Conceptual Application Domain Modelling

Bernhard Thalheim[1] **Klaus-Dieter Schewe**[2] **Hui Ma**[3]

[1] Christian-Albrechts-University Kiel, Institute of Computer Science, Kiel, Germany, thalheim@is.informatik.uni-kiel.de

[2] Information Science Research Centre, Palmerston North, New Zealand kdschewe@acm.org

[3] Victoria University Wellington, School of Mathematics, Statistics and Computer Science, Wellington, New Zealand, Hui.Ma@mcs.vuw.ac.nz

Abstract

Application domain description precedes requirements engineering, and is the basis for the development of a software or information system that satisfies all expectations of its users. The greatest challenge in this area is the evolution of the application domain itself. In this paper we address this problem by explicit consideration of *application cases* that are defined by user profiles and intentions and the system environment, i.e. scope and context. User profiles and intentions are captured through the concept of *persona*. We show how the application domain description can be mapped to requirements and discuss engineering of application domain descriptions.

1 Modelling Information and Software Systems

Information Systems are software systems with a focus on operating on data and thus the information they provide. Typically, the engineering of Information Systems (or software systems in general) is divided into several phases, the first of which being requirements engineering. However, requirements may vary depending on the evolution of the application itself or changes in the technical environment or the system users.

An an example consider the classical example of elevator control that is used in many textbooks, e.g. (Wieringa 2003). An elevators control system coordinates the movement of a number of elevator cages that serve a number of floors. However, this problem description already contains a fundamental decision with respect to the application domain, which is to enable easy movement of a number of users from one floor to another one. Instead of giving preference to an elevator solution the decision could have been to use only stairs or escalators, or to install a "pater noster", i.e. a system using open boxes that turn around continuously. In terms of throughput and energy costs, a pater noster is significantly better than an elevator, but for elderly people it may be a safety hazard, and for wheelchair access it is not suited. Nevertheless, in many old business buildings in Northern Germany pater nosters were quite common.

A similar observation can be made for the specification of traffic control using signaling, control by a computer, street topology, error handling, etc. (Jackson & Sztipanovits 2006). In this case the technical solution has already been fixed prior to the elicitation of requirements.

This shows that different solutions envisioned in the application domain may lead to completely different requirements and consequently system specifications. Which solution is the most appropriate and which flexibility and variability is needed depends on the application domain. Moreover, Jackson observes that solutions based on the single-model paradigm are not achievable and integration of different viewpoints is only achievable for very simplistic situations (Jackson & Sztipanovits 2006).

1.1 Application Domain Description and Modelling

Bjørner divides Software Engineering into three main phases: *application domain description*, *requirement prescriptions*, and *system specifications* (Bjørner 2006). He calls these phases the *system development triptych*. The application domain description is a model describing the application domain, its entities, functions, events, and behaviour. It utilises formal, semi-formal or natural language, which permits the formulation of a set of theorems or postulates or properties that are claimed to hold for the domain model.

Thus, application domain modelling (similarly, but more specifically: product line engineering, enterprise modelling) is the first step of software development processes, for which a large body of knowledge has already been developed, e.g. (Boehm 2006, Lowe 2003, Maciaszek 2001, Wieringa 2003). In particular, the goal-oriented framework KAOS (Darimont & van Lamsweerde 1996) uses an outer semi-formal layer for capturing requirements engineering concepts and their structuring and presentation, and an inner formal assertion layer for their precise definition and reasoning about them.

A theory basis for application domain languages is currently under development, based for instance on description logics (Lambrix & Padgham 1996), artificial intelligence (Anh & Moore 1996), logical calculi (Hansen & Hung 2007), ontologies (Jackson & Sztipanovits 2006, Missikoff & Schiappelli 2005), or formal methods (Bjørner 2006). In the last case the formal VDM language is used to cover most (but not all) facets of an application domain such as business processes, intrinsics, support technology, management and organisation, rules and regulations, scripts, and human behaviour, but it is insufficient for the main facet of information systems: modelling of information states and evolution of states.

Typically, application domain modelling is based on the elicitation of properties of one application domain solution and the description of this solution. This approach is appropriate as long as the application domain is relatively stable and solutions do not change. There are, however, applications for which the solutions evolve and change over time. In this case, requirements that have been the basis for the software system are also evolving and changing, whereas the problem supported by the system remains relatively stable. Therefore, we propose to first model the application problem itself.

1.2 Requirements Prescription and Information Systems Modelling

Application domain modelling can be based on features or business events (Robertson & Robertson 2005), mission statements (Wieringa 2003), or business use cases (Maciaszek 2001). The application domain model is mapped to requirements that prescribe further system development. A *requirement* (Berztiss 2001, Gunter, Gunter, Jackson & Zave 2000) is a verifiable statement prescribing some property that a software system should possesses. Requirements are specified through declarations or remarks, reports of facts, or opinions. Their fulfillment can be verified or measured.

Requirement statements are compiled into a documentation which prescribes desired properties of machines, i.e. what the machine should and should not offer regarding data, control and functions. Since machines do not operate without an environment, environment description is often included into requirements. Such statements can be supported by diagrammatic methods, e.g. UML.

Following requirements engineering the association between the requirements model and the conceptual model is well understood (Rolland 2006) and supported by a large variety of UML diagrams or by advanced ER models (Thalheim 2000*a*, Thalheim 2000*b*). The database schema specifies the structuring of the database. Functionality can be added on the basis of the HERM algebra. Distribution is specified through extended views and services. The user presentation system can be described on the basis of the website specification language SiteLang (Schewe & Thalheim 2005). Typically it is required that the specification satisfies the requirements.

1.3 Application Solution Modelling used for most Applications

Classical software development is based on a vision of the solution to a number of problems to which solution the software system has been devoted. Requirements describe the application solution for which a software support is going to be developed. Problem analysis targets in development of application solutions. These solution can be mapped to requirements. Therefore, classical software engineering is devoted to *application solution engineering*, and requires severe changes within the system, if another solution is selected.

Application solution modelling is the basis for a prescription of requirements for one envisioned solution. This solution solves a problem that might have a number of other solutions. The solution that has been chosen for the software system has advantages over other solutions, but may also be a less optimal one, if the environment, the culture or the users change.

Therefore, in this paper we extend application domain description by a framework for *application cases* that are mapped to business use cases, stories, and portfolios. The challenge of application domain description and modelling is

- to achieve a set of languages, principles, methods, theories and techniques,
- as well as a set of management practices,
- which together cover all of today's and the immediately foreseen applications, their variations and their evolution in future,
- which by careful use and thoughtful consideration support software systems during evolution from initial development via repeated adaptive and perfective maintenance to final disposition, and
- which ensure software correctness as much as humanly conceivable.

The grand challenge of application domain modelling is the evolution of the application domain itself. In particular, this becomes crucial for web information systems (WISs) due to their low 'half-life' period and the high potential of WISs for evolution, migration and integration.

2 Specific Demands of Web Information Systems

Our framework of application cases has already been successfully applied in the area of WISs, where a broad coverage of solutions for the same problem can be observed. Developing more than three-score WIS such as infotainment, community, edutainment and e-commerce websites we had to realise that the application domain itself must be far better taken into consideration than it is classically done for information systems or for software systems. The classical approach of assuming that the application solution is fixed is not applicable to WIS. The application solutions evolve and change over time, vary for different regions or countries, depend on the user, are dependent on the context, and must be robust for different architectures. The treatment of the application cases depends on the habits of users, on laws and regulations of countries, on the culture of system utilisation and on the policy and profile of supporting companies.

A given problem in the application domain has typically more than one solution. Some of these solutions might be senseless at the given moment of time but might be preferable at another. This change of preference is often observed when application domains become a broader scope or merge with others. Whenever users or providers are changing a different solution must be enforced. Therefore, we also base our modelling approach on problem models.

2.1 Web Information Systems Engineering

WIS engineering also requires description, prescription, and specification of the presentation system. Presentation systems generalise approaches developed for human-computer interfaces (Lewerenz 2000) by storyboarding, by explicit treatment of portfolio of users, by adaptation and syndication facilities, by orchestration for different platforms, and by services for delivery, collection and compilation of information. User models are typically broader and must cover a large variety of users with specific requirements. Web applications also require systems that are easy and intuitively to use. Classical user-oriented and information-intensive applications could be nowadays based on WIS.

Classically user interfaces are built in dependence of the facilities the software system is supporting. In this case, the user has to learn how the system behaves and must adapt his/her behaviour to the systems behaviour. We overcome this mismatch by primarily considering the user worlds, the user stories, and the applications. Our approach is going to be based on *application domain description*. It extends ethnographical approaches developed for software engineering.

The main ingredients for application domain description for WIS are the characterisation of the user, the determination of the kind or scope of the system and of the context, and the description of the subject world. The latter is described through *application cases* or life case. We extract the portfolio and the tasks from application cases. Context and scope are defined in (Kaschek, Schewe, Thalheim & Zhang 2003) and (Schewe & Thalheim 2005) and thus not considered in this paper. We base the description of the profile and intention of users on (Schewe, Thalheim & Tretjakow 2006). Task modelling (Schewe & Thalheim 2005, Schewe & Thalheim 2007*b*) extends participatory task modelling (O'Neill & Johnson 2004) by explicit, integrated and formal specification of tasks into WIS specification. We, thus, concentrate our this paper on the first dimension of the triptych, i.e. application domain description through application cases.

2.2 Challenges of Modern Web Applications and of Evolving Applications

Nowadays software systems are supported by web presentation systems or are becoming web systems. They allow to cope with more complex tasks compared with systems of the past, with application stories in a large variety, and with dynamic requirements from the user and system side. They support users in their everyday life independently from their environment, their knowledge and skills level, their progress and their preferences. This change of attitudes and applications can only be satisfied if the user tasks and the user abilities are taken into consideration, if systems can be adapted on demand and on context, and if the application domain is driving the software. Early reports such as the Cutter Consortium report (Epner 2000) show that most WIS projects fail in meeting user and business needs, suffer from project delays, result in budget overrun, lack of required functionality, and provide poor quality of deliverables. These pitfalls illustrate the urgent necessity of a conceptual WIS development that is based on a severe analysis of application domain properties.

On-demand systems support utilisation just on demand, just to the right place, just for the user, just for the content that is available and just for the application situation that happens at the moment. These systems support applications from an application domain.

Example 1 Let us consider three situations typical for web application systems into which development we are or have been involved:

SeSAM parliamentarian support: The SeSAM system[1] aims in supporting parliamentarians in their everyday life as parliamentarian. The variety of applications changes with the change of rules how groups of parliamentarians are acting, with new rules that are settled after each new election, with new situations parliamentarians are involved into, and with discarding old approaches.

Digicult scout services: Scouts are collecting content for the Digicult system[2]. They interview witnesses and knowledgable people, compare their narrations with content obtained so far, annotate the new content depending on the situation, and interact between each other. These scouts are typically people that are temporarily hired.

eGovernment 'Fachverfahren': Government applications are supporting the work of governmental institutions though sophisticated web-based system. Classical system are 'hard-wired' in the sense that a small number of possible variants of governmental processes whereas much more variants are requested[3]. Companies such as DataPort are currently developing 'generic' processes.

These systems are oriented towards utilisation by users with very different background, knowledge, abilities and habits. Users do neither want to go through lengthy and annoying learning of software usage nor want to study and to study manuals. They want to continue with their everyday life and want to use software systems that are *embedded* into their application domain.

2.3 Manifold of Possible Solutions for the same Problem

WIS must support application solutions is a wide range. For instance, the German government initiative also targets on a proliferation and syndication of governmental processes('Fachverfahren'). These processes typically vary from county to county, from region to region and from city to city. For instance, Schleswig-Holstein is currently heading the initiative for syndication of processes supporting sovereign rights [4].

Example 2 The illustration example in the project is the application case *relocation* of a person, which consists of

- the change of basic relocation data including the possible removal of data on the old location,
- the change of official documents such as the passport,
- the optional change of relation enhancements such as the registration of pets, relocation of cars,
- the change of personal specific data such as family enhancements, or relationships to religious bodies,
- the change of data for additional relocation announcements such as tax, insurance changes, and
- specific additional tasks such as applications for housing allowances.

The person acts in the role of an *issuer*. We observe that *relocation* is enhanced by the profile of the issuer, by the specific tasks related to the *relocation* of the issuer, by specific laws and regulations, and

[1]The system has been developed as an addendum to infotainment services of cities such as `www.cottbus.de`, regions and counties. It is currently extensively used, e.g. in Cottbus.

[2]The Digicult system is currently the museum portal of Schleswig-Holstein. It is currently extended for support on demand for visitors and employees of museums and for information scouts who are collecting folklore knowledge.

[3]Governmental processes are of high complexity and of super-high variability. The German book of governmental processes consists of more than 40 books (more than 7.500 pages) describing in detail any possible process and sketching possible variations. This complexity is not supported by any system. Rather cities are selecting their process bundle and are interpreting it in dependence on their (organisational) structure, their needs and opportunities.

[4]The project is headed by DataPort and the author acts as a consultant in the project.

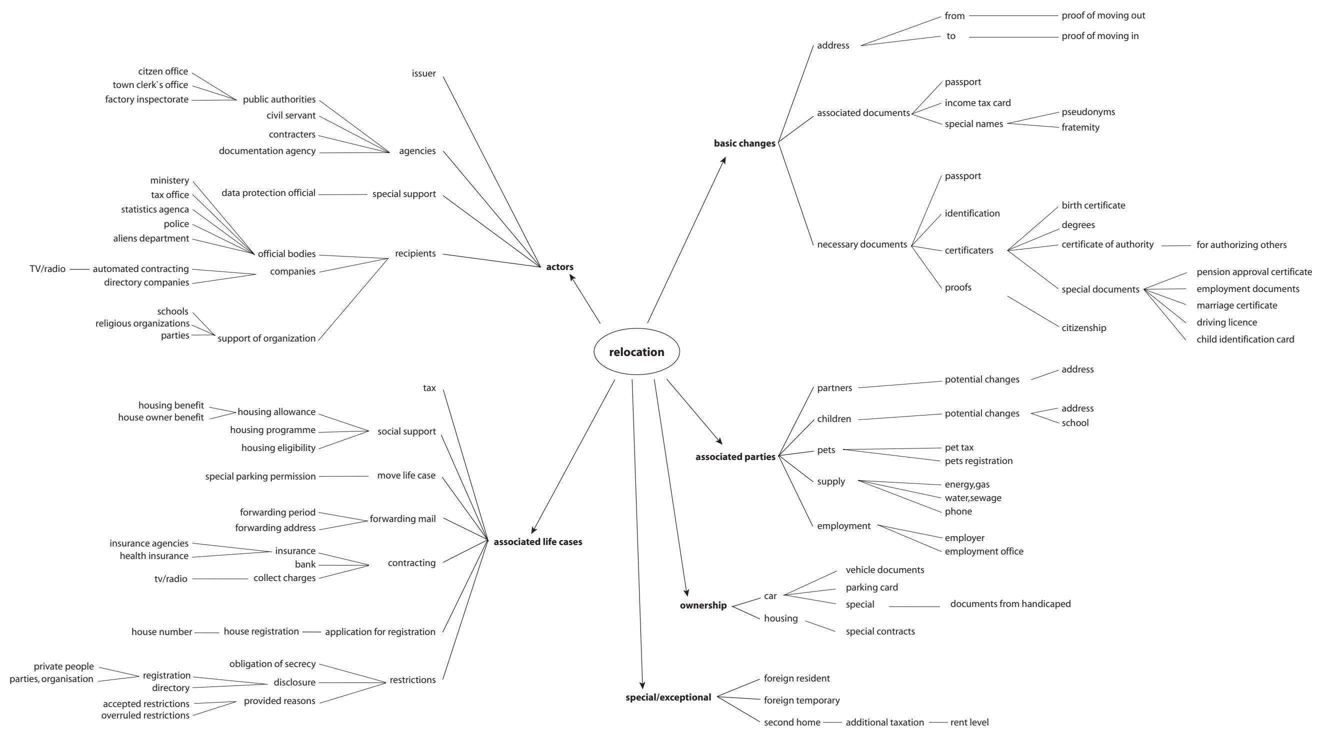

Figure 1: Facets of the relocation application case

by advanced functionality required for associating the application case with other application cases.

The application case *relocation* in Figure 1 consists of steps such as *change of address data*, *change of data for associated people*, *change of registration data* for cars, pets, etc., *change of specific data*, e.g. data for public authority responsible for aliens, *change of data for social aid*, etc. These steps are bundled together due to their relationship to one person and to one application case. The associations may be represented by adhesion of different steps, e.g. representing the association of steps by a hypergraph.

The application case in the example above has overall more than 150 variations in Germany. The general target is the same everywhere. The treatment is very different. We can group these variations in more than 2 dozen different solutions. About half of them are already supported by corresponding WIS. The integration or matching of these WIS is far from being trivial.

Moreover, the application case is associated to more than a dozen other application cases. Some of those cases are still only managed through paper exchange. This application case is one of the simplest among the application cases for proliferation and syndication of sovereign rights.

3 Application Cases and Problem Spaces for WIS

3.1 Demands and Essentials of Application Domain Description

Application domain description is the first dimension of WIS development. (Bjørner 2006) describes how the entire application domain theory can be developed. A general application domain model can be given that describes any facet within the application. This theory becomes huge and superficial complex. The corresponding description might consist of hundreds of VDM pages. We claim that we do not need to describe the entire application domain. We describe those parts that should be supported by the WIS.

The application domain is described by a number of properties, phenomena and various aspects that are desirable. A description includes designations, definitions, and refutable assertions. It can be formal or informal, sets a scope and a span, and expresses moods. Designations consist of a name, a recognition rule which purports to designate the phenomenon, and a general specification of the set of possible interpretations. Lexical definitions state the meaning of an expression in terms of other expressions. Ostensive definitions point to examples. Stipulative definitions assign a new meaning to an expression. Definitions may set bounds. Refutable assertions are claims that may be shown to be true or to be wrong.

The application domain description can be based on the following major steps:

1. Define the purpose of the application domain model
2. Select a paradigm for the application domain model
3. Determine the specific domain, specific situations, or scope of the application domain model
4. Identify an optimal process on which to develop the application domain model
5. Develop general criteria for goals, methods, and conditions
6. Develop goals for the application domain model
7. Develop methods for the application domain model
8. Identify conditions for the application domain model
9. Create a variable taxonomy or ontology for the application domain model
10. Finalize the application domain model prototype

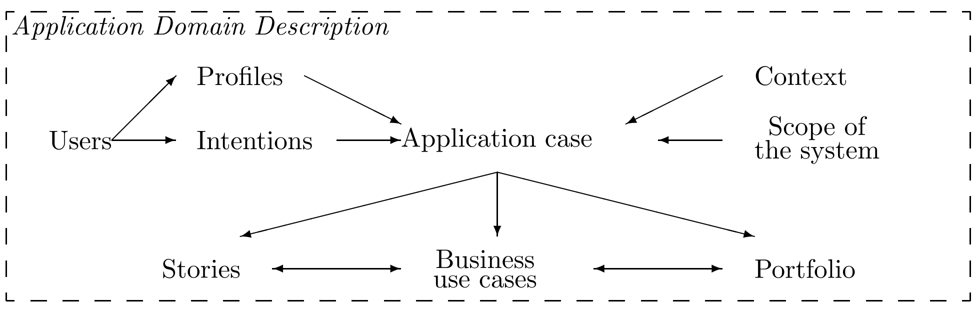

Figure 2: Elements to be specified for the application domain description

11. Formatively research the prototype application domain model

12. Revisit the goals, methods and conditions

13. Derive plans for testing the application domain model

14. Write up the application domain model

15. Derive requirements from the application domain model and improve the model

Application cases are the major element of the application domain model. They characterise something that occurs, happens or takes place in an application or actions or instances of occurring. They may apply to a happening without intent, volition, or plan. They may denote events, incidents, episodes, and circumstances within an application. They are associated with the context description, the characterisation of the kind of the system, and the user characterisation that consists of profiles of users and of description of their intention. The description framework depicted in Figure 2 supports an elegant and sophisticated elicitation of requirements and the derivation of the presentation system specification.

The application case study combines cognitive techniques with contextual approaches. It is extended during WIS utilisation analysis by traditional approaches. This combination allows to avoid the known disadvantages of the more specific elicitation approaches.

Users are characterised (Schewe & Thalheim 2007*b*) through their profile and their intentions. The profile consists of the working profile, the knowledge and abilities profile and of the psychological profile. The context specification allows to characterise the entire environment of the application case. Additionally, the solution may be restricted to certain systems and platforms.

The concept of business use cases (Maciaszek 2001, Robertson & Robertson 2005) generalises the use case concept and reflects cases or case of basic tasks in the reality. Context can be specified using the solution discussed in (Kaschek et al. 2003). The storyboard describes the story space that consists of all possible stories for the given application (Schewe & Thalheim 2005). Portfolio have been formally described in (Schewe & Thalheim 2007*b*) and consist of tasks and the supporting instruments that are necessary. In the sequel we decompose the application cases into business use cases and extract the portfolio. The decomposition must be invertible by a composition operation. This composition operation is the kernel of basic stories that consist of the business use cases and their story flow.

We may condense this application domain description by an abstraction of users into groups depending on their general profile and their main intentions. A group is called *actor*. Actors can be represented by *persona*, e.g. 'Jack-of-all-trade' as a representation of a business man. The persona is characterized by an expressive name, their kind of profession, their purposes and intents, their technical equipment, their behaviour, skills and profile, disabilities , and specific properties such as hobbies and habits. Context and scope my be condensed to environment. We thus might use the elements in Figure 3.

3.2 Description of Application Cases

We may extract application cases from observations in reality, which are very useful source, whenever a WIS is going to be developed from scratch or is required to provide a 'natural' behaviour. In the latter case users are not required to learn the behaviour of the WIS. Instead, the user can continue using a 'classical' behavioural pattern.

The classical principle of the six big W and one H suffices for description of application cases: Who will be using the system? When will the system be used? Where is the information system used? What is represented in the system? Why is the system used? Which problem needs to be solved? How will the system be used?

The W^{6}H principle can be structured into three dimensions: user dimension describing the 'who' and 'why', the flow dimension describing the 'what' and 'how', and the context dimension describing the 'when' and 'where'. The entry dimension is the description of the purpose and the problems that should be solved. So we shall use the formal template:

`Application case:`	⟨application case name⟩
`Purpose:`	⟨purpose description⟩
`Problems:`	⟨list of problems⟩
`Characterisation:`	⟨outcome description⟩
`Activities:`	⟨list of user activities⟩
`Background:`	⟨general characterisation⟩
`Objectives:`	⟨list of objectives⟩
`Application case flow:`	⟨general description⟩
`Milestones:`	⟨graph of milestones⟩
`Content consumed:`	⟨consumed content items⟩
`Content produced:`	⟨produced content items⟩
`Themes:`	⟨class of intents⟩
`Actors:`	⟨list of actors involved⟩
`Characterisation of actors:`	⟨general profile⟩ ⟨ and intension description⟩
`Collaboration among actors:`	⟨general collaboration ⟩ ⟨ description⟩
`Context:`	⟨general context description⟩
`Time:`	⟨temporality limitations⟩
`Place:`	⟨assignment of places⟩
`System:`	⟨general system context⟩

Example 3 (Continuation of examples 1 and 2)
The system examples introduced for illustration of the challenges are based on a number of complex application cases:

- Application cases of parliamentarians are related to their official portfolio, their need in support, their context, their technical environment, and their life circumstances. The purpose of these cases ranges

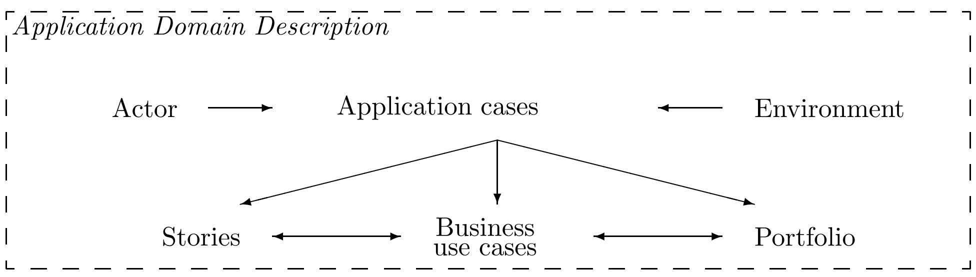

Figure 3: The abstractions of users, profile, intention, context and scope to actors and environment

from session support to support for contributions. Typically, collaborations become very complex.

- The Hallig scout in the Digicult project is gathering (folklore) information based on the content that is currently available, based on the people to be interviewed, based on their environments, evaluating their beliefs and their observations, and condensing the content produced for integration into the Digicult database.
- A typical eGovernment application is the registration support of inhabitants. It includes a number of basic stories such as change of address date, notification of contractors about the change, consideration of social etc. support, special registration for foreigners, extended registration for dependents etc.

3.3 Problem Space Specification

The application cases describe problems for which solutions may be envisioned. The same problem can have several solutions. Therefore, we are interested in a characterisation of problems. This characterisation is given through the problem space. This space characterises the

The problem space specification can be given in a natural language. We prefer a more formal approach whenever this is possible. Consider for instance the example depicted in Figure 1. Which solution is going to be supported depends on the state, on the county or on the city in Germany. The preference of one solution over others is either based on official rules and laws or on habits of the area. Therefore, we must be very flexible in our choices that are mapped to requirements. The same flexibility is also necessary for systems such as the parliamentarian support system SeSAM. The last system is used in the states of Brandenburg and of Saxony and cannot be used in its current form in the state Schleswig-Holstein.

A problem in an application domain may be formally specified by four components. The following frame generalises the approach of (Polya & Polya 1945) or approaches used in AI:

The ***state space*** consists of the collection of all those states that are reachable from the *initial state*. Some of the states are considered to be desirable, i.e. are *goal states*. States can be modelled through languages such as ER. State may have properties such as suitability for certain purposes.

The ***actions*** allow to move from one state to another state under certain conditions. We may assume that the effect of the actions is observable to a certain extent by the user. User may use several actions in parallel. Actions may be blocked or enabled depending on conditions. Actions may be used at some cost.

The ***goal test*** determines whether a given state or state set satisfies the goals. The goal test may be defined through a set of states or through properties. The goal test may also allow to state which quality has the state set for the problem solution.

The ***problem solution controller*** evaluates the actions undertaken by the user. Some solutions may be preferred over other, e.g. have less costs, or are optimal according to some optimality criterion. Controllers can be based on evaluators of the pathes from the initial state to the current state.

We assume typically that states can be observed and distinguished from each other to a certain extent by users.

3.4 The Algebra for Application Cases

The *application case world* $(\mathcal{A}, \mathcal{O}, \mathcal{P})$ consists of

- basic application cases $\mathcal{A}$, and
- algebraic operations $\mathcal{O}$ for computing complex cases such as combination $\boxtimes$ of cases, abstraction $\boxdot$ of cases by projections, quotient $\boxminus$ of cases, renaming ρ of cases, union $\Cup$ of cases, intersection $\Cap$ of cases, full negation $\neg$ of cases, and minimal negation $\rightharpoondown$ of cases within a given context,

  and
- predicates $\mathcal{P}$ stating associations among cases such as the sub-case relation $\preccurlyeq$, a statement $\overset{\exists}{\Cap}$ whether cases can be potentially associated with each other, a statement $\overset{\exists}{\not\Cap}$ whether cases cannot be potentially associated with each other, a statement $\overset{\exists}{\Cup}$ whether cases are potentially compatible with each other, and a statement $\overset{\exists}{\not\Cup}$ whether cases are incompatible with each other.

We require that the sub-case relation is not transitively reflexive. The compatibility and incompatability predicate are not contradicting. The potential association and its negation must not conflict.

We may use expressions defined by the operations and derived predicates:

The predicate $\curlyvee := \overset{\exists}{\Cap} \wedge \overset{\exists}{\not\Cup}$ is used for diverging cases.

The predicate $\between := \overset{\exists}{\not\Cap} \wedge \overset{\exists}{\not\Cup}$ is used for isolated from each other cases.

The predicate $\curlywedge := \overset{\exists}{\Cap} \wedge \overset{\exists}{\Cup} \wedge \not\succcurlyeq \wedge \not\preccurlyeq$, and is used for homogenizable cases.

The predicate $\overset{\exists}{\oplus} := \overset{\exists}{\not\Cap} \wedge \overset{\exists}{\Cup}$ is used for heterogeneous cases.

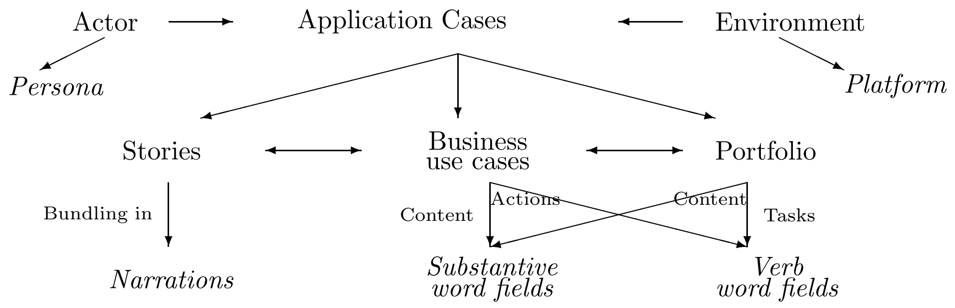

Figure 4: The natural language representation of application domain elements

Application cases are either basic or constructed cases. We use a number of representations for visualisation of the application cases within the application domain:

Application case maps are graphs consisting of nodes representing application cases and edges representing the sub-case relation and the combination of cases to complex cases.

Application case hierarchies are forests representing cases and their sub-case relation.

Full application case maps are graphs consisting of nodes representing application cases and edges representing the sub-case relation and the combination, abstraction, union, intersection, quotient, full negation and minimum negation of cases if these cases are used within the application.

These operations form the basis for the mapping of application cases to business use cases, to stories and to portfolio. The extraction of business use case from application cases is supported by five operations that are easing the task of efficiently acquiring relevant requirement prescriptions and of documenting them:

1. Application case *projection* narrows or scopes the application case to those parts (entities or concepts, axioms or invariants relating entities, functions, events, and behaviours) that are of concern for the business use cases.

2. Application case *instantiation* lifts the general cases to those that are of interest within the solution and instantiates variables by values which are fixed for the given system.

3. Application case *determination* is used for selecting those traces or solutions to the problem under inspection that are the most perspective or best fitting for the system envisioned. The determination typically results in a small number of scenarios for the application cases that are going to be supported.

4. Application case *extension* is used for adding those facets that are not given by the application case but are given by the environment or by the platforms which might be chosen or that might be used for simplification or support of the application case (e.g., additional data, auxiliary functionality).

5. Application case are often associated, adjacent, interact or fit with each other. Application case *join* is used to combine application cases into more complex and combined cases that describe a complex solution.

Business use cases prescribe the "operating" part of the application cases for actors and the supporting system. Therefore, we determine which part of the application cases is going to be left out, which parts are emulated, and which solution is going to be preferred. The stories combine these business use cases to flows of activities. The combination is determined though the application of the five operations.

The application of these operations also allows to extract which subcases, which functionality, which events and which behaviour is *shared* among the business use cases. These shared facilities provide crosscutting concerns among all business use cases and the exchange activities or scenes in the stories. They also hint on possible architectures of information systems and on separation into candidate components. For instance, entity sharing describe which information flow and development can be observed in the application. Functionality may be either shared by several components or may be a part of the user interface.

The profiles are specifying tasks and obligations or permissions of users of the system. These profiles can be extracted from the application cases by application of these operations. The interaction between the users and the system can directly be mapped to message or life sequence charts. Additionally, we can extract quality properties (Jaakkola & Thalheim 2005) such as faithfulness, didactic behaviour, pedagogic surveyability, physiological widgets, psychological behaviour and user-friendliness for the *human-computer interface*.

This information is later enhanced by computer requirements such as performance, dependability, maintenance, platform, and documentation requirements.

4 Transforming Application Cases to Requirements

4.1 Mapping Application Cases to Stories, Portfolio and Business Use Cases

Application domain engineering is based on a tight collaboration with the customer, with the user, and with the owners and planers. We may use diagram techniques at that level. In this case we assume that partners in the development process are well informed and well educated. Another representation technique is the sophisticated use of natural language utterances. The sophistication is based on a theory of word fields (called capsules in (Berztiss 2001)). Word fields consist of syntactical and semantical representations of words similar to approaches developed for WordNet (WordNet 2007). Additionally a pragmatical dimension can be added to word fields. *Verb word fields* can be classified into 10 word categories (Kunze 1992). A similar classification has been developed for *substantive word fields* in the RADD project (Albrecht, Altus, Buchholz, Cyriaks, Düsterhöft, Lewerenz, Mehlan, Steeg, Schewe & Thalheim 1998). These theories allow to use substan-

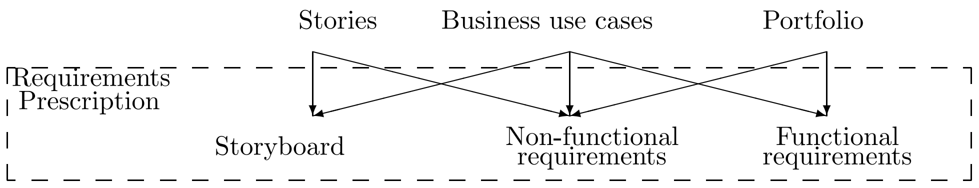

Figure 5: The use of application domain information for requirements elicitation and analysis

tive and verb word fields for the verbal description of the application. We thus support the verbal descriptions for description of application cases. We also support *narrations* since verbal description are often given as real-life application *stories* describing real application cases.

The mapping is based on the first three phases of the C3S3P framework(Krogstie & Jorgensen 2004): concept study, scaffolding, scoping, solution modelling, platform integration, piloting in real projects and performance monitoring and management. Concept study uses word fields and extracts the concept. The scaffolding phase is used for analysis of the current solution to the application problem and for envisioning other solutions. Scoping focuses on creation of executable pilots supporting the application case. This knowledge is used for identification and consolidation of requirements for structuring and functionality of the (information) system, for platform plans, methodology choice and sketching these requirements.

Word fields allow to derive generic functions and generic content (Bienemann, Schewe & Thalheim 2006). These functions and content provide the basis for a specification of tasks and business use case (Robertson & Robertson 2005). Tasks are the main element of *portfolio*. The action components of business use cases are represented by verbs and the content component is characterised by substantives. Therefore business use cases and portfolio are templates that are used for representation of chunks of word fields. Finally, the environment is often given through a hint to *platforms* intended or requested.

4.2 Mapping Application Cases to Requirements

The application domain description can be used for requirements gathering. Application cases describe all parts of the application domain that are relevant for the application. The application cases are subject to peer reviews. It starts with a feasibility and completeness study. Next we check whether all activities are represented. Additionally, we check whether application cases are truly generic in the sense that they do not go into detail that may apply to the present application alone. Finally, we check whether there are any special cases additional to those that have already listed in the application cases. Applying this check results in a set of tasks that are integrated into the application portfolio, in a set of business use cases and in a set of stories.

We can now map the three results of application domain description to properties typically used for requirements prescription. Software engineering has divided properties into functional and non-functional properties, restrictions and pseudo-properties. This separation can be understood as a separation into essential properties and non-essential ones. Functional and non-functional properties require prescription of main data structures, of the main functionality, of control facilities, and of collaboration of components of a system envisioned.

If we separate the information system from the presentation system then this separation leads to a far more natural separation into information system requirements and presentation systems requirements. The system perspective considers properties such as performance, efficiency, maintainability, portability, and other classical functional requirements. Typical presentation system requirements are usability, reliability, and requirements oriented to high quality in use, e.g., effectiveness, productivity, safety, privacy, and satisfaction. Safety and security are also considered to be restrictions since they specify undesired behaviour of systems. Pseudo-properties are concerned with technological decisions such as language, middleware, operating system or are imposed by the user environment, the channel to be used, or the variety of client systems.

Figure 5 depicts the mappings of elements of the application domain description to elements used in requirements engineering. The storyboard can be understood as a step-wise or scene-wise prescription of how a particular actor interacts with the system.

As requirement acquisition and elicitation progresses there is likely a shift in the preferred representation from textual and formal to graphical. A number of approaches have been developed for graphical representation such as storyboarding approaches (Schewe & Thalheim 2005), quality-driven design methodologies (Jaakkola & Thalheim 2005), and UML diagrams (use case, class, sequence, statechart, and activity diagrams). The storyboard is going to be mapped to a number of activity diagrams.

5 Conclusion

This paper enhances application domain modelling by explicit consideration of application cases. These application cases can be mapped to requirements. The extraction of requirements is based on the C3S3P framework. The approach reported has intentionally been applied in about half dozen of our website development projects. We realised that requirements engineering is to system-focused and that we need a formal representation of the elements at the strategic layer. Our description of this layer led to the development of a theory of application domain description, to a number of formal templates that can be used for description, and to mappings to the elements of the requirements prescription layer. This paper summarises those ideas.

Application cases are typically already based on proposals for solutions or on a number of solutions for which one of them is going to be preferred at the moment. For instance, eGovernment application cases must correspond to the expectations of users and their needs within life situations. These life situations might be supported by application cases. Due to the variety of users, to the variety of utilisation of the system, to the variety of real usage, we extend the application domain description by life cases (Schewe & Thalheim 2007*a*). Life cases reflect the life situations in the complexity of everyday life. Life cases are going to be decomposed, segmented and mapped to application cases.

References

Albrecht, M., Altus, M., Buchholz, E., Cyriaks, H., Düsterhöft, A., Lewerenz, J., Mehlan, H., Steeg, M., Schewe, K.-D. & Thalheim, B. (1998), ‘RADD - Rapid application and database development. Readings - Main papers published in the RADD project’, CAU Kiel, Department of Computer Science, http://www.is.informatik.uni-kiel.de/∼thalheim/indeeerm.htm.

Anh, D. N. & Moore, R. (1996), Formal modeling of large domains, *in* ‘APSEC’, IEEE Computer Society, pp. 246–.

Berztiss, A. (2001), Requirements engineering, *in* ‘Handbook of Software Engineering and Knowledge Engineering’, Vol. I, World Scientific Pub. Co., pp. 59–70.

Bienemann, A., Schewe, K.-D. & Thalheim, B. (2006), Towards a theory of genericity based on government and binding, *in* ‘Proc. ER’06, LNCS 4215’, Springer, pp. 311–324.

Bjørner, D. (2006), *Software Engineering 3: Domains, requirements, and software design*, Springer, Berlin.

Boehm, B. (2006), A view of 20th and 21st century software engineering, *in* ‘Proc. ICSE’06’, ACM Press, pp. 12–29.

Darimont, R. & van Lamsweerde, A. (1996), Formal refinement patterns for goal-driven requirements elaboration, *in* ‘SIGSOFT FSE’, pp. 179–190.

Epner, M. (2000), Poor project management number-one problem of outsourced e-projects, Research briefs, Cutter Consortium.

Gunter, C. A., Gunter, E. L., Jackson, M. & Zave, P. (2000), ‘A reference model for requirements and specifications’, *IEEE Software* **17**(3), 37–43.

Hansen, M. R. & Hung, D. V. (2007), A theory of duration calculus with application, *in* ‘Domain Modeling and the Duration Calculus’, pp. 119–176.

Jaakkola, H. & Thalheim, B. (2005), Software quality and life cycles, *in* ‘ADBIS’05’, Springer, Tallinn, pp. 208– 220.

Jackson, E. K. & Sztipanovits, J. (2006), Towards a formal foundation for domain specific modeling languages, *in* ‘EMSOFT’, ACM, pp. 53–62.

Kaschek, R., Schewe, K.-D., Thalheim, B. & Zhang, L. (2003), Integrating context in conceptual modelling for web information systems, web services, e-business, and the semantic web, *in* ‘WES 2003’, LNCS 3095, Springer, pp. 77–88.

Krogstie, J. & Jorgensen, H. (2004), Interactive models for supporting networked organisations, *in* ‘CAiSE’04’, LNCS, Springer, Berlin, pp. 1–14.

Kunze, J. (1992), Generating verb fields, *in* ‘Proc. KONVENS’, Informatik Aktuell, Springer, pp. 268–277. in German.

Lambrix, P. & Padgham, L. (1996), A description logic for composite objects for domain modeling in an agent-oriented application, *in* ‘Description Logics’, Vol. WS-96-05 of *AAAI Technical Report*, AAAI Press, pp. 146–149.

Lewerenz, J. (2000), Human-computer interaction in heterogeneous and dynamic environments: A framework for its conceptual modelling and automatic customization, PhD thesis, Brandenburg University of Technology at Cottbus, Faculty Mathematics, Natural Sciences and Computer Science.

Lowe, D. (2003), ‘Web system requirements: an overview’, *Requirements Engineering* **8**(2), 102–113.

Maciaszek, L. (2001), *Requirements analysis and design*, Addison-Wesley, Harlow, Essex.

Missikoff, M. & Schiappelli, F. (2005), A method for ontology modeling in the business domain, *in* ‘EMOI-INTEROP’, CEUR Workshop Proceedings, CEUR-WS.org.

O’Neill, E. & Johnson, P. (2004), Participatory task modelling: users and developers modelling users’ tasks and domains, *in* ‘TAMODIA’, ACM, pp. 67–74.

Polya, G. & Polya, G. (1945), *How to solve it: A new aspect of mathematical method*, Princeton University Press, Princeton.

Robertson, S. & Robertson, J. (2005), *Requirements-led project management*, Pearson, Boston.

Rolland, C. (2006), From conceptual modeling to requirements engineering, *in* ‘Proc. ER’06’, LNCS 4215, Springer, Berlin, pp. 5–11.

Schewe, K.-D. & Thalheim, B. (2005), ‘Conceptual modelling of web information systems’, *Data and Knowledge Engineering* **54**, 147–188.

Schewe, K.-D. & Thalheim, B. (2007*a*), Life cases: An approach to address pragmatics in the design of web information systems, *in* J. Filipe, J. Cordeiro, B. Encarnacao & V. Pedrosa, eds, ‘Proc. WebIST’, Vol. II (WIA), pp. 5–12.

Schewe, K.-D. & Thalheim, B. (2007*b*), ‘Pragmatics of storyboarding for web information systems: Usage analysis’, *Int. Journal Web and Grid Services* **3(2)**, 128–169.

Schewe, K.-D., Thalheim, B. & Tretjakow, A. (2006), Formalization of user preferences, obligations, and rights, *in* R. Kaschek, ed., ‘Perspectives of Intelligent Systems Assistents, PISA’05’, IDEA.

Thalheim, B. (2000*a*), *Entity-relationship modeling – Foundations of database technology*, Springer, Berlin.

Thalheim, B. (2000*b*), ‘Readings in fundamentals of interaction in information systems’, Reprint, BTU-Cottbus, accessible through http://www.is.informatik.uni-kiel.de/∼thalheim, Collection of papers by C. Binder, W. Clauß, A. Düsterhöft, T. Feyer, T. Gutacker, B. Heinze, J. Lewerenz, M. Roll, B. Schewe, K.-D. Schewe, K. Seelig, S. Srinivasa, B. Thalheim.

Wieringa, R. (2003), *Design methods for reactive systems: Yourdan, Statemate, and the UML*, Morgan Kaufmann, Amsterdam.

WordNet (2007), ‘Antonym finder and synonym thesaurus – Synonyms and definitions for english words’, http://www.synonym.com/.