios across two abstraction layers (the system layer and the functional decomposition layer; see Section 2.2) using a simplified example of an adaptive cruise control (ACC) system.

We document goals and their relationships using the KAOS goal model [13]. A simplified goal model for the ACC system is depicted in Fig. 4 on the left. In addition, details about the defined goals are documented using a goal template (see [7] for details). Scenarios are documented using a use case diagram, use case templates, and sequence diagrams. The use cases identified for the ACC system to fulfill the identified system goals are depicted in Fig. 4 (on the right-hand side).

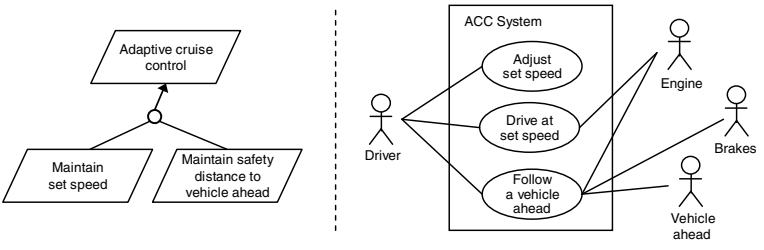

Fig. 4. Examples of a goal model (left) and a use case diagram (right) for the ACC system

Based on the system goals and scenarios, the system architects define an initial, coarse architecture for the ACC system that satisfies the defined goals and scenarios. The initial architecture depicted in Fig. 5 defines the major functional components of the ACC system and their interfaces.

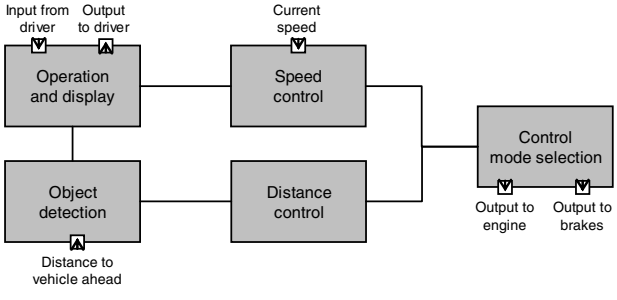

Fig. 5. Example of an initial, coarse architecture for the ACC system

Based on the initial architecture, the system goals and scenarios are (if required) refined and related to the architectural elements. Thereby, mismatches between the requirements (goals and scenarios) and the suggested architectural solution can be detected early. For instance, an architectural component may have no associated goals and thus no justification for its existence. Fig. 6 depicts the refinement of a system goal and the assignment of the resulting sub-goals to individual, architectural components.

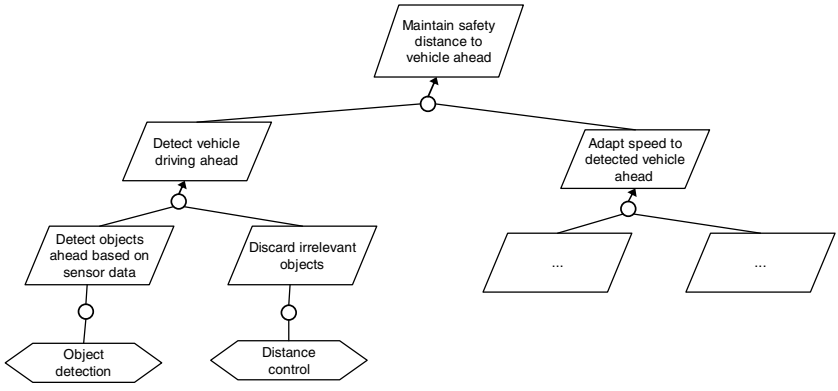

Fig. 6. Example of refining a system goal into a set of component goals

Based on the relations between the refined goals and scenarios and the initial architecture, problems and inconsistencies are detected and removed in order to align the goals, scenarios, and architecture. Furthermore, the alignment often stimulates new ideas for goals, scenarios, or architectural solutions (see [7] for more details on aligning goals, scenarios, and the architecture). The aligned goals, scenarios, and architectural artifacts are subsequently used as a basis for defining the detailed, solution-oriented system requirements (see [7] for details).

2.4 Further Reading and Outlook

Additional details on COSMOD-RE can be found in several publications. The idea of co-developing requirements and architecture is explained in more detail in [14]. A comprehensive description of COSMOD-RE has been included in a recent textbook on requirements engineering, see [7].

During the construction of COSMOD-RE, an initial industrial evaluation of the method has been performed. During this evaluation, COSMOD-RE was successfully applied to various industrial examples. Presently, a systematic evaluation of in a number of different domains such as automotive, avionics, or medical technology is performed. Initial results of the evaluation of COSMOD-RE in the automotive domain are reported in [15].

To support automated, formal reasoning in COSMOD-RE, parts of the method are being formalized [16]. Furthermore, COSMOD-RE is being integrated into a seamless, model-based, overall development process for software-intensive embedded systems. This work is conducted within the German Innovation Alliance SPES 2020 (Software Platform Embedded Systems).

3 Requirements Engineering in Community Networks

Business process modeling and business process management, including their software equivalents of ERP and workflow systems, have been the dominant

paradigm of the 1990's. Since the turn of the century, however, the growing role of the Internet begins to lead to a shift in emphasis, which may lead beyond well-structured process organizations to much more flexible capability-based network organizations. They blur organizational boundaries and emphasize *generic platforms*, *innovation at the margin* of the network, and *rapid reorganization* over central process planning. Social software has begun to play a serious role in many enterprises, as a medium of knowledge management, project management, or public relations. Important contributors to this success are *low entry barriers*, *easy usability*, and *great flexibility*. Thus, community networks have become an informal, not necessarily friendly counterpart to the structured systems described in section 2.

In stark contrast to the easy initial entry, the long-term structuring of these information networks, e.g. in order to interface them with organizational requirements management, is far more difficult. Bottom-up voices from "community of practice" in Web 2.0 media differ strongly from contract-based requirements engineering documents with their many formal and semi-formal interdependencies. An architecture intended to support community information systems on this basis requires particular flexibility. A community needs to be able to *observe* itself, to *analyze* and maybe even *simulate* its behavior, in order to evolve its rules of cooperation. In short, it needs *reflective capabilities*. Only then it can survive as a community, and even interact with more formal parts of the structured process organizations. In this section, we shall be interested in the question how these properties can be achieved, such that – in the long run – the present disparity between the structured product family and process world of formal organizations can reach a more fruitful interplay with the social network and software environment.

Our approach originates from an operational theory of media called *Transcriptivity Theory* [17]. It describes the operational semantics of media artifacts with three basic operations *transcription* (of sets of media into new media), *localization* (filtering and adapting received media to your own culture) and *addressing* (media to specific target groups of readers). In our abstract software architecture *ATLAS* (*Architecture for Transcription, Localization, and Addressing Systems*) [18], transcriptivity theory has been re-interpreted as a design theory for information systems in which scalable and interoperable repositories on top of databases support communities by web service technologies for multimedia content and metadata management.

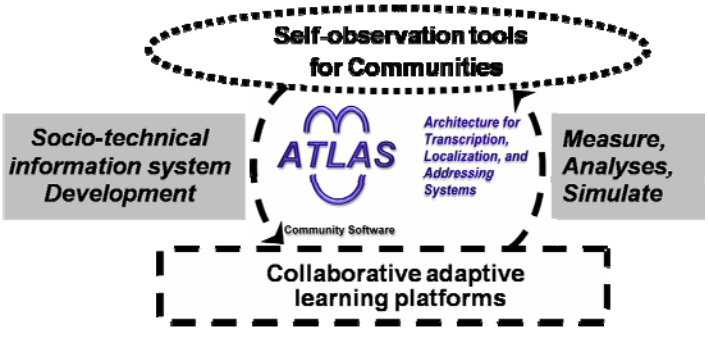

Fig. 7. The ATLAS Development Model

In its reflective conception, ATLAS-based community information systems are tightly interwoven with a set of media-centric self-monitoring tools. Hence, communities can constantly *measure*, *analyze* and *simulate* their activities to improve the understanding of their needs, in turn simplifying the collaboration between developers and users (cf. Fig. 7). In the future, all community design and engineering activities should be carried out by the community members, regardless of their technical knowledge. In the sequel, we illustrate challenges and solutions for this concept using a specific example of network organization, cooperative learning.

The ROLE (Responsive Open Learning Environments) architecture (cf. Fig. 8) is an example application of the ATLAS architecture specialized on the development of Internet based personal learning environments. ROLE is developed in the context of an EU-funded Integrated Project with the same title.

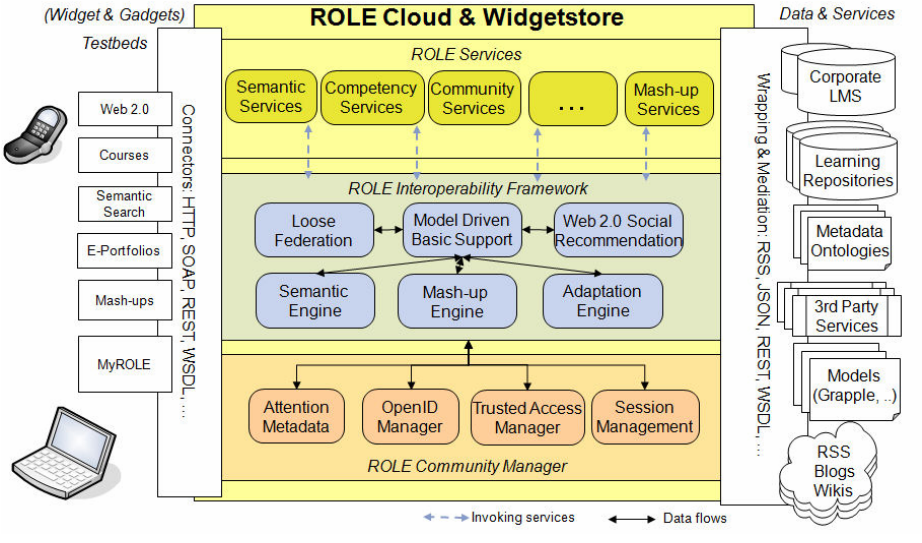

Fig. 8. The ROLE Architecture

The design of the *ROLE Cloud* – the open network of learning resources and services -- is based on a lightweight service-oriented architecture fostering communication and data provision. The *ROLE Community Manager* provides a basic service layer for privacy and trust mechanisms as well as traceable learning histories for learner communities. The *ROLE Interoperability Framework* improves the support for distributed third party developers from industrial as well as open source contexts to foster collective enhancements of the ROLE framework and to open new opportunities for SMEs in the cooperative learning domain.

The ROLE methodologies and specifications evaluate solutions for deployment, repurposing, customization, personalization, and validation in the *ROLE test-bed frameworks*, before outreaching to further communities. We aim at a self-sustaining

networked community of online learners and developers, at developing a best practice sharing strategy for virtual, collaborative learning activities, and at examining characteristics and circumstances of such a learning network. A special focus is thereby put on the elicitation of early phase requirements in communities. In the next subsections, we first discuss the special RE needs of such communities which are quite different from those of formal organizations. We then structure these needs using Eric Yu's i* model, and finally discuss a couple of community monitoring and analysis tools we have developed to support the reflection process.

3.1 Community of Practice

The conceptualization of RE for communities of practice (CoP) requires a thorough investigation of the existing forces, which can influence community member behavior. Communities are influenced by many external factors called *disturbances*. These disturbances may have a negative, positive or neutral influence on the learning processes within the community [19]. Without disturbances, communities are in danger of falling into social or cognitive lock-in situations [20, 21]. Usually, the disturbances are coming from outside but are changing the community inside. Thus, the evolution of communities takes place in a lifelong loop.

A CoP is characterized by three dimensions introduced by Wenger [22]:

Mutual engagement (ME): Community members are required to be engaged in interaction within their community. Hereby, membership in a community is not just belonging to one organization, and a comunity is not only a set of members having personal contacts with other members.

Joint enterprises (JE): The common goal of a community of practice, which binds members together is the result of a collective process of negotiation which reflects the full complexity of mutual engagement.

Shared repertoire (SR): The communal resources include routines, words, stories, gestures, symbols, genres, actions, tools, ways of doing things, concepts, etc. that the community has produced or adopted during its existence, as part of its practice.

A community is thus an open group of people who share a concern or a passion and who interact regularly about it [22]. Two aspects are combined: the social practice of the community as a collective phenomenon and the identity of its members as an individual phenomenon. Individual learning is inherent in the processes of social participation in the community. Knowledge and learning are not abstract models but relations *"between a person and the world"* [23] or *"among people engaged in an activity"* [24]. Individual learning is mainly based on *"legitimate peripheral participation"* [22]. During the participation process, an individual might enter the community as a beginner at the periphery and then gain a more central position over time by cognitive apprenticeship. This acquisition process leads to an intensified inclusion in the social practice of the community. Learning is based on this process of inclusion of outsiders. The communities of practice themselves can be seen as *"shared histories of learning"* [22]. The mechanism of (social) identification of individuals in the social context of the community plays a key role for community

formation as well as community survival but there does not yet exist a stable lifecycle theory how to achieve continued success of a community; our hope is that a more systematic community RE process can contribute to the development of such a lifecycle support.

3.2 Towards a Community RE Process

To map learning communities participating in RE to community concepts, we need a theory that explains socio-psychological aspects, which influence community members through relations between human agents, technologies and resources. For this purpose, Actor-network-theory (ANT) [25], which makes no distinction between human and non-human actors, can be adopted, because it intertwines actions, influences, and results of actions independently if automated or human.

In addition to this action-related approach to network modeling, RE also needs to capture intentions of (usually human) actors and the strategic dependencies which are considered as the long-term basis for cooperation in a community of specialists. To capture this intentional aspect of networks, we chose the i* framework [26], which enables the description of strategic relations between actors within a particular socio-technical system in a clear way [27]. In i* we find the following premises: Agents are dependent on other agents. They act intentionally, because they follow goals, have beliefs, competences, commitments, needs and desires. At the same time, agents are strategic actors, because they have to cooperate to reach their goals. Intentional dependencies can be made explicit with i* to disclose the reasons behind observable processes. In the following we provide a brief overview of i* modeling components.

In i*, strategic dependencies (SD) are used for modeling intentional, strategic relationships among actors in form of an actor diagram. Strategic rationale (SR) models cover the individual rationale behind dependencies, and analyzes alternatives and dependencies fulfillment by a goal diagram. The model of our community RE process is an example of strategic rationale. The main syntax elements of i* are: *Actors*, *Actor Associations*, *Goals*, *Softgoals*, *Tasks*, *Resources* and *Links*. An *actor* can be any active entity that carries out actions to achieve goals by exercising its know-how. An *agent* is an actor with concrete, physical manifestations, such as a human individual or a software system. Actors in SR contain intentional *boundaries*. All elements within such a boundary are explicitly desired by an actor. The fulfillment of goals within boundaries can depend on intentions of other actors and is represented as *strategic dependencies*. Such a dependency consists of a *dependee*, who is required to fulfill a *dependum* by a *depender*. Links in SR can be of three types: *decomposition*, *means-ends* and *contribution*. A decomposition link describes a decomposition of a task in subtasks, subgoals, resources or softgoals. Means-ends-links bind a goal to achieve with a task, which has to be executed for attaining the goal. Finally, contribution links connect softgoals to tasks.

An i* model for RE within a community of practice (*CoP*) is presented in Fig. 9. The main idea is firstly to combine analysis of community-generated content and system usage, and secondly to provide the *Social System* of CoP members with a service for expressing their requirements.

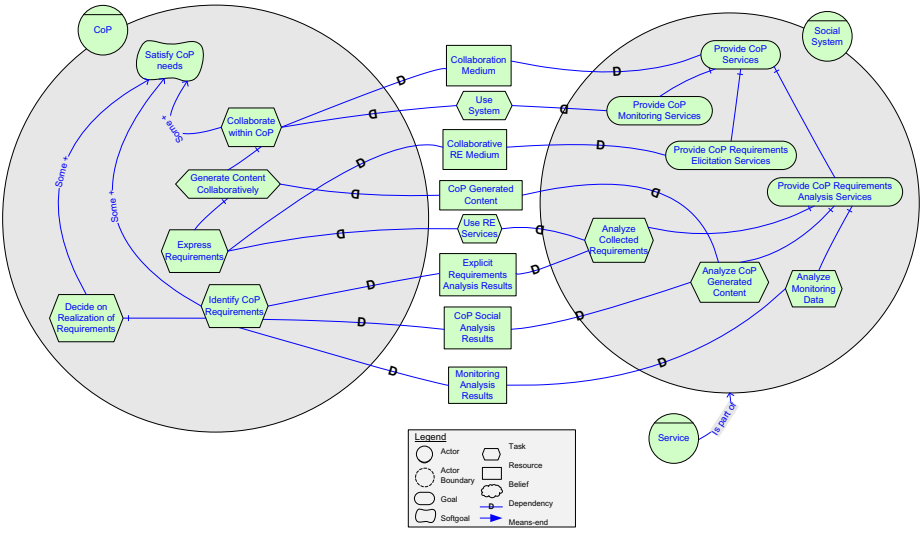

Fig. 9. Model of Requirements Engineering in CoPs

CoP and *Social System* are modeled as agents (big dark circles) according to i* terminology. The *Social System* is a software environment used by CoP members in order to fulfill the softgoal of satisfying CoP needs (Joint Enterprise). Thus, the *Social System* presents a composition of different Services, depending on the community and its practices. The *Social System* should additionally provide services specified for the RE process: *Provide CoP Monitoring Services*, *Provide CoP Requirements Analysis Services*, *Provide CoP and Requirements Elicitation Services*.

Any community requires a shared repertoire, which can be created and enriched, as community members collaborate with each other (*generate content collaboratively*). Therefore, the *Social System* should offer CoP services as a collaboration medium. However, the availability of a service is not enough for collaboration. According to the definition of the mutual engagement dimension of CoP, community members are required to be engaged in interaction; thus, they do not only consume, but also *generate content collaboratively*. Hereby, CoP-generated content can be applied to get insights in social characteristics of the community and thus *identify CoP requirements*. Log data of system usage can serve as the next source for requirements mining. For this reason, the *Social System* provides a *CoP monitoring service*, which collects data for later analysis.

In our ATLAS prototype, the services of the *Social System* are technically delivered though extensions to LAS, a Lightweight Application Server for prototyping, testing and hosting new community web services [29]. In the remainder of this section, we briefly discuss two of these interactive services intended to assist the RE process in a community, one for the structured monitoring interactive assessment of the quality of existing services under special consideration of mobile services (MobSOS [28]), the other for the self-analysis of undesirable developments in the community social network structure itself (PALADIN [19]). Both services have already been successfully used in numerous research and practice projects.

The main purpose of MobSOS is the integrated support for (mobile) web service quality measurement and evaluation. The underlying success model was inspired by the widely used DeLone/McLean IS success model [30]. MobSOS offers two modules for the acquisition of model test data: monitoring and user surveys. The monitoring module logs communication between users and services together with context information, thus enabling context-dependent quality analysis, according to the data model shown in Fig. 10. All other quality data required by the success model are collected by the user survey module.

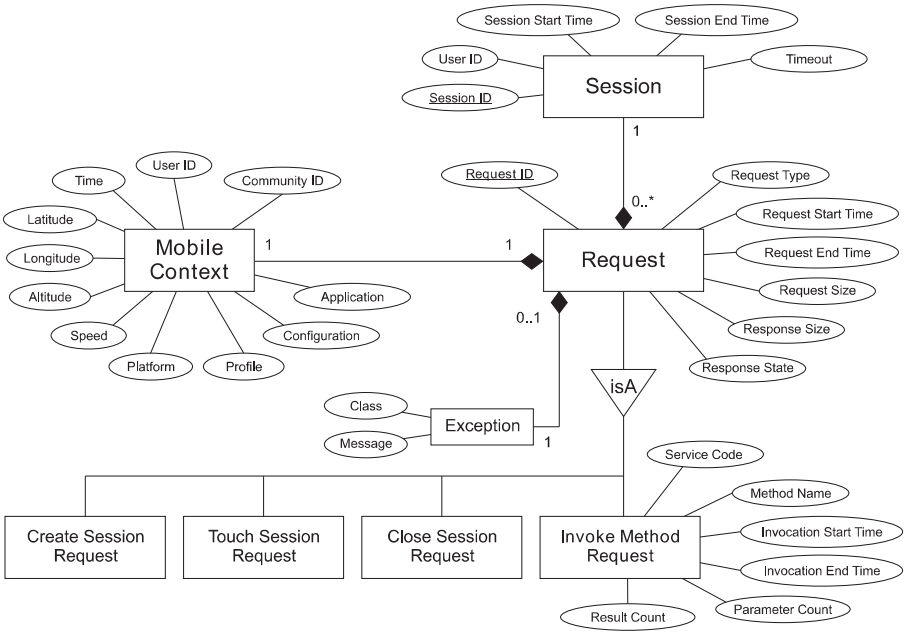

Fig. 10. Monitored Community System Usage and Context in MobSOS

The PALADIN service has been developed for the detection of multidimensional disturbance patterns in communities. It includes a pattern structure definition, the *Formal Expression Language for Patterns* (*FELP*), and algorithms for the application of the patterns on the data. Each pattern consists of a *name*, a *disturbance*, a *description*, *forces*, *force relations*, a *solution*, a *rationale*, and *pattern relations*. The *forces* are relevant actors of the pattern; the *forces relations* are relations between actors. The *solution* provides the advices to solve the pattern situation. The relations are used for reasoning about the forces and the disturbances. PALADIN has been successfully applied in the detection of community patterns in ten thousands of mailing lists. As an example (cf. Fig. 11), the *conversationalist pattern* identifies members (dark circles) in communities which are engaging also in conversations by participating in threads without having started them (comparable to “flame warriors”). The conversationalists discuss ideas, carry on conversations and share opinions. The

conversationalist pattern has two relations - `rcnv1 = (own thread, conversationalist, thread)` which relates the member to the threads which he/she has started and `rcnv2 = (post thread, conversationalist, thread)` which relates the member to the threads where he/she has posted. The conditions for a member to be a conversationalist are:

> `The member has started at least one thread; AND the member has posted in at least one thread started by other members; AND the maximum number of messages, part of a conversationalist's thread, must be greater than n (in the figure: n = 4); AND the member has posted more than k messages overall.`

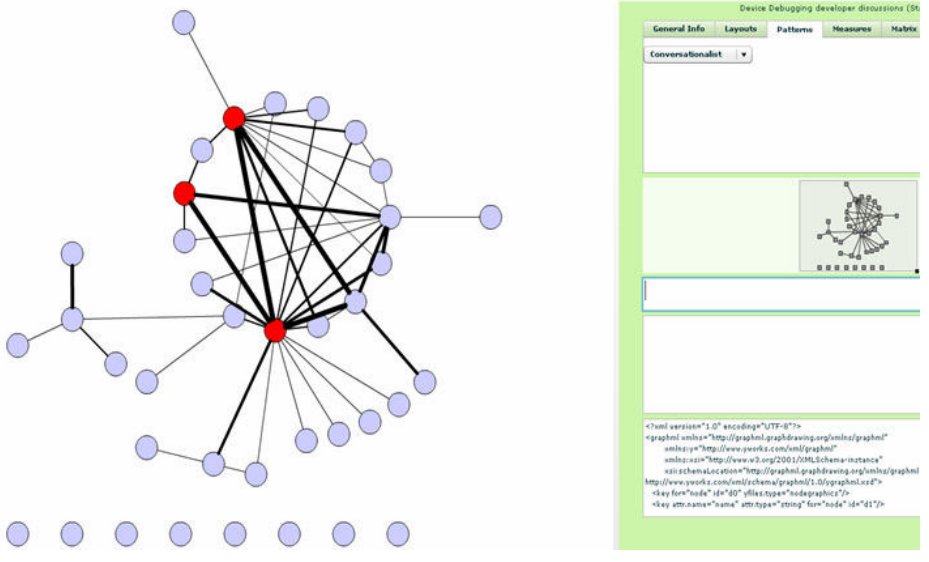

Fig. 11. Matching the Conversationalist Community Pattern in PALADIN

Beside these two implicit methods of requirements identification, the ATLAS environment also provides communities with a mobile collaborative RE service, where community members can express requirements explicitly in a multi-media assisted scenario-based approach.

From these varied sources and RE services, the generated pool of requirements can be huge and diverse. Hence, the *Social System* (cf. Fig. 9) has to analyze collected requirements first and then reports the results to the *CoP*. Using the results from social, system usage and requirements analysis, it is the task of the *CoP* to prioritize requirements and to decide on their particular realization.

4 Discussion and Outlook

The two approaches described in the previous sections demonstrate that, with the ever-growing spread of software and information technologies in all branches of engineering, business organizations and society, requirements engineering approaches have diversified.

The COSMOD-RE method presented in Section 2 addresses specific issues in the engineering of complex, technical systems. The main goal of the COSMOD-RE method is to support the tightly intertwined development of the requirements and the architecture (structure) of complex technical systems from coarse-grained system requirements down to detailed hardware and software component requirements. The approach has the advantage that the development of requirements and architectural models is structured by means of well-defined abstraction layers and the engineers are supported in choosing the appropriate level of abstraction for their models throughout the process.

The community RE approach presented in Section 3 focuses on involving a diversified community in the system development process. In contrast to COSMOD-RE, the main emphasis of the community RE approach is in fact the community itself. The community RE approach has the advantage of a very direct impact of new user and developer ideas on not just the evolution of the system but also of its user community. Fresh ideas from the margin of the community network can assist in radical innovation, and there is some evidence from practice applications that the additional community self-control offered by the monitoring services can reduce to some degree the brittleness many community systems have suffered from in the past.

The two approaches described in this chapter address two different aspects of complexity in requirements engineering: the complexity of the user and developer community and the complexity of the system itself. While in this contribution, the complexity of the community and the complexity of the system are treated separately, in the near future, the need arises to address both kinds of complexity in coherent requirements engineering approaches: With the upcoming, converged future internet, embedded systems controlling physical processes will be tightly integrated with information systems providing innovative services to their users based on the up-to-date information about the physical world. For instance, in the energy domain, a network of smart metering devices can provide real-time information about power consumption. Business information systems can exploit this data to offer advanced analysis services for decision makers in energy supply and trading companies. Similarly, in logistics, sensor networks can capture information about the location and status of physical goods in order to support advanced services for logistics planning.

At the same time rigid organization forms will dissolve into often loosely coupled and rapidly changing community networks. Organizations who have their roots in technical domains (such telecommunications or power supply infrastructure) are being transformed into service providers offering information-intensive software services to a complex user community via the internet. One of greatest challenges for the next decade will be to support this change by providing development approaches that incorporate the ideas, needs, and wishes of user communities as well as the technical knowledge of domain experts in order to successfully develop the next generation of software-intensive systems in the converged, future internet.

References

1. Jarke, M., Pohl, K.: Establishing Visions in Context – Towards a Model of Requirements Processes. In: Proc. 14th Intl. Conference on Information Systems, pp. 23–34 (1993)
2. Nagl, M., Marquardt, W.: Collaborative and Distributed Chemical Engineering – From Understanding to Substantial Design Process Support – Results of the IMPROVE Project. LNCS, vol. 4970. Springer, Heidelberg (2008)
3. Brooks, F.: No Silver Bullet – Essence and Accidents of Software Engineering. IEEE Computer 20(4), 10–19 (1987)
4. Pohl, K., Sikora, E.: COSMOD-RE: Supporting the Co-Design of Requirements and Architectural Artifacts. In: Proc. 15th IEEE Intl. Requirements Engineering Conference, pp. 258–261. IEEE Computer Society, Los Alamitos (2007)
5. Leveson, N.: Intent Specifications – An Approach to Building Human-Centered Specifications. IEEE Transactions on Software Engineering 26(1), 15–35 (2000)
6. Weber, M., Weisbrod, J.: Requirements Engineering in Automotive Development – Experiences and Challenges. IEEE Software 20(1), 16–24 (2003)
7. Pohl, K.: Requirements Engineering – Fundamentals, Principles, Techniques. Springer, Heidelberg (to appear, 2010)
8. McMenamin, S., Palmer, J.: Essential Systems Analysis. Prentice Hall, London (1984)
9. Swartout, W., Balzer, R.: On the Inevitable Intertwining of Specification and Implementation. Communications of the ACM 25(7), 438–440 (1982)
10. Nuseibeh, B.: Weaving Together Requirements and Architectures. IEEE Computer 34(3), 115–117 (2001)
11. Harel, D., Pnueli, A.: On the Development of Reactive Systems. NATO ASI Series, vol. F13, pp. 477–498. Springer, Heidelberg (1985)
12. Kohring, C., Lefering, M., Nagl, M.: A Requirements Engineering Environment within a Tightly Integrated SDE. Requirements Engineering, vol. 1, pp. 137–156. Springer, Heidelberg (1996)
13. Van Lamsweerde, A.: Requirements Engineering – From System Goals to UML Models to Software Specifications. Wiley, Chichester (2009)
14. Pohl, K., Sikora, E.: Structuring the Co-Design of Requirements and Architecture. In: Proc. 13th Intl. Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2007), Trondheim, Norway (2007)
15. Sikora, E., Pohl, K.: Evaluation eines modellbasierten Requirements-Engineering-Ansatzes für den Einsatz in der Motorsteuerungs-Domäne. In: Proc. Erster Workshop zur Zukunft der Entwicklung softwareintensiver, eingebetteter Systeme (ENVISION 2020). Gesellschaft für Informatik, LNI, vol. 160 (2010)
16. Sikora, E., Daun, M., Pohl, K.: Supporting the Consistent Specification of Scenarios Across Multiple Abstraction Levels. In: 16th Intl. Working Conference on Requirements Engineering – Foundation for Software Quality (REFSQ 2010) (2010)
17. Jäger, L.: Transkriptivität - Zur medialen Logik der kulturellen Semantik. In: Jäger, L., Stanitzek, G. (eds.) Transkribieren - Medien/Lektüre, Fink, München, pp. 19–41 (2002)
18. Jarke, M., Klamma, R.: Reflective community information systems. In: Manolopoulas, Y., et al. (ed.) ICEIS 2006. LNBIP, vol. 3, pp. 17–28. Springer, Heidelberg (2006)
19. Klamma, R., Spaniol, M., Denev, D.: PALADIN: A Pattern Based Approach to Knowledge Discovery in Digital Social Networks. In: Tochtermann, K., Maurer, H. (eds.) Proceedings of I-KNOW 2006, 6th International Conference on Knowledge Management, Graz, Austria. J.UCS (Journal of Universal Computer Science) Proceedings, pp. 457–464. Springer, Heidelberg (2006)

20. Granovetter, M.S.: The strength of weak ties: A network theory revisited. In: Lin, P.M.N. (ed.) Social Structure and Network Analysis, pp. 105–130. Sage, Beverly Hills (1982)
21. Krippendorf, K.: Some principles of information storage and retrieval in society. General Systems 20, 15–35 (1975)
22. Lave, J., Wenger, E.: Situated Learning: Legimate Peripheral Participation. Cambridge University Press, Cambridge (1991)
23. Duguid, P.: The Art of Knowing: Social and Tacit Dimensions of Knowledge and the Limits of the Community of Practice. Information Society 21(2), 109–118 (2005)
24. Østerlund, C., Carlile, P.: How practice matters: A relational view of knowledge sharing. In: Huysman, M., Wenger, E., Wulf, V. (eds.) Communities and Technologies - Proceedings of the First International Conference on Communities and Technologies (C&T 2003), pp. 1–22. Kluwer Academic Publishers, Dordrecht (2003)
25. Latour, B.: On recalling ANT. In: Law, J., Hassard, J. (eds.) Actor-Network Theory and After, Oxford, pp. 15–25 (1999)
26. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE 1997), Washington D.C., USA, January 6-8, pp. 226–235 (1997)
27. Bryl, V., Giorgini, P., Mylopoulos, J.: Designing socio-technical systems: from stakeholder goals to social networks. In: Requirements Engineering, vol. 14(1), pp. 47–70. Springer, New York (2009)
28. Renzel, D., Klamma, R., Spaniol, M.: MobSOS - A Testbed for Mobile Multimedia Community Services. In: 9th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2008), Klagenfurt, Austria (May 2008)
29. Spaniol, M., Klamma, R., Janssen, H., Renzel, D.: LAS: A Lightweight Application Server for MPEG-7 Services in Community Engines. In: Tochtermann, K., Maurer, H. (eds.) Proceedings of I-KNOW 2006, 6th International Conference on Knowledge Management, Graz, Austria. J.UCS Proceedings, pp. 592–599. Springer, Heidelberg (2006)
30. DeLone, W.D., McLean, E.R.: Information Systems Success: The Quest for the Dependent Variable. Information Systems Research 3(1), 60–95 (1992)