INSTITUT FÜR INFORMATIK

UND PRAKTISCHE MATHEMATIK

Web Information Systems: Usage, Content, and Functionality Modelling

Klaus-Dieter Schewe, Bernhard Thalheim

Bericht Nr. 0405
April 2004

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Institut für Informatik und Praktische Mathematik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Web Information Systems: Usage, Content, and Functionality Modelling

Klaus-Dieter Schewe, Bernhard Thalheim

Bericht Nr. 0405
April 2004

e-mail: k.d.schewe@massey.ac.nz ,
thalheim@is.informatik.uni-kiel.de

Abstract

The design of large-scale data-intensive web information systems (WIS) requires a clear picture of the intended users and their behaviour in using the system, a support of various access channels and the technology used with them, and an integration of traditional methods for the design of data-intensive information systems with new methods that address the challenges arising from the web-presentation and the open access.

This paper presents the conceptual modelling parts of a methodology for the design of WISs that is based on an abstract abstraction layer model (ALM). It concentrates on the two most important layers in this model: a business layer and a conceptual layer.

The major activities on the business layer deal with user profiling and storyboarding, which addresses the design of an underlying application story. The core of such a story can be expressed by a directed multi-graph, in which the vertices represent scenes and the edges actions by the users including navigation. This leads to story algebras which can then be used to personalise the WIS to the needs of a user with a particular profile.

The major activities on the conceptual layer address the support of scenes by modelling media types, which combine links to databases via extended views with the generation of navigation structures, operations supporting the activities in the storyboard, hierarchical presentations, and adaptivity to users, end-devices and channels. Adding presentation style options this can be used to generate the web-pages that will be presented to the WIS users.

Table of Contents

Chapter 1

Introduction

A *web information system* (WIS) is a database-backed information system that is realized and distributed over the web with user access via web browsers. Information is made available via pages including a navigation structure between them and to sites outside the system. Furthermore, there should also be operations to retrieve data from the system or to update the underlying database(s).

Various approaches to develop design methods for WISs have been proposed so far. Most of them such as the ARANEUS framework [4], the OOHDM framework [37] and the WebML work in [10] focus on a problem triplet consisting of content, navigation and presentation. This leads to modelling databases, hypertext structures and page layout. In addition, the personalisation of a WIS to the particular needs of users is still a hot research topic. We will give a more detailed overview on related work in Section 2.

Our own work in [41] emphasises a methodology oriented at abstraction layers and the co-design of structure, operations and interfaces. As WISs are open systems in the sense that everyone who has access to the web may turn up as a user, their design requires a clear picture of the intended users and their behaviour. This includes knowledge about the used access channels and end-devices. At a high level of abstraction this first leads to *storyboarding*, an activity that addresses the design of an underlying application story [17, 27]. As soon as WISs become large, it becomes decisive that such an underlying application story is well designed. Furtheron, the scenes in the stories have to be adequately supported. For this our work focusses on the integration of traditional methods for the design of data-intensive information systems with new methods addressing the challenges arising from the web-presentation and the open access. This leads to *media types*, which cover extended views, adaptivity, hierarchies and presentation style options. In Section 2 we will also give a more detailed account on our previous work. Furthermore, in Section 3 we give a brief overview of the abstraction layer model (ALM) that underlies our work.

In this article we concentrate on the two most important abstraction layers in our methodology: the business layer and the conceptual layer. As already stated the major activities on the business layer deal with user profiling and storyboarding. The core of the story space can be expressed by a directed multi-graph, in which the vertices represent scenes and the edges actions by the users including navigation. If more details are added, application stories can be expressed by some form of process algebra. That is, we need atomic activities and constructors for sequencing, parallelism, choice, iteration, etc. to write stories. The language `SiteLang` [14] is in fact such a process algebra for the purpose of storyboarding. In addition to the mentioned constructors it emphasises the modelling of *scenes*, which can be expressed by indexing the atomic activities and using additional constructs for entering and leaving scenes.

In Section 4 we first present the fundamentals of storyboarding. Then we show that such WIS-oriented process algebras lead to many-sorted Kleene algebras with tests, where the sorts correspond to scenes in the story space. Kleene algebras (KAs) have been introduced in [28] and extended to Kleene algebras with tests (KATs) in [30]. In a nutshell, a KA is an algebra of regular expressions, but there are many different interpretations other than just regular sets. A KAT imposes an additional structure of a Boolean algebra on a subset of the carrier set of a Kleene algebra. For both KAs and KATs interesting decidability and completeness results hold [29, 32].

If we ignore assignments, KATs can be used to model abstract programs. Doing this, it has been shown in [31] that KATs subsume propositional Hoare logic [24]. This subsumption is even strict, as the theory of KATs is complete, whereas propositional Hoare logic is not. Therefore, we consider KATs an ideal candidate for reasoning about the story space. In this article we apply KATs to the *personalisation* of the story space to the preferences and information needs of users. We will use the on-line loan application example from [6, 39] to illustrate this application in the practically relevant area of electronic banking.

In Section 5 we present *media types* extending the work in [15, 42]. The emphasis is on the data processed in the storyboard and the actions on these data in order to describe content and functionality in a formal way, and to develop database support. In this sense the modelling task is quite similar to the task for dialogue-based information systems, for which the theory of *dialogue objects* was developed in [40].

That is, we have to group suitable data and specify operations for the actions and restructure the data in order to define connections to databases. As database design follows different objectives (no redundancy, optimised access), the content specification should lead to views [47]. We will extend the approach to query languages in [38] for this purpose. Furthermore, we extend media types in a way that they can be tailored to different users, different end-devices and different communication channels without designing multiple sites, and finally link the media types with the storyboard and ensure that all scenarios are adequately supported. The result will be a *media schema*, i.e. a collection of media types, which adequately represent the storyboard.

Chapter 2

Related and Previous Work

The number of publications on the topic of WIS design is enormous. Some of them are on isolated topics in the area, e.g. the work in [20] on application in e-commerce. Another example is the work in [23] which applies formal specification techniques. The work in [22] presents general guidelines and studies web site quality in the area of e-commerce, thus is more analytical than on conceptual modelling. The same applies to the very valuable design patterns in [49]. The work in [34] emphasises a distinction between static and dynamic WIS. While most work on WIS modelling only addresses the static aspects, the authors favour an approach that takes also operations into account.

The ARANEUS framework in [4] emphasises that conceptual modelling of web information systems should approach a problem triplet consisting of content, navigation and presentation. This leads to modelling databases, hypertext structures and page layout. However, ARANEUS originally does not explicitly separate between the database content and the data presented by the WIS, i.e. the aspect of dealing with views is neglected. The navigation is not treated as an integrated part of such views, but more as an "add-on" to the content specification, and besides navigation no further operations that could cover the functionality of the system are handled. In [35] it has been investigated, how ARANEUS could be supported by XML. Other authors have referred to the ARANEUS framework and extended it. The work in [5] addresses the integrated design of hypermedia and operations, but remains on a very informal level. Similarly, the work in [9] presents a web modelling language WebML and starts to discuss personalisation of web information systems and adaptivity, but again is very informal. The work in [33] also picks up the issue of views and describes a MVC framework (model-view-controller) dealing with this. In particular, the authors define an extension of UML to cope with this aspect.

OOHDM [36, 37] is quite similar to ARANEUS. Differences are that its origins are not in the area of databases, but in hypertext [44, 43], and that it explicitly refers to an object oriented approach. OOHDM emphasises an object layer, hypermedia components, i.e. links, and an interface layer. The interface layer refers to the MVC triplet. The work in [21] also starts from hypertext design. The work introduces "authoring in the large", i.e., the conceptual modelling of information elements and navigation, and uses this to categorise different types of links.

Another similar approach was presented in [12, 11, 19, 18], which by now is summarised in [10]. The work emphasises a multi-level architecture for the data-driven generation of WIS, thus takes the view aspect into account. It also addresses the personalisation of WIS by providing user-dependent views, thus is aware of the problem of adaptivity. Furthermore, it emphasises structures, derivation and composition, i.e. views, navigation and presentation, thus addresses the same problem triplet as the ARANEUS framework.

A completely different direction of research is the work in [45, 46] on contextual information systems (CIS), which provides a formal framework for the specification of contexts. One important use of contexts is to help WIS users not to lose track, while navigating through the WIS. The work in [3] describes a prototypical implementation of a CIS.

Our own work on storyboarding has been reported in [16, 17], and lead to the definition of the storyboarding language `SiteLang` in [14]. The role and use of metaphors in storyboarding was investigated in [48]. The work in [7, 39] applies storyboarding to electronic business. An application to electronic learning was presented in [8]. The work in [25] reverses the use of `SiteLang` and investigates stories in existing WIS. Media types were introduced in [15] and coupled with the specification of contexts in [26].

Chapter 3

Abstraction Layers in WIS Modelling and Design

Our methodology is based on an *Abstraction Layer Model* for web-based systems, which is illustrated in Figure 3.1. The general ideas of this model is to identify and use several layers of abstraction.

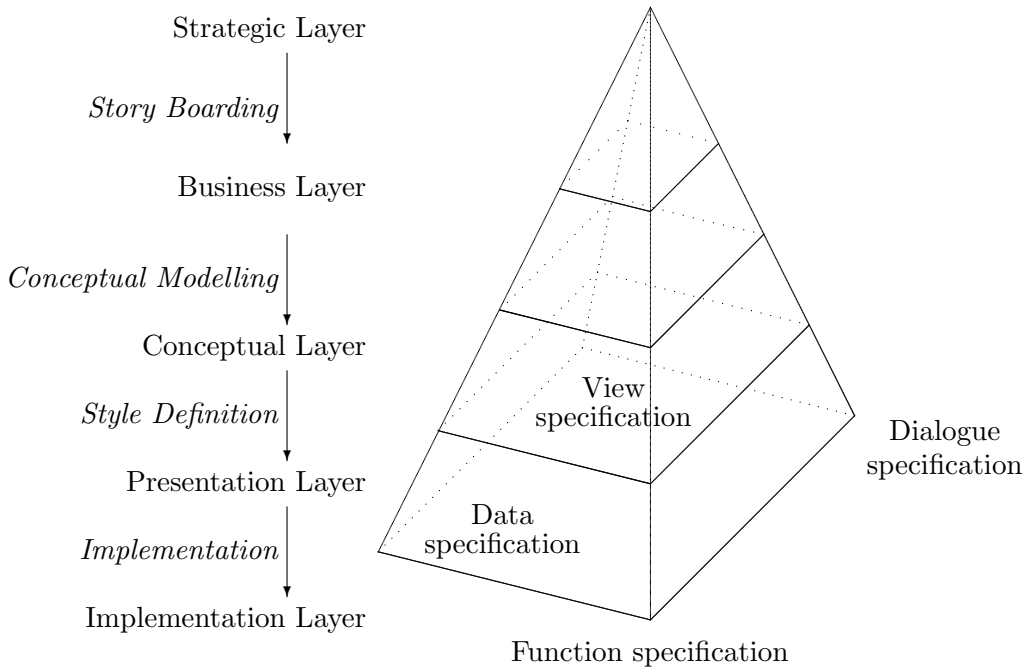

Fig. 3.1. Abstraction Layers in Web-based Information Systems

The top layer is called the *strategic layer*. It is used to describe the system in a general way: What are the intentions? Who are the expected users? This reflects the request from [13] to start WIS development with a "mission statement".

The next lower layer is called the *business layer*, which is used to concretise the ideas gathered on the strategic layer. This means to get a clearer picture of the different kinds of users and their profiles. This may also include the different roles of users and tasks associated with these roles. The major part of this layer, however, deals with the description of the storyboard. Stories identify possible paths through the system and the information that is requested to

enable such paths. So the general purpose of the business layer is to anticipate the behaviour of the system's users in order to set up the system in a way that supports the users as much as possible.

The central layer is the *conceptual layer*. The various scenes appearing in the story board have to be analysed and integrated, so that each scene can be supported by a unit combining content and functionality. This will lead to designing abstract media types. The information content of the media types must be combined to design the structure of an underlying database.

The next lower layer is the *presentation layer* which is devoted to the problem of associating presentation options to the media types. This can be seen as a step towards implementating the system. Finally, the lowest layer is the *implementation layer*. All the aspects of the physical implementation have to addressed on this layer. This includes setting up the logical and physical database schemata, the page layout, the realisation of functionality using scripting languages, etc. As far as possible, components on the implementation layer, especially web-pages, should be generated from the description on the higher layers.

Furthermore, on each layer except the strategic layer we identify two dimensions for the description of the system: *focus* and *modus*. The focus dimension distinguishes between local and global components; the modus dimension distinguishes between static and dynamic components. As this leads to four combinations, we distinguish between the following components:

- global and static components, which are addressed by a *data specification*,
- global and dynamic components, which are addressed by a *function specification*,
- local and static components, which are addressed by a *view specification*, and
- local and dynamic components, which are addressed by a *dialogue specification*.

Finally, each layer is associated with layer specific modelling tasks. The transition from the strategic to the business layer is associated with the activities of story boarding and user profiling. The transition from the business layer to the conceptual layer is associated with conceptual modelling, which addresses database modelling, operations modelling, view modelling, and media type modelling. The transition to the presentation layer is associated with the definition of presentation styles. Finally, the transition to the implementation layer is associated with all implementation tasks.

The fact that these layers exist in the model and that the methodology is based on transitions between these layers does not imply that a development project must first finish the work associated with one layer before it can proceed to the next lower one.

Chapter 4

Usage Modelling with Storyboarding

As WISs are open in the sense that anyone who has access to the web could become a user, the design of such systems requires some anticipation of the users' behaviour. Storyboarding addresses this problem. Thus, a *storyboard* will describe the ways users may choose to interact with the system.

A storyboard consists of three parts. The first part describes the paths users follow while navigating through the WIS. These paths are the *stories*, and their integrated description will be called the *story space*. The second part describes the *actors*, i.e. groups of users with the same profile. We will outline a rather simple way of modelling *user profiles* based on an analysis of the various dimensions of such profiles. The third part are *tasks*, which link the activities of the actors with the story space.

4.1 Story Spaces

On a high level of abstraction we may think of a WIS as a set of abstract locations, which abstract from actual pages. A user navigates between these locations, and on this navigation path s/he executes a number of actions. We regard a location together with local actions, i.e. actions that do not change the location, as a unit called *scene*.

Then a WIS can be decribed by a edge-labelled directed multi-graph, in which the vertices represent the scenes, and the edges represent transitions between scenes. Each such transition may be labelled by an action executed by the user. If such a label is missing, the transition is due to a simple navigation link. The whole multi-graph is then called the *story space*.

4.1.1 Scenario Modelling

Roughly speaking, a *story* is a path in the story space. It tells what a user of a particular type might do with the system.

The combination of different stories to a subgraph of the story space can be used to describe a "typical" use of the WIS for a particular task. Therefore, we call such a subgraph a *scenario*. Usually storyboarding starts with modelling scenarios instead of stories, coupled by the integration of scenarios to the story space.

At a finer level of details we may add a triggering *event*, a *precondition* and a *postcondition* to each action, i.e. we specify exactly, under which conditions an action can be executed and which effects it will have. Further extensions to scenes such as adaptivity, presentation, tasks and roles have been discussed in [6] and [15], but these extensions are not yet relevant here.

Looking at scenarios or the whole story space from a different angle, we may concentrate on the flow of actions:

- For the purpose of storyboarding actions can be treated as being atomic, i.e. we are not yet interested in how an underlying database might be updated. Then each action also belongs to a uniquely determined scene.
- Actions have pre- and postconditions, so we can use annotations to express conditions that must hold before or after an action is executed.
- Actions can be executed sequentially or parallel, and we must allow (demonic) choice between actions.
- Actions can be iterated.
- By adding an action `skip` we can then also express optionality and iteration with at least one execution.

These possibilities to combine actions lead to operators of an algebra, which we will call a *story algebra*. Thus, we can describe a story space by an element of a suitable story algebra. We should, however, note already that story algebras have to be defined as being many-sorted in order to capture the association of actions with scenes.

4.1.2 Story Algebras

Let us take now a closer look at the storyboarding language `SiteLang` [14], which in fact defines a story algebra. So, let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be a set of scenes, and let $\mathcal{A} = \{\alpha_1, \ldots, \alpha_k\}$ be a set of (atomic) actions. Furthermore, assume a mapping $\sigma : \mathcal{A} \to \mathcal{S}$, i.e. with each action $\alpha \in \mathcal{A}$ we associate a scene $\sigma(\alpha)$.

This can be used to define inductively the set of *processes* $\mathcal{P} = \mathcal{P}(\mathcal{A}, \mathcal{S})$ determined by $\mathcal{A}$ and $\mathcal{S}$. Furthermore, we can extend σ to a partial mapping $\mathcal{P} \to \mathcal{S}$ as follows:

- Each action $\alpha \in \mathcal{A}$ is also a process, i.e. $\alpha \in \mathcal{P}$, and the associated scene $\sigma(\alpha)$ is already given.
- `skip` is a process, for which $\sigma(\texttt{skip})$ is undefined.
- If p_1 and p_2 are processes, then also the *sequence* $p_1; p_2$ is a process. Furthermore, if $\sigma(p_1) = \sigma(p_2) = s$ or one of the p_i is `skip`, then $\sigma(p_1; p_2)$ is also defined and equals s, otherwise it is undefined.
- If p_1 and p_2 are processes, then also the *parallel process* $p_1 \| p_2$ is a process. Furthermore, if $\sigma(p_1) = \sigma(p_2) = s$ or one of the p_i is `skip`, then $\sigma(p_1 \| p_2)$ is also defined and equals s, otherwise it is undefined.
- If p_1 and p_2 are processes, then also the *choice* $p_1 \Box p_2$ is a process. Furthermore, if $\sigma(p_1) = \sigma(p_2) = s$ or one of the p_i is `skip`, then $\sigma(p_1 \Box p_2)$ is also defined and equals s, otherwise it is undefined.
- If p is a process, then also the *iteration* p^* is a process with $\sigma(p^*) = \sigma(p)$, if $\sigma(p)$ is defined.
- If p is a process and φ is a boolean condition, then the *guarded process* $\{\varphi\}p$ and the *post-guarded process* $p\{\varphi\}$ are processes with $\sigma(\{\varphi\}p) = \sigma(p\{\varphi\}) = \sigma(p)$, if $\sigma(p)$ is defined.

Doing this we have to assume tacitly that navigation between scenes is also represented by an activity in $\mathcal{A}$, and the assigned scene is the origin of the navigation. `SiteLang` provides some few more constructs, which we have omitted here. Constructs such as non-empty iteration p^+ and optionality $[p]$ can be expressed by the constructs above, as we have $p^+ = p; p^*$ and $[p] = p \Box \texttt{skip}$.

Furthermore, we deviated from the `SiteLang` syntax used in [14] and [6]. For instance, `SiteLang` provides constructors $\nearrow$ and $\searrow$ to enter or leave a scene, respectively. We simply use parentheses and make the associated scene explicit in the definition of σ.

Parallel execution is denoted by $\lceil\!\rceil$ and choice by $\lceil\!\oplus\!\rceil$ in `SiteLang`, whereas here we use the more traditional notation.

`SiteLang` also uses $\lceil\!\varphi\!\rceil$ to mark a parallel execution with a synchronisation condition φ, which in our language here can be expressed by a post-guarded parallel process $(\ldots \| \ldots)\{\varphi\}$.

Example 4.1. Consider the loan application from [39]. A rough sketch of the story space can be described as follows:

enter_loan_system ;
(({φ_0} look_at_loans_at_a_glance □
({φ_1} request_home_loan_details ;
(look_at_home_loan_samples □ `skip`) {φ_3}) □
({φ_2} request_mortgage_details ;
(look_at_mortgage_samples □ `skip`) {φ_4}))* {φ_5}) ;
(select_home_loan {φ_6} □ select_mortgage {φ_7}) ;
(({φ_6} (provide_applicant_details ;
(provide_applicant_details □ `skip`) ;
(describe_loan_purpose ‖ enter_amount_requested ‖
enter_income_details) ;
select_hl_terms_and_conditions) {φ_8}) □
({φ_7} (provide_applicant_details ; provide_applicant_details* ;
(describe_object ‖ enter_mortgage_amount ‖
describe_securities*) ;
(enter_income_details ‖ enter_obligations*) ;
(({$\neg\varphi_{12}$} select_m_terms_and_conditions ;
calculate_payments)* ;
{φ_{12}} select_m_terms_and_conditions)) {φ_9})) ;
confirm_application {$\varphi_{10} \vee \varphi_{11}$}

involving the conditions

$\varphi_0 \equiv$ information_about_loan_types_needed
$\varphi_1 \equiv$ information_about_home_loans_needed
$\varphi_2 \equiv$ information_about_mortgages_needed
$\varphi_8 \equiv$ home_loan_application_completed
$\varphi_9 \equiv$ mortgage_application_completed
$\varphi_{10} \equiv$ applied_for_home_loan
$\varphi_{12} \equiv$ payment_options_clear
$\varphi_3 \equiv$ home_loans_known
$\varphi_4 \equiv$ mortgages_known
$\varphi_5 \equiv$ available_loans_known
$\varphi_6 \equiv$ home_loan_selected
$\varphi_7 \equiv$ mortgage_selected
$\varphi_{11} \equiv$ applied_for_mortgage

The set of scenes is $\mathcal{S} = \{s_1, \ldots, s_{14}\}$ with

$s_1 =$ type_of_loan $\quad s_2 =$ applicant_details $\quad s_3 =$ home_loan_details
$s_4 =$ home_loan_budget $\quad s_5 =$ mortgage_details $\quad s_6 =$ securities_details
$s_7 =$ mortgage_budget $\quad s_8 =$ confirmation $\quad s_9 =$ income
$s_{10} =$ loan_overview $\quad s_{11} =$ home_loan $\quad s_{12} =$ home_loan_samples
$s_{13} =$ mortgage $\quad s_{14} =$ mortgage_samples

The set of actions is $\mathcal{A} = \{\alpha_1, \ldots, \alpha_{20}\}$ using

$$\alpha_1 = \text{enter\_loan\_system} \quad \alpha_2 = \text{look\_at\_loans\_at\_a\_glance}$$
$$\alpha_3 = \text{request\_home\_loan\_details} \quad \alpha_4 = \text{request\_mortgage\_details}$$
$$\alpha_5 = \text{look\_at\_home\_loan\_samples} \quad \alpha_6 = \text{look\_at\_mortgage\_samples}$$
$$\alpha_7 = \text{select\_home\_loan} \quad \alpha_8 = \text{provide\_applicant\_details}$$
$$\alpha_9 = \text{describe\_loan\_purpose} \quad \alpha_{10} = \text{enter\_amount\_requested}$$
$$\alpha_{11} = \text{enter\_income\_details} \quad \alpha_{12} = \text{select\_hl\_terms\_and\_conditions}$$
$$\alpha_{13} = \text{select\_mortgage} \quad \alpha_{14} = \text{describe\_object}$$
$$\alpha_{15} = \text{enter\_mortgage\_amount} \quad \alpha_{16} = \text{describe\_securities}$$
$$\alpha_{17} = \text{enter\_obligations} \quad \alpha_{18} = \text{select\_m\_terms\_and\_conditions}$$
$$\alpha_{19} = \text{calculate\_payments} \quad \alpha_{20} = \text{confirm\_application}$$

Finally, we get the scene assignment σ with

$$\sigma(\alpha_1) = s_1 \quad \sigma(\alpha_2) = s_{10} \quad \sigma(\alpha_3) = s_{11} \quad \sigma(\alpha_4) = s_{13} \quad \sigma(\alpha_5) = s_{12}$$
$$\sigma(\alpha_6) = s_{14} \quad \sigma(\alpha_7) = s_1 \quad \sigma(\alpha_8) = s_2 \quad \sigma(\alpha_9) = s_3 \quad \sigma(\alpha_{10}) = s_3$$
$$\sigma(\alpha_{11}) = s_9 \quad \sigma(\alpha_{12}) = s_4 \quad \sigma(\alpha_{13}) = s_1 \quad \sigma(\alpha_{14}) = s_5 \quad \sigma(\alpha_{15}) = s_5$$
$$\sigma(\alpha_{16}) = s_6 \quad \sigma(\alpha_{17}) = s_7 \quad \sigma(\alpha_{18}) = s_7 \quad \sigma(\alpha_{19}) = s_7 \quad \sigma(\alpha_{20}) = s_8$$

4.1.3 Mathematical Foundations of Story Algebras

Abstracting from regular expressions we obtain the notion of a Kleene algebra as follows.

Definition 4.1. A *Kleene algebra* (KA) $\mathcal{K}$ consists of

- a carrier-set K containing at least two different elements 0 and 1, and
- a unary operation * and two binary operations $+$ and $\cdot$ on K

such that the following axioms are satisfied:

- $+$ and $\cdot$ are associative, i.e. for all $p, q, r \in K$ we must have $p + (q + r) = (p + q) + r$ and $p(qr) = (pq)r$;
- $+$ is commutative and idempotent with 0 as neutral element, i.e. for all $p, q \in K$ we must have $p + q = q + p$, $p + p = p$ and $p + 0 = p$;
- 1 is a neutral element for $\cdot$, i.e. for all $p \in K$ we must have $p1 = 1p = p$;
- for all $p \in K$ we have $p0 = 0p = 0$;
- $\cdot$ is distributive over $+$, i.e. for all $p, q, r \in K$ we must have $p(q + r) = pq + pr$ and $(p + q)r = pr + qr$;
- p^*q ist the least solution x of $q + px \leq x$ and qp^* is the least solution of $q + xp \leq x$, using the partial order $x \leq y \equiv x + y = y$.

We adopted the convention to write pq for $p \cdot q$, and to assume that $\cdot$ binds stronger than $+$, which allows us to dispense with some parentheses. In the sequel we will write $\mathcal{K} = (K, +, \cdot, ^*, 0, 1)$ to denote a Kleene algebra.

Of course, the standard example is regular sets. For other non-standard examples refer to [28] and [29].

Here, we want to use Kleene algebras to represent story algebras as discussed in the previous subsection. Obviously, + will correspond to the choice-operator, $\cdot$ to the sequence-operator, and * to the iteration operator. Furthermore, 1 will correspond to `skip` and 0 to the undefined process `fail`. However, we will need an extension to capture guards and post-guards, we have to think about the parallel-operator, and we have to handle associated scenes. Capturing guards and post-guards leads to Kleene algebras with tests, which were introduced in [30].

Definition 4.2. A *Kleene algebra with tests* (KAT) $\mathcal{K}$ consists of

- a Kleene algebra $(K, +, \cdot, ^*, 0, 1)$;
- a subset $B \subseteq K$ containing 0 and 1 and closed under + and $\cdot$;
- and a unary operation $^-$ on B, such that $(B, +, \cdot, ^-, 0, 1)$ forms a Boolean algebra.

We write $\mathcal{K} = (K, B, +, \cdot, *, ^-, 0, 1)$.

Now obviously the conditions appearing as guards and post-guards in a story algebra, form the set B of tests. So, if we ignore the parallel-constructor $\|$ for the moment, a story algebra gives rise to a KAT. However, we have to be aware that in such a KAT the operators + and $\cdot$ and the constants 0 and 1 play a double role:

- The operator + applied to two tests $\varphi, \psi \in B$ represents the logical OR, whereas in general it refers to the choice between two processes. As we have $(\varphi + \psi)p = \varphi p + \psi p$ this does not cause any problems.
- The operator $\cdot$ applied to two tests $\varphi, \psi \in B$ represents the logical AND, whereas in general it refers to the sequencing of two processes. As we have $(\varphi\psi)p = \varphi(\psi p)$ this also does not cause any problems.
- The constant 1 represents both TRUE and `skip`, whereas 0 represents both FALSE and `fail`, which both do not cause problems, as can be easily seen from the axioms of Kleene algebras.

Furthermore, we may define a *scene assignment* to a KAT by simply following the rules for the scene assignment in story algebras. That is, we obtain a partial mapping $\sigma : K \to \mathcal{S}$ with a set $\mathcal{S} = \{s_1, \ldots, s_n\}$ of scenes as follows:

- For $p_1, p_2 \in K$ with $\sigma(p_1) = \sigma(p_2) = s$ or one of the p_i is 1 or 0 or a test in B, then $\sigma(p_1 p_2) = s$.
- For $p_1, p_2 \in K$ with $\sigma(p_1) = \sigma(p_2) = s$ or one of the p_i is 1 or 0 or a test in B, then $\sigma(p_1 + p_2) = s$.
- For $p \in K$ with $\sigma(p) = s$ we obtain $\sigma(p^*) = s$.

Finally, let us look at parallel processes. From the intuition of scenes as abstract locations, we should assume that atomic actions from different scenes can be executed in parallel, which could be rephrased in a way that the order does not matter. Obviously, this extends to processes that are associated with different scenes. Therefore, the operator $\|$ is not needed for processes that belong to different scenes – any order will do. More formally, this means the following:

- If we have $p_1 \| p_2$ with $\sigma(p_i)$ both defined, but different, then this will be represented by $p_1 p_2$ (or $p_2 p_1$) in the KAT.

– In the KAT we then need $p_1p_2 = p_2p_1$, whenever $\sigma(p_1) \neq \sigma(p_2)$.

This leads to our definition of a many-sorted Kleene algebra with tests.

Definition 4.3. A *many-sorted Kleene algebra with tests* (MKAT) is a KAT $\mathcal{K} = (K, B, +, \cdot, ^*, ^-, 0, 1)$ together with a set $\mathcal{S} = \{s_1, \ldots, s_n\}$ of scenes and a scene assignment $\sigma : K \to \mathcal{S}$ such that $p_1p_2 = p_2p_1$ holds for all $p_1, p_2 \in K$ with $\sigma(p_1) \neq \sigma(p_2)$.

From our discussion above it is clear that we can represent a story space by an element of the MKAT that is defined by the atomic actions, the tests and the scenes.

Example 4.2. If we rewrite the story space from Example 4.1 we obtain the following KAT expression:

$$\alpha_1((\varphi_0\alpha_2 + \varphi_1\alpha_3(\alpha_5 + 1)\varphi_3 + \varphi_2\alpha_4(\alpha_6 + 1)\varphi_4)^*\varphi_5)(\alpha_7\varphi_6 + \alpha_{13}\varphi_7)$$
$$(\varphi_6\alpha_8(\alpha_8 + 1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8 + \varphi_7\alpha_8\alpha_8^*\alpha_{14}\alpha_{15}\alpha_{16}^*\alpha_{11}\alpha_{17}(\overline{\varphi_{12}}\alpha_{18}\alpha_{19})^*\varphi_{12}\alpha_{18}\varphi_9)$$
$$\alpha_{20}(\varphi_{10} + \varphi_{11})$$

4.1.4 Personalisation of Story Spaces

In order to reason about story spaces, we may now exploit the fact that they can be described by many-sorted KATs. Hoare logic [24] is the oldest formal system for reasoning about abstract programs. Its basic idea is to use partial correctness assertions – also called *Hoare triplets* – of the form $\{\varphi\}p\{\psi\}$. Here p is a program, and φ and ψ are its pre- and postcondition, respectively, i.e. logical formulae that can be evaluated in a program state.

The informal meaning of these triplets is that "whenever the program p is started in a state satisfying φ and terminates, then it will do so in a state satisfying ψ".

Using KATs, such a Hoare triplet corresponds to a simple equation $\varphi p\bar{\psi} = 0$. Equivalently, this can be formulated by $\varphi p \leq p\psi$ or $p\bar{\psi} \leq \bar{\varphi}p$ or $\varphi p = \varphi p\psi$.

In [31] it has been shown that KATs subsume propositional Hoare logic (PHL), i.e. all derivation rules of Hoare logic can be proven to be theorems for KATs. However, the theory of KATs is complete, whereas PHL is not.

In order to use KATs to reason about story spaces, the general approach is as follows. First we consider the atomic actions and the many-sorted KAT defined by them. In this KAT we can express the story space or parts of it by some process expression p. We then formulate a problem by using equations or conditional equations in this KAT. Furthermore, we obtain (conditional) equations, which represent application knowledge. This application knowledge arises from events, postconditions and knowledge about the use of the WIS for a particular purpose. We then apply all equations to solve the particular problem at hand.

The application knowledge contains at least the following equations:

1. If an action p has a precondition φ, then we obtain the equation $\bar{\varphi}p = 0$.
2. If an action p has a postcondition ψ, we obtain the equation $p = p\psi$.
3. If an action p is triggered by a condition φ, we obtain the equation $\varphi = \varphi p$.
4. In addition we obtain exclusion conditions $\varphi\psi = 0$ and tautologies $\varphi + \psi = 1$.

The problem of story space personalisation according to the preferences of a particular WIS user can be formalised as follows. Assume that $p \in K$ represents the story space. Then we

may formulate the *preferences* of a user by a set Σ of (conditional) equations. Let χ be the conjunction of the conditions in Σ. Then the problem is to find a minimal process $p' \in K$ such that $\chi \Rightarrow px = p'x$ holds for all $x \in K$. Preference equations can arise as follows:

1. An equation $p_1 + p_2 = p_1$ expresses an unconditional preference of activity (or process) p_1 over p_2.
2. An equation $\varphi(p_1 + p_2) = \varphi p_1$ expresses an conditional preference of activity (or process) p_1 over p_2 in case the condition φ is satisfied.
3. Similarly, an equation $p(p_1 + p_2) = pp_1$ expresses another conditional preference of activity (or process) p_1 over p_2 after the activity (or process) p.
4. An equation $p_1p_2 + p_2p_1 = p_1p_2$ expresses a preference of order.
5. An equation $p^* = pp^*$ expresses that in case of an iteration it will at least be executed once.

For instance, assume that the story space is $p = p_1(\varphi(p_2 + p_3) + \bar{\varphi}p_4^*p_5)$ and that we have the conditional preference rules $\varphi(p_2 + p_3) = \varphi p_2$ and $p_1\bar{\varphi}p_4^* = p_1\bar{\varphi}p_4p_4^*$. Then we get

$$px = p_1(\varphi(p_2 + p_3) + \bar{\varphi}p_4^*p_5)x = p_1\varphi p_2x + p_1\bar{\varphi}p_4^*p_5x = p_1\varphi p_2x + p_1\bar{\varphi}p_4p_4^*p_5x = p_1(\varphi p_2 + \bar{\varphi}p_4p_4^*p_5)x.$$

That is, we can simplify p by $p' = p_1(\varphi p_2 + \bar{\varphi}p_4p_4^*p_5)$. Obviously, we have $p' \leq p$, but further equations in our application knowledge may give rise to even a smaller solution. Let us finally illustrate this application with a non-artificial example.

Example 4.3. Let us continue Example 4.2. Assume that we have to deal with a user who already knows everything about loans. This can be expressed by the application knowledge equation $\varphi_5x = x$ for all $x \in K$. Furthermore, as knowledge about loans implies that there is no need for information about loans, we obtain three additional exclusion conditions:

$$\varphi_5\varphi_0 = 0 \qquad \varphi_5\varphi_1 = 0 \qquad \varphi_5\varphi_2 = 0$$

Taking these equations to the first part of the expression in Example 4.2 we obtain

$$\begin{aligned}
&\alpha_1((\varphi_0\alpha_2 + \varphi_1\alpha_3(\alpha_5 + 1)\varphi_3 + \varphi_2\alpha_4(\alpha_6 + 1)\varphi_4)^*\varphi_5)x = \\
&\alpha_1((\varphi_0\varphi_5\alpha_2 + \varphi_1\varphi_5\alpha_3(\alpha_5 + 1)\varphi_3 + \varphi_2\varphi_5\alpha_4(\alpha_6 + 1)\varphi_4)^*\varphi_5)x = \\
&\alpha_1((0\alpha_2 + 0\alpha_3(\alpha_5 + 1)\varphi_3 + 0\alpha_4(\alpha_6 + 1)\varphi_4)^*\varphi_5)x = \\
&\alpha_1 1\varphi_5x = \\
&\alpha_1x
\end{aligned}$$

That is, the whole story space can be simplified to

$$\alpha_1(\alpha_7\varphi_6 + \alpha_{13}\varphi_7)$$
$$(\varphi_6\alpha_8(\alpha_8 + 1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8 + \varphi_7\alpha_8\alpha_8^*\alpha_{14}\alpha_{15}\alpha_{16}^*\alpha_{11}\alpha_{17}(\overline{\varphi_{12}}\alpha_{18}\alpha_{19})^*\varphi_{12}\alpha_{18}\varphi_9)$$
$$\alpha_{20}(\varphi_{10} + \varphi_{11})$$

This means that for a user who knows about loans the part of the story space that deals with information about loans including sample applications will be cut out.

Personalisation also means satifying the information needs of a particular WIS user. This can be formalised by assuming that there is a *goal* that can be represented by some formula ψ. Thus, we can take $\psi \in B$. Furthermore, assume that our story space is represented by some process expression $p \in K$. Then the problem is to find a minimal process $p' \in K$ such that $p\psi = p'\psi$.

In order to find such a p' we have to use the application knowledge. In this case, however, we only obtain the general application knowledge that we already described above, unless we combine the application with personalisation.

For instance, assume we can write the story space p as a choice process $p_1 + p_2$. Let equations $\varphi\psi = 0$ and $p_2 = p_2\varphi$ (postcondition) be part of our application knowledge. If the goal is ψ, we get

$$p\psi = (p_1 + p_2)\psi = p_1\psi + p_2\psi = p_1\psi + p_2\varphi\psi = p_1\psi.$$

This means we can offer the simplified story space p_1 to satisfy the goal ψ. Let us finally illustrate this application with a non-artificial example.

Example 4.4. Let us continue Example 4.2 and look at a user who is going to apply for a home loan. This can be expressed by the goal φ_{10}. Then we express application knowledge by the equations $\varphi_{10}\varphi_{11} = 0$ (a user either applies for a home loan or a mortgage, not for both), $\varphi_{10}\varphi_9 = 0$ (a user applying for a home loan does not complete a mortgage application) and $\varphi_6\varphi_7 = 0$ (a user either selects a home loan or a mortgage, but not both).

Then we can simplify $p\varphi_{10}$ with the expression p from Example 4.2 step by step. First we get $(\varphi_{10} + \varphi_{11})\varphi_{10} = \varphi_{10}$, which can then be used for

$$\begin{aligned}
&(\varphi_6\alpha_8(\alpha_8 + 1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8 \\
&\qquad + \varphi_7\alpha_8\alpha_8^*\alpha_{14}\alpha_{15}\alpha_{16}^*\alpha_{11}\alpha_{17}(\overline{\varphi_{12}}\alpha_{18}\alpha_{19})^*\varphi_{12}\alpha_{18}\varphi_9)\varphi_{10} = \\
&\varphi_6\alpha_8(\alpha_8 + 1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8\varphi_{10} \\
&\qquad + \varphi_7\alpha_8\alpha_8^*\alpha_{14}\alpha_{15}\alpha_{16}^*\alpha_{11}\alpha_{17}(\overline{\varphi_{12}}\alpha_{18}\alpha_{19})^*\varphi_{12}\alpha_{18}\varphi_9\varphi_{10} = \\
&\varphi_6\alpha_8(\alpha_8 + 1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8\varphi_{10}
\end{aligned}$$

Then finally we get

$$\begin{aligned}
&(\alpha_7\varphi_6 + \alpha_{13}\varphi_7)\varphi_6\alpha_8(\alpha_8 + 1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8\varphi_{10} = \\
&\alpha_7\varphi_6\varphi_6\alpha_8(\alpha_8 + 1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8\varphi_{10} \\
&\qquad + \alpha_{13}\varphi_7\varphi_6\alpha_8(\alpha_8 + 1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8\varphi_{10} = \\
&\alpha_7\varphi_6\alpha_8(\alpha_8 + 1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8\varphi_{10}
\end{aligned}$$

This means that the story space can be simplified to

$$\begin{aligned}
&\alpha_1((\varphi_0\alpha_2 + \varphi_1\alpha_3(\alpha_5 + 1)\varphi_3 + \varphi_2\alpha_4(\alpha_6 + 1)\varphi_4)^*\varphi_5) \\
&\qquad \alpha_7\varphi_6\alpha_8(\alpha_8 + 1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8\alpha_{20}\varphi_{10}
\end{aligned}$$

This simply means that for a user who is looking for a home loan application the part of the story space that deals with mortgage application will be cut out.

In addition, a user who will apply for a home loan does not need to be informed about mortgages. This is again application knowledge about preferences that can be formalised by $\varphi_2 x = 1$ for all $x \in K$. This implies that the story space can be further simplified to

$$\alpha_1((\varphi_0\alpha_2 + \varphi_1\alpha_3(\alpha_5 + 1)\varphi_3)^*\varphi_5)\alpha_7\varphi_6\alpha_8(\alpha_8 + 1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8\alpha_{20}\varphi_{10}$$

This simply means that for a user who is looking for a home loan application the part of the story space that deals with information about mortgage applications can be cut out as well.

4.2 Actors

Users can be classified according to their roles, intentions and behaviour. We use the term *actor* for such a group of users. The *role* of an actor indicates a particular purpose of the system. As such it is usually associated with obligations and rights, which lead to deontic integrity constraints. Roles are also connected with the *tasks*, but tasks will be handled separately. The *intention* of an actor can be modelled by goals, i.e. postconditions to the story space. Modelling the behaviour of an actor leads to *user profiles*, which can be modelled by giving values for various properties that characterize a user. Furthermore, each profile leads to rules that can again be expressed by constraints on the story space.

In addition, each actor has an *information portfolio*, which specifies the information needs as well as the information entered into the system. We do not model the information portfolio as part of an actor, but instead of this we will model the information "consumed" and "produced" with each more detailed specification of a scene.

4.2.1 Modelling Information Portfolios

Each WIS user who enters the system with a particular goal has information needs that have to be satisfied by the system. In addition, an active WIS will also request information from its users. We use the term *information consumption* for the information provided by the system to its users, and *information production* for the information entered by a user into the system.

When a user enters the WIS, the information needs are usually not known in advance. Part of the needed information may depend on other parts, on decisions made while navigating through the WIS, and even on the information provided by the actor him/herself. That is, the information consumption and production depends on the path through the WIS, i.e. in our terminology on the story. Therefore, we associate information consumption and production with each scene of the story space. Assuming that there is a database for the data content of the WIS with database schema $\mathcal{S}$, information consumption on a scene s definitely accounts for a *view* V_s over $\mathcal{S}$. That is, we have another schema $\mathcal{S}_V$ and a computable transformation from databases over $\mathcal{S}$ to databases over $\mathcal{S}_V$. Such a transformation is usually expressed by a query q_V. Such views will form the basis of the theory of media types, which we will treat intensively in Section 5.

However, a little subtlety comes in here. Views $V_s = (\mathcal{S}_V, q_V)$ are defined in a way that applying them to a database db over the schema $\mathcal{S}$ results in a database $q_V(db)$ over $\mathcal{S}_V$, i.e. in sets. If we assume that $q_V(db)$ is just one set of objects – this is what we will do in Section 5 – then the information that is made available to an actor corresponds to only one object in this set. This object is determined by parameters provided by the actor, i.e. by the information production on the previous scene of the story. This further implies that the information production should be associated not just with a scene, but with an action at that scene.

Thus, we obtain the following extensions to story spaces:

- With each scene s we associate a view $V_s = (\mathcal{S}_V, q_V)$ called *information consumption view*. Elements of $q_V(db)$ for some database db represent the *information consumption* of an actor.

– With each action α we associate a data type t_α called *information production type*. Values of type t_α represent the *information production* by an actor.

Information consumption and information production of an actor for all scenes together define the information portfolio of the actor.

Example 4.5. Take another look at the scene s_1 = type_of_loan in Example 4.1. In this case the information consumption of any actor is a list of loans. In scene s_{10} = loan_overview this would be extended by a short explanation for each of them. For the scene s_{13} = mortgage the information consumption of an actor would be the description of one particular type of mortgage. This implies that action α_4 = request_mortgage_details needs this particular type of mortgage to be selected by the actor. Thus, the data type describing mortgages would be used to define the information production associated with α_4.

In the next chapter we will take a closer and more formal look at information production and consumption in the context of media types.

4.2.2 Modelling Roles, Obligations, Rights and Intentions

The presence of roles indicates a particular purpose of the system. For instance, in a web-based conference system we may have roles for the programme committee chair(s), the programme committee members, and for authors. On the other hand, in an on-line loan system we may not wish to distinguish roles, as all actors will only appear in the one role of a customer.

A *role* is defined by the set of actions that an actor with this role may execute. Thus, we first associate with each scene in the story space a set of role names, i.e. whenever an actor comes across a particular scene, s/he will have to have one of these roles. Furthermore, a role is usually associated with obligations and rights, i.e. which actions have to be executed or which scenes are disclosed.

An *obligation* specifies what an actor in a particular role has to do. A *right* specifies what an actor in a particular role is permitted to do. Both obligations and rights together lead to complex deontic integrity constraints. We use the following logical language $\mathcal{L}$ for this purpose:

– All propositional atoms are also atoms of $\mathcal{L}$.
– If α is an action on scene s and r is a role associated with s, then $\mathbf{O}\ do(r, \alpha)$ is an atom of $\mathcal{L}$.
– If α is an action on scene s and r is a role associated with s, then $\mathbf{P}\ do(r, \alpha)$ is an atom of $\mathcal{L}$.
– If α is an action on scene s and r is a role associated with s, then $\mathbf{F}\ do(r, \alpha)$ is an atom of $\mathcal{L}$.
– For $\varphi, \psi \in \mathcal{L}$ we also have $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \Rightarrow \psi$ and $\varphi \Leftrightarrow \psi$ are also formulae in $\mathcal{L}$.

The interpretation is standard. In particular, $\mathbf{O}\ do(r, \alpha)$ means that an actor with role r is obliged to perform action α, $\mathbf{P}\ do(r, \alpha)$ means that an actor with role r is permitted to perform action α, and $\mathbf{F}\ do(r, \alpha)$ means that an actor with role r is forbidden to perform action α.

Example 4.6. Though the on-line loan example from Example 4.1 only contains one role customer, this customer has the obligation to leave his/her details (action α_8), once a home loan or a mortgage has been selected (condition φ_6 or φ_7), Furthermore, if a mortgage is

selected (condition φ_7), the customer is obliged to describe securities (action α_{16}) and to enter obligations (action α_{17}), i.e. we obtain the deontic constraints

$$\varphi_6 \vee \varphi_7 \Rightarrow \mathbf{O}\ do(\text{customer}, \alpha_8)$$

and

$$\varphi_7 \Rightarrow \mathbf{O}\ do(\text{customer}, \alpha_{16}) \wedge \mathbf{O}\ do(\text{customer}, \alpha_{17})$$

Furthermore, a customer is allowed to look at mortgage samples (action α_6), which is simply expressed by $\mathbf{P}\ do(\text{customer}, \alpha_6)$.

We may of course extend the on-line loan system in a way that the processing of a loan application will be included. In this case we would obtain additional roles, e.g. bank_clerk or mortgage_advisor, and additional deontic constraints.

The *intention* of an actor can be expressed by goals, which can be modelled by postconditions to the story space. In the previous section, in particular in Example 4.4, we already discussed such goals.

4.2.3 Modelling User Profiles

Modelling the behaviour of an actor leads to *user profiles*. We may ask which properties characterize a user and provide values for each of these properties. Each combination of such values defines a profile, but usually the behaviour for some of these profiles is the same. Furthermore, each profile leads to rules that can again be expressed by constraints on the story space. In the previous section, in particular in Example 4.3, we already discussed such rules.

The dimensions used in user profiles depend on the application. A rough classification of sources for such dimensions is the following:

- the ability to search for solutions, solve problems, detect and resolve conflicts, schedule work tasks;
- the communication skills and computer literacy;
- the knowledge and education level regarding the task domain;
- the frequency and intensity of system usage;
- the way information is handled, i.e. the direction of the information flow, the necessary and optional input, the intended information usage, the amount and size of information and the complexity of information;
- the experience in working with the system and with associated tasks.

Formally, in order to describe such user profiles, we start with a finite set Δ of *user dimensions*, e.g. $\Delta = \{\text{experience, skill, goal\_orientation, presentation\_preferences, training}\}$. For each dimension $\delta \in \Delta$ we assume to be given a scale $sc(\delta)$. Formally, a *scale* is a totally ordered set.

Example 4.7. The scale for goal-orientation may be

$$sc(\text{goal-orientation}) = \{\text{surfer, navigator, searcher}\}$$

with surfer $\leq$ navigator $\leq$ searcher. As another example consider

$$sc(\text{presentation\_preferences}) = \{\text{detailed, normal, condensed, terse}\}$$

with detailed $\leq$ normal $\leq$ condensed $\leq$ terse.

Definition 4.4. Let $\Delta = \{\delta_1, \ldots, \delta_n\}$ be a set of user dimensions. Then the *set of user profiles* (or the *user-grid*) over Δ is $gr(\Delta) = sc(\delta_1) \times \cdots \times sc(\delta_n)$. A *user type* over Δ is a convex region $U \subseteq gr(\Delta)$.

We then add user types to the story space by assigning a set of user types to each scene. This indicates which stories will be supported for which user profiles.

4.3 Tasks

The primary purpose of a WIS is to provide information to its users. This information is usually used to perform a certain task. Such tasks can be performed by a single user or they can be the cooperative effort of several users. That is, we consider WISs to be task-oriented systems.

For instance, a task in a home loan system can be to submit a mortgage application. This involves only one actor in a single role customer. In an extended system, the task may be to submit, approve and implement a mortgage, in which case other actors in the role of a bank clerk or a mortgage advisor would participate in the task.

Tasks describe the general purposes of the WIS. They combine roles that are involved in the task, actions executed by actors in these roles, consequently scenes to which these actions belong, information consumed by the actions, and data flowing between the actions. In addition, there is an event that triggers the task. Actions can be grouped together into subtasks to provide a more concise form of task specification. Thus, a task is largely specified by what has already been defined for the story space including the integrity constraints that define rights, obligations, intentions and behaviour.

Formally, a *task* τ consists of a set $act(\tau) = \{\tau_1, \ldots, \tau_n\}$ of subtasks, which may be actions in the story space, and a triggering event $ev(\tau)$, which is the combination of a boolean condition φ on the story space and the fact that a particular action α was executed by some role r, i.e. $ev(\tau) = (\varphi, do(r, \alpha))$. Furthermore, with each subtask τ_i we associate a set of scenes and a set of roles. If τ_i is atomic, i.e. an action α, then is will be associated with exactly one scene s and exactly one role r. In this case, we also associate a view $V_i = (\mathcal{S}_i, q_i)$ with $\tau_i = \alpha$, which is a view over the view V_s. Furthermore, with any pair of subtasks (τ_i, τ_j) $(i \neq j)$ we associate another view $V_{ij} = (\mathcal{S}_{ij}, q_{ij})$ describing the data communicated from subtask τ_i to subtask τ_j.

Example 4.8. Look again at the loan application system from Example 4.1. In this system we may have a task $\tau = \text{application\_for\_mortgage}$ with triggering event $ev(\tau) = (\text{TRUE}, do(\text{customer}, \alpha_1))$, i.e. the task is triggered when a customer enters the loan system.

Then the set of subtasks can be

$$act(\tau) = \{\alpha_1, \alpha_2, \alpha_4, \alpha_6, \alpha_8, \alpha_{11}, \alpha_{13}, \alpha_{14}, \alpha_{15}, \alpha_{16}, \alpha_{17}, \alpha_{18}, \alpha_{19}, \alpha_{20}\},$$

i.e. it is a set of actions that would equally arise from a personalisation with the goal φ_{11}.

The scenes are as specified in Example 4.1, and the associated role for all subtasks is always customer.

Of course, the deontic constraints specified for the story space apply to each task specification.

Chapter 5

Modelling Content and Functionality

We now present the central concepts used for modelling the content and the functionality of a WIS. We may assume that we have an underlying database. Abstracting from a specific data model the schema of this database can be described by *database types*. We define the semantics in such a way that schemata can be easily mapped to an XML representation [1]. We also shall distinguish *data types* as means for the conceptual description of domain values.

The next step is to introduce *interaction types* based on extended views over the database schema. This will allow us to apply completely different design criteria for the database schema and the *interaction schema*. One major facility used in interaction types is the possibility to create a navigation structure via URLs and links. We present an algebraic approach to querying and view definition following [38].

Finally, we introduce *media types* as the building blocks used to define a *site schema*. These media types arise from tailoring the information types in such a way that different presentation options will be enabled. In addition, we show how to support contextual information.

5.1 The Structure: Database Types

In principle, it is not important what kind of database we have as long as there exists a sufficiently powerful query mechanism that permits to define views. For the purpose of our presentation here, however, we focus on a conceptual description of such databases using a datamodel close to the higher-order Entity-Relationship model (HERM) [47].

Let an underlying type system be defined as

$$t = b \mid (a_1 : t_1, \ldots, a_n : t_n) \mid \{t\} \mid [t].$$

Here b represents an arbitrary collection of *base types*, e.g., *BOOL* for boolean values **T** and **F**, $\mathbb{1}$ for a single value **1**, *TEXT* for text, *PIC* for images, *MPIC* for video data, *CARD* and *INT* for numbers, *DATE* for dates, *URL* for URL-addresses, *MAIL* for e-mail-addresses, etc. The constructors $(\cdot)$, $\{\cdot\}$ and $[\cdot]$ are used for records (or tuples), finite sets and finite lists. We sometimes omit the field labels a_i in record types and simply write $t_1 \times \cdots \times t_n$ instead of $(a_1 : t_1, \ldots, a_n : t_n)$.

Definition 5.1. A *database type of level k* has a name E and consists of a set $comp(E) = \{r_1 : E_1, \ldots, r_n : E_n\}$ of components with pairwise different role names r_i and database types (or clusters) E_i on levels lower than k with at least one database type of level exactly $k-1$,

a set $attr(E) = \{a_1, \ldots, a_m\}$ of attributes, each associated with a data type $dom(a_i)$ as its domain, and a key $id(E) \subseteq comp(E) \cup attr(E)$. We shall write $E = (comp(E), attr(E), id(E))$.

A *cluster of level k* has a name E and consists of a set $frag(E) = \{f_1 : E_1, \ldots, f_n : E_n\}$ of fragments with pairwise different fragment names r_i and database types (or clusters) E_i on levels at most k with at least one of the E_i of level exactly k.

A *database schema* is a finite set $\mathcal{S}$ of database types and clusters such that for all $E \in \mathcal{S}$ and all $(r_i : E_i) \in comp(E)$ or $(f_i : E_i) \in frag(E)$, respectively, we also have $E_i \in \mathcal{S}$.

Following [47] the definition of *databases* over a given database schema $\mathcal{S}$ is straightforward. However, we would like to change the semantics of a schema in such a way that it allows an easy transformation to XML [1]. Note that the major difference is that we have to consider partial mappings instead of total mappings in order to define tuples.

Definition 5.2. A *tuple* over a database type $E = (comp(E), attr(E), id(E))$ is a partial mapping t defined on $comp(E) \cup attr(E)$ such that the following conditions hold:

- $t(r_i : E_i)$ is either undefined or a tuple over E_i (for $(r_i : E_i) \in comp(E)$);
- $t(a_j)$ is either undefined or a value in $dom(a_j)$ (for $a_j \in attr(E)$);
- t is always defined on all element of the key $id(E)$.

A *tuple* over a cluster E is a pair $(f_i : t_i)$ for some $(f_i : E_i) \in frag(E)$ and a tuple t_i over E_i.

A *database* $\mathcal{D}$ over a database schema $\mathcal{S}$ is an $\mathcal{S}$-indexed family $\{\mathcal{D}(E)\}_{E \in \mathcal{S}}$ of finite sets $\mathcal{D}(E)$ of tuples over E such that the following conditions hold:

- If E is a type, then for each $(r_i : E_i) \in comp(E)$ and each $t \in \mathcal{D}(E)$ for which $t(r_i : E_i)$ is defined $t(r_i : E_i) \in \mathcal{D}(E_i)$ holds;
- If E is a cluster, then for each $(f_i : E_i) \in frag(E)$ and each $(f_i : t_i) \in \mathcal{D}(E)$ we have $t_i \in \mathcal{D}(E_i)$.

Furthermore, we can easily define a graphical representation for a database schema.

Example 5.1. We continue the on-line loan application from Example 4.1. For this application we obtain the following database type definitions:

Loan_Type = (∅, { type, conditions, interest }, { type })

Customer = (∅, { customer_no, name, address, date_of_birth },
{ customer_no })

Home_Loan = ({ type : Loan_Type }, { loan_no, amount,
interest_rate, begin, end, terms_of_payment }, { loan_no })

Mortgage = ({ type : Loan_Type }, { mortgage_no, amount, disagio,
interest_rate, begin, end, object }, { mortgage_no })

Loan = Home_Loan ⊕ Mortgage

Account = ({ ℓ : Loan }, { account_no, amount }, { account_no })

Account_Record = ({ a : Account }, { record_no, type, amount,
date }, { a : Account, record_no })

Owes = ({ who : Customer, what : Loan }, { begin, end },

{ who : Customer, what : Loan, begin })

Security = ({ whose : Customer, for : Mortgage }, { value, object, type }, {whose : Customer, for : Mortgage, object })

Income = ({ who : Customer }, { type, amount, frequency, account }, { who : Customer, account })

Obligation = ({ who : Customer }, { type, amount, frequency, account }, { who : Customer, account })

Figure 5.1 provides a graphical representation of a database for this application. According to the common convention in Entity-Relationship diagrams we represented types on level 0 by rectangles and types on higher levels by diamonds. Attributes and role names have been omitted in the schema. Clusters are represented by the $\oplus$ symbol, but we omitted fragment names.

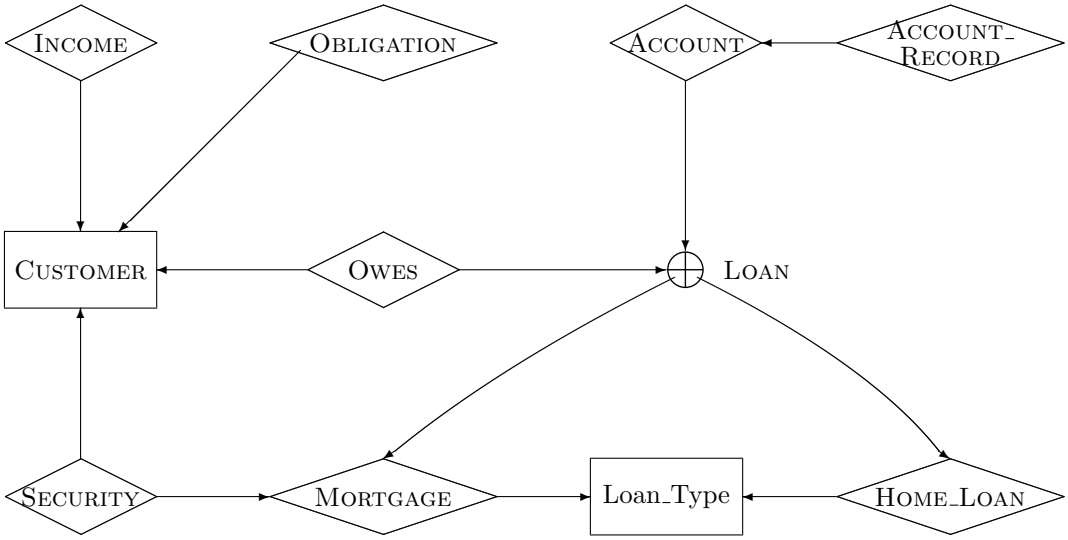

Fig. 5.1. Graphical Representation of a Database Schema

To complete the structure definition of the underlying database we have to add static integrity constraints to the definition of a database schema. We refer to [47] for a detailed description of such constraints.

5.2 Generated Content and Functionality: Interaction Types

Interaction types describe the content and functionality at a particular scene in the story space. We have seen that the content is defined by a view on some underlying database, so the core of an interaction type will be defined by a view, which roughly speaking is itself defined by a stored query. Furthermore, scenes are parameterised, so we will refer to the individual members of the view as *interaction objects*.

Functionality is defined by adding operations to the interaction types. These operations have to be understoof as realisations of the actions used in the story space.

Definition 5.3. A *view* V on a database schema $\mathcal{S}$ consists of a view schema $\mathcal{S}_V$ and a defining query q_V, which transforms databases over $\mathcal{S}$ into databases over $\mathcal{S}_V$.

An *interaction type* has a name M and consists of a content data type $cont(M)$ with the extension that the place of a base type may be occupied by a pair $\ell : M'$ with a label ℓ and the name M' of an interaction type, a defining query q_M such that $(\{t_M\}, q_M)$ defines a view, and a set of operations. Here t_M is the type arising from $cont(M)$ by substitution of *URL* for all pairs $\ell : M'$.

In principle we could use any query language. However, for our purposes the query language used in the views must be powerful enough to create navigation links, i.e. we must create URLs in the result of a query. Thus, our first task is to introduce such a query language. Then we will discuss the extension by adding operations.

Finite closed sets $\mathcal{C}$ of interaction types define *content schemata*. Then a database $\mathcal{D}$ over the underlying database schema $\mathcal{S}$ and the defining queries determine finite sets $\mathcal{D}(M)$ of pairs (u, v) with URLs u and values v of type t_M for each $M \in \mathcal{C}$. We use the notion *pre-site* for the extension of $\mathcal{D}$ to $\mathcal{C}$. The pair (u, v) will be called an *interaction object* in the pre-site $\mathcal{D}$.

5.2.1 Query Algebras

The defining query of an interaction type may be expressed in any suitable query language, e.g. query algebra, logic or an SQL-variant. We shall outline a general algebraic approach following [38]. There it has been shown that any query algebra can be defined by operations defined from the underlying type system plus one generalized join-operation. This extends to rational tree types, hence is suitable for our case, as we may expand the links defined via values of type URL to obtain rational trees. On the other hand, it can be shown that such a query language is equivalent to a query language which allows to create URLs in a non-deterministic way.

Roughly speaking, a rational tree in our case would correspond to a page of infinite size which results from replacing all links by copies of the referenced page. As the navigation structure may contain cycles, the resulting page would be infinite.

Consider a trivial type denoted $\mathbb{1}$ and a boolean type $BOOL$. Values of these types are $\mathbf{1}$ and $\mathbf{T}$, $\mathbf{F}$, respectively. There is no operation on $\mathbb{1}$, but for $BOOL$ we may consider the operations $\wedge : BOOL \times BOOL \to BOOL$ (conjunction), $\neg : BOOL \to BOOL$ (negation) and $\Rightarrow$: $BOOL \times BOOL \to BOOL$ (Implication). Furthermore, we consider two constants $\texttt{true} : \mathbb{1} \to BOOL$ and $\texttt{false} : \mathbb{1} \to BOOL$.

For tuple types we consider *projection* $\pi_i : t_1 \times \cdots \times t_n \to t_i$ and *product* $o_1 \times \cdots \times o_n : t \to t_1 \times \cdots \times t_n$ for given operations $o_i : t \to t_i$.

For set types me may consider $\cup$ (union), $-$ (difference), the constant $\texttt{empty} : \mathbb{1} \to \{t\}$ and the *singleton* operation $\texttt{single} : t \to \{t\}$ with well known semantics. In addition, we consider structural recursion, which will be discussed below. We dispense with a discussion of the similar situation for list and multiset types.

For function types we consider *composition* $\circ : (t_2 \to t_3) \times (t_1 \to t_2) \to (t_1 \to t_3)$, *evaluation* $\texttt{ev} : (t_1 \to t_2) \times t_1 \to t_2$, and *abstraction* $\texttt{abstr} : (t_1 \times t_2 \to t_3) \to (t_1 \to (t_2 \to t_3))$. All these operations are standard.

For completeness assume an equality predicate $=_t$: $t \times t \to BOOL$ for all types t except function types and a membership predicate $\in$: $t \times \{t\} \to BOOL$. We shall also use a unique "forget"-operation $\texttt{triv} : t \to \mathbb{1}$ for each type t. Combining all the operations for all types of the type system gives all operations induced from the type system.

Let us now take a closer look at a powerful class of operations defined by the method of structural recursion [47]. For set types there are three natural constructors: the constant `empty`, the singleton operation and the union operation. In order to define an operation on a set type, say $\mathtt{op} : \{t\} \to t'$ it is therefore sufficient to define it on the empty set, on singleton sets and on unions.

Formally, we define $\mathtt{op} = \mathtt{src}[e, g, \sqcup]$ with a value e of type t', a function $g : t \to t'$ and a function $\sqcup : t' \times t' \to t'$. Then $\mathtt{src}[e, g, \sqcup]$ is defined as follows:

$$\begin{aligned} \mathtt{src}[e,g,\sqcup](\emptyset) &= e \\ \mathtt{src}[e,g,\sqcup](\{x\}) &= g(x) && \text{for each } x \text{ of type } t, \text{ and} \\ \mathtt{src}[e,g,\sqcup](X \cup Y) &= \mathtt{src}[e,g,\sqcup](X) \sqcup \mathtt{src}[e,g,\sqcup](Y) && \text{for disjoint } X, Y \text{ of type } \{t\}. \end{aligned}$$

Analogously, for lists we have the empty list $[]$, a singleton operation giving $[x]$ and list concatenation $X.Y$. In this case we obtain the analogous definition

$$\begin{aligned} \mathtt{src}[e,g,\sqcup]([]) &= e \\ \mathtt{src}[e,g,\sqcup]([x]) &= g(x) && \text{for each } x \text{ of type } t, \text{ and} \\ \mathtt{src}[e,g,\sqcup](X.Y) &= \mathtt{src}[e,g,\sqcup](X) \sqcup \mathtt{src}[e,g,\sqcup](Y) && \text{for each } X, Y \text{ of type } [t]. \end{aligned}$$

Let us illustrate structural recursion by some more or less standard examples. First consider a function $f : t \to t'$ for arbitrary types t and t'. We want to "raise" f to a function $\mathtt{map}(f) : \{t\} \to \{t'\}$ by applying f to each element of a set. Obviously, we have

$$\mathtt{map}(f) = \mathtt{src}[\emptyset, \mathtt{single} \circ f, \cup].$$

Next consider a function $\varphi : t \to \mathit{BOOL}$. We want to define an operation $\mathtt{filter}(\varphi) : \{t\} \to \{t\}$, which associates with a given set the subset of all elements "satisfying the predicate" φ, i.e. elements that are mapped to $\mathbf{T}$. Then we may write

$$\mathtt{filter}(\varphi) = \mathtt{src}[\emptyset, \mathtt{if\_then\_else} \circ (\varphi \times \mathtt{single} \times (\mathtt{empty} \circ \mathtt{triv})), \cup]$$

with the function $\mathtt{if\_then\_else} : \mathit{BOOL} \times t \times t \to t$ with $(\mathbf{T}, x, y) \mapsto x$ and $(\mathbf{F}, x, y) \mapsto y$.

As a third example assume that t is a "number type", on which addition $+ : t \times t \to t$ is defined. Then $\mathtt{src}[0, id, +]$ with the identity function id on t defines the sum of the elements in a set. In this way all the known aggregate functions of SQL (and more) can be defined by structural recursion.

In [38] it has been shown that the operations defined so far are sufficient to express operations such as `nest` and `unnest`.

5.2.2 The Join Operation

In order to obtain also a generalised join it is a natural idea to exploit subtyping on the type system. This is a preorder $\leq$ on the types.

Suppose, our collection of base types contains at least the type $\mathbb{1}$. *BOOL* may be identified with $\{\mathbb{1}\}$. Then subtyping can be defined in the standard way as the smallest preorder such that the following holds:

- For any type t we have $t \leq \mathbb{1}$.

- For set types (or list types, respectively) we have $\{t\} \leq \{t'\}$ (or $[t] \leq [t']$, respectively) iff $t \leq t'$ holds.
- For tuple types we have $t_1 \times \cdots \times t_m \leq t'_1 \times \cdots \times t'_n$ iff $t_{\sigma(i)} \leq t'_i$ holds for some injective $\sigma : \{1, \ldots, n\} \to \{1, \ldots, m\}$.

Then each subtype relation $t \leq t'$ defines an associated subtype function $\pi_{t'} : t \to t'$. Note that the projections in relational algebra are just such subtype functions. Indeed, t is the least common supertype of t_1 and t_2; $t_1 \bowtie_t t_2$ is a common subtype.

The following theorem is central for the definition of the general join [38].

Theorem 5.4. *Consider a type system with the trivial type $\mathbb{1}$ as one of its base types and with constructors among the tuple, set and list constructors. If t is a common supertype of t_1 and t_2 with associated subtype functions $\pi^i_t : t_i \to t$, then there exists a common subtype $t_1 \bowtie_t t_2$ together with subtype functions $\pi_{t_i} : t_1 \bowtie_t t_2 \to t_i$ such that $\pi^1_t \circ \pi_{t_1} = \pi^2_t \circ \pi_{t_2}$ holds. Furthermore, for any other common subtype t' with subtype functions $\pi'_{t_i} : t' \to t_i$ with $\pi^1_t \circ \pi'_{t_1} = \pi^2_t \circ \pi'_{t_2}$ there is a unique subtype function $\pi : t' \to t_1 \bowtie_t t_2$ with $\pi_{t_i} \circ \pi = \pi'_{t_i}$.*

For $t = \mathbb{1}$ we obtain $t_1 \bowtie_t t_2$ simply as the product $t_1 \times t_2$. With the existence of the *join types* $t_1 \bowtie_t t_2$ the join over t can be defined as in the relational case. For this let C_1 and C_2 be classes. We define

$$C_1 \bowtie_t C_2 \quad = \quad \{z : T_{C_1} \bowtie_t T_{C_2} \mid \exists z_1 \in C_1 . \exists z_2 \in C_2 . \pi_{t_1}(z) = z_1 \wedge \pi_{t_2}(z) = z_2\} \quad .$$

Example 5.2. Consider $t_1 = \{b_1 \times \{b_2 \times b_3\} \times b_4\}$ and $t_2 = \{b_1 \times \{b_5 \times b_3\} \times b_6\}$ with the common supertype $t = \{b_1 \times \{b_3\}\}$. Then we obtain the join type

$$t_1 \bowtie_t t_2 \quad = \quad \{b_1 \times \{b_2 \times b_5 \times b_3\} \times b_4 \times b_6\} \quad .$$

5.2.3 Handling URLs

The structures allowed by the definition of databases in the previous section are all finite. In fact, values can be represented as finite trees. A slight generalization would be to allow infinite trees, but of course only such infinite trees that can be represented in a finite way. For this we introduce *labels* ℓ. We extend any given type system in such a way allowing types to be adorned with labels and labels themselves to be treated in the same way as base types. Thus, our type system extends to

$$t \quad = \quad b \mid \ell \mid t_1 \times \cdots \times t_n \mid \{t\} \mid [t] \mid \ell : t.$$

Furthermore, we have to restrict ourselves to *well-defined* types. For this we require that for each label ℓ occurring within a type t—in a place, where we could have a base type instead—some decorated type $\ell : t'$ must occur in t, too.

Values of such types with labels can be written as an infinite tree. Figure 5.2 a) shows such a tree. We call a tree *rational* iff the number of different subtrees is finite. Then only rational trees will be allowed as values of well-defined types with labels. For our example, this means to restrict to values of the form

$$(n_1, a_1, (n_2, a_2, (\ldots, (n_k, a_k, (\ldots)))),$$

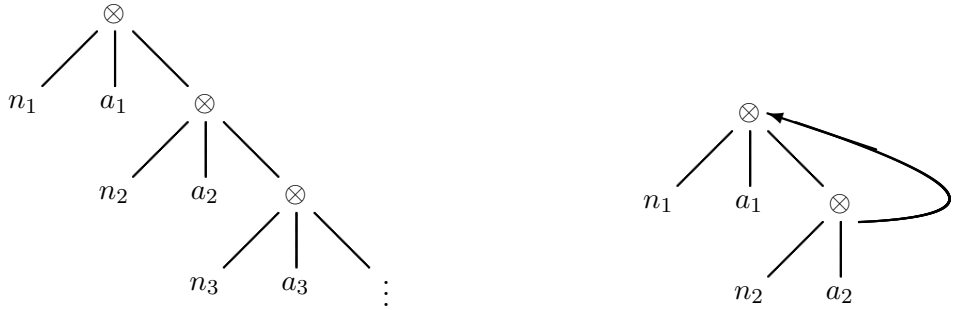

Fig. 5.2. Rational Tree **a)** **b)**

such that $n_i = n_j$ and $a_i = a_j$ holds for some i and j. In addition, we would like to add a constraint and require $i = 1$, $j = 3$, but this constraint must be added explicitely; it is not captured by the type system. Figure 5.2 b) illustrates such a rational tree.

Since we restrict ourselves to well-defined types with labels, which can be written as rational trees, and allow only rational trees as values, we shall talk of *rational tree types*.

One important feature of rational tree types is that the query algebra outlined in the previous subsections extends naturally to rational tree types. Furthermore, as the representation with URLs can be regarded as a means to finitely represent rational trees, it can also be shown that we can replace the rational trees by the URLs, iff the query language is extended in such a way that it can create URLs and links. This can be done as follows:

- `create_urls` transforms a set $\{v_1, \ldots, v_m\}$ of values into a set $\{(u_1, v_1), \ldots, (u_m, v_m)\}$ of pairs with new created URLs u_i of type *URL*;
- It also transforms a list $[v_1, \ldots, v_m]$ of values into a list $[(u_1, v_1), \ldots, (u_m, v_m)]$ of pairs with new created URLs u_i of type *URL*;
- The operation `create_url` transforms a value v of any type into a pair (u, v) with a new URL u.

Theorem 5.5. *Let S be a database schema, where the types of classes are rational tree types and let S' be an equivalent schema that uses the type URL, but no rational tree types. Then the result of an algebra query on S' with URL and link creation is the URL-based representation of the result of the same query applied to S and vice versa.*

Figure 5.3 illustrates the relationship between querying with URL creation and querying with rational trees.

Example 5.3. Consider the scene home_loan in Example 4.1. For this scene the information consumption is the description of a particular loan type. So we get

$$\begin{aligned} \mathit{cont}(\text{home\_loan}) = (&\text{type} : \mathit{STRING}, \text{conditions} : \mathit{STRING}, \\ &\text{interest} : \{\ (\text{amount} : \mathit{CARD}, \text{rate} : \mathit{FLOAT})\ \})\ . \end{aligned}$$

In this case the defining query is simply $q_{\text{home\_loan}} =$ `create_urls`(Loan_Type).

5.2.4 Operations on Interaction Types

Conceptual abstraction of database behaviour is achieved via *operations* associated with database types. These operations can be described in a way known from programming languages. Here

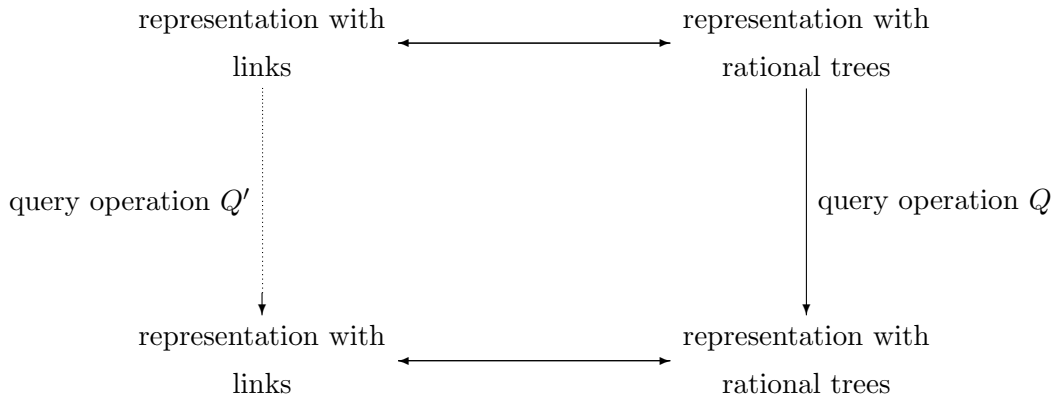

Fig. 5.3. Handling URLs in queries

we adopt an imperative style. Then, in order to model the required functionality we also add operations to interaction types. This is completely analogous to the d-operations on dialogue types [40].

Definition 5.6. An *operation* on a database type C consists of a *signature* and a *body*. The *signature* consists of an operation name O, a set of input-parameter/type pairs $\iota_i :: T_i$ and a set of output-parameter/type pairs $o_j :: T'_j$. The *body* is recursively built of the following constructs:

- *assignment* $x_E := exp$, where x is a variable representing the content of the type E itself or a local variable (including the output-parameters), and exp is an expression of the same type as x_E,
- *local variable declaration* `Let` $x : t$,
- *skip* and $fail$,
- *sequencing* $S_1 \,;\, S_2$ and *branching* `IF` $\mathcal{P}$ `THEN` S_1 `ELSE` S_2 `ENDIF` ,
- *operation call* $E' :\!\!- O'(\text{in} : exp'_1, \ldots, exp'_j, \text{out} : x'_1, \ldots, x'_i)$, where O' is an operation on the type E' with compatible signature, and
- non-deterministic *selection* of values $New.f(x)$, where f is a selector on E.

An *operation* on an interaction type M consists of an operation signature, i.e., name, input-parameters and output-parameters, a selection type which is a supertype of $cont(M)$, and a body which is defined via operations accessing the underlying database.

There exist several standard operations that are of particular interest in web information systems:

- *Generalization functions* are used for generation of aggregated data. They are useful especially in the case of insufficient space or for the display of complementary, generalized information after terminating a task. Hierarchy rules are used for the specification of applicability of generalization functions. The roll-up function in [2], slicing, and grouping are special generalization functions.
- *Specialization functions* are used for querying the database in order to obtain more details for aggregated data. The user can obtain more specific information after he has seen the aggregated data. Hierarchy rules are used for the specification of applicability of specialization functions. The drill-down function used in the data warehouse approach is a typical example.

- *Reordering functions* are used for the rearrangement of units. The pivoting, dimension destroying, pull and push functions [2] and the rotate function are special reordering functions.
- *Browsing functions* are useful in the case that information containers are too small for the presentation of the complete information.
- *Sequentialization functions* are used for the decomposition of sets or sequences of information.
- *Linking functions* are useful whenever the user is required to imagine the context or link structure of interaction types.
- *Survey functions* can be used for the graphical visualization of the contents of the interaction type.
- *Searching functions* can be attached to interaction types in order to enable the user for computation of add-hoc aggregates.
- *Join functions* are used for the construction of more complex interaction types on the basis of the given metaschema.

5.3 Media Types

Interaction objects are not yet sufficient for modelling web information systems. In order to allow the information content to be tailored to specific user needs and presentation restrictions according to channel or end-devices, we must extend interaction types. Taking together all extensions to interaction types will define media types.

Definition 5.7. A *media type* is an order-extended interaction type M together with an cohesion order $\preceq_M$ (or a set of promimity values) and a set of hierarchical versions $H(M)$.

We will define all these extensions in this section. Then a *media schema* is defined in the same way as a content schema replacing interaction types by media types. A database $\mathcal{D}$ over the underlying database schema $\mathcal{S}$ extends to a unique pre-site. Furthermore, we may extend $\mathcal{D}$ to all hierarchical versions $M' \in H(M)$ and all $M'' \succ_{M'} M'$ defined by the cohesion orders. This wide extension of $\mathcal{D}$ will be called a *site*.

Finally, we collect all media types $M_1, \ldots, M_k$ together with their hierarchical versions and types defined by the cohesion order such that $(u, v_i) \in \mathcal{D}(M_i)$ holds for a given URL u. The pair $(u, (M_1 : v_1, \ldots, M_k : v_k))$ will be called a *media object* in the site $\mathcal{D}$.

5.3.1 Adding Order

Since the interaction types are used to model the content of the WIS, order is important. Therefore, we claim that the set constructor should no longer appear in content expressions. Then we need an *ordering-operator* $ord_\leq$ which depends on a total order $\leq$ defined on a type t and is applicable to values v of type $\{t\}$. The result $ord_\leq(v)$ has the type $[t]$. We shall tacitly assume that ordering operators are used in the defining queries q_M. In this case we talk of an *order-extended* interaction type.

5.3.2 Adaptivity through Cohesion

Cohesion introduces a controlled form of information loss. Formally, we define a partial order $\leq$ on content data types, which extends subtyping:

- For any expression exp we have $exp \leq \mathbb{1}$;
- For records we have $(a_1 : exp_1, \ldots, a_m : exp_m) \leq (a_{\sigma(1)} : exp'_{\sigma(1)}, \ldots, a_{\sigma(1)} : exp'_{\sigma(n)})$ with injective $\sigma : \{1, \ldots, n\} \to \{1, \ldots, m\}$ and $exp_{\sigma(i)} \leq exp'_{\sigma(i)}$;
- For list and set expressions we have $\{exp\} \leq \{exp'\}$ (or $[exp] \leq [exp']$, respectively) iff $exp \leq exp'$ holds.

Definition 5.8. If $cont(M)$ is the content data type of a interaction type M and $sup(cont(M))$ is the set of all content expressions exp with $cont(M) \leq exp$, then a preorder $\preceq_M$ on $sup(cont(M))$ extending the order $\leq$ on content expressions is called an *cohesion preorder*.

Small elements in $sup(cont(M))$ with respect to $\preceq_M$ define information to be kept together, if possible. Clearly, $cont(M)$ is minimal with respect to $\preceq_M$. This enables a controlled form of information decomposition. If we want to decompose an interaction type or if we are forced to decompose according to user requirements or technical restrictions, then we may choose a minimal elements $t_1 \in sup(cont(M))$ with respect to $\preceq_M$ such that it satifies the representation requirements. Note that if we only provide a preorder, not an order, then there may be more than one such t_1.

Taking just t_1 instead of $cont(M)$ means that some information is lost, but this only refers to the first data transfer. When transferring t_1, we must include a link to a possible successor containing detailed information. In order to determine such a successor we can continue looking at all content types $t' \in sup(cont(M))$ with $t_1 \not\preceq_M t'$. These are just those containing the complimentary information that was lost. Again we can choose a least type t_2 among these t' with respect to $\preceq_M$ that requires not more than the available capacity. t_2 would be the desired successor.

Proceeding this way the whole communication is broken down into a sequence of suitable units $t_1, t_2, \ldots, t_n$ that together contain the information provided by the interaction type. Of course, the cohesion pre-order suggests that the relevance of the information decreases, while progressing with this sequence. The user may decide at any time that the level of detail provided by the sequence $t_1, \ldots, t_i$ is already sufficient for his/her needs.

Example 5.4. Let us ignore the defining query as well as operations. Just take the following simplified content type of a media type Home_Loan:

(type : *STRING*, conditions : *STRING*, interest :
{ (amount : *CARD*, rate : *FLOAT*) }).

So we could have the following instance:

$$(\&o_{11}, (\text{type} : \text{“Flexi”}, \ \text{conditions} : \text{“up to \$10,000 for 12 to 36 months”}, \\ \text{interest} : \{(\text{amount} : 3,000, \text{rate} : 8.2), (\text{amount} : 5,000, \text{rate} : 7.6), \\ (\text{amount} : 10,000, \text{rate} : 7.3)\})) \ .$$

We used the convention to denote URLs with a starting &. So, the URL for this *media object*

is $\&o_{11}$. Neglecting the inner set type we could define a cohesion order (not just a preorder) by

$$\begin{array}{l}(\text{type}:\ldots,\text{conditions}:\ldots,\text{interest}:\ldots)\\ \quad\preceq(\text{type}:\ldots,\text{conditions}:\ldots)\\ \quad\quad\preceq(\text{type}:\ldots,\text{interest}:\ldots)\\ \quad\quad\quad\preceq(\text{conditions}:\ldots,\text{interest}:\ldots)\\ \quad\quad\quad\quad\preceq(\text{type}:\ldots)\\ \quad\quad\quad\quad\quad\preceq(\text{conditions}:\ldots)\\ \quad\quad\quad\quad\quad\quad\preceq(\text{interest}:\ldots)\end{array}$$

Then assume that we can at most transfer $t_1 = (\text{type}:\ldots,\text{conditions}:\ldots)$. This would leave us with

$$\begin{array}{l}(\text{type}:\ldots,\text{conditions}:\ldots,\text{interest}:\ldots)\\ \quad\preceq(\text{type}:\ldots,\text{interest}:\ldots)\\ \quad\quad\preceq(\text{conditions}:\ldots,\text{interest}:\ldots)\\ \quad\quad\quad\preceq(\text{interest}:\ldots)\end{array}$$

Thus, the first part of the information that we transfer for the media object above would be

$$(\&o_{11}, (\text{type}: \text{“Flexi”},\ \text{conditions}: \text{“up to }\ldots\text{”},\ \text{more}: \&o_{11}^1))\ .$$

Next, we could choose $t_2 = (\text{type}:\ldots,\text{interest}:\ldots)$. The second bit of transferred information of the media object above would be

$$\begin{array}{l}(\&o_{11}^1, (\text{type}: \text{“Flexi”},\ \text{interest}: \{(\text{amount}: 3,000, \text{rate}: 8.2),\\ (\text{amount}: 5,000, \text{rate}: 7.6), (\text{amount}: 10,000, \text{rate}: 7.3)\}))\ .\end{array}$$

In fact, we could stop here, as further processing would not lead to more information. We would be left with just

$$(\text{type}:\ldots,\text{conditions}:\ldots,\text{interest}:\ldots) \quad\preceq\quad (\text{conditions}:\ldots,\text{interest}:\ldots)$$

Here, all maximal elements in $sup(cont(M))$ have disappeared, which indicates that no information has been left out.

An alternative to cohesion preorders is to use *proximity values*.

Definition 5.9. Let $exp_1, \ldots, exp_n$ be an antichain with respect to $\preceq$. A symmetric $(n \times n)$-matrix $\{p_{ij}\}_{1\leq i,j\leq n}$ with $0 \leq p_{ij} \leq 1$ is called a *set of proximity values*.

The antichain in the definition represents a possible split of the information content. The higher the proximity value, the more do we wish to keep the components together.

Applying proximity values $\{p_{ij} \mid 1 \leq i, j \leq n\}$ requires also to determine first the maximum amount of data that should be transferred. Then for each $X \subseteq \{1, \ldots, n\}$ determine its *weight*, i.e.,

$$w(X) \quad = \quad \sum_{i,j\in X, i<j} p_{i,j}$$

and its *greatest common subtype* $gcs(X)$, i.e., the greatest element $t_1 \in sup(cont(M))$ with $t_1 \leq exp_i$ for all $i \in X$. We choose the X with largest weight such that the $gcs(X)$ satisfies the representation requirements. Proceeding this way we also construct a sequence of content types $t_1, \ldots, t_n$, all appearing in the chosen anti-chain, such that together they provide the information of the media type. Same as for cohesion preorders the relevance of the information decreases, while progressing with this sequence, and the user may decide to stop the transfer, after receiving $t_1, \ldots, t_i$.

Example 5.5. Let us take the same media type as in Example 5.4. We choose the antichain $exp_1 = (\text{type} : \ldots)$, $exp_2 = (\text{conditions} : \ldots)$ and $exp_3 = (\text{interest} : \ldots)$ and the proximity values $p_{1,2} = 0.8$, $p_{1,3} = 0.5$ and $p_{2,3} = 0.1$. Then we get the following weights and greatest common subtypes:

X	$w(X)$	$gcs(X)$
{ 1 }	0	(type : ...)
{ 2 }	0	(conditions : ...)
{ 3 }	0	(interest : ...)
{ 1, 2 }	0.8	(type : ..., conditions : ...)
{ 1, 3 }	0.5	(type : ..., interest : ...)
{ 2, 3 }	0.1	(conditions : ..., interest : ...)
{ 1, 2, 3 }	1.4	(type : ..., conditions : ..., interest : ...)

Assuming that the whole information, i.e. $gcs(\{2,3\})$ has to be decomposed, the result would be the same sequence of types as in the previous example.

There is no general preference for cohesion preorders or proximity values. The major difference is that the proximity values provide an information split that is defined a priori, whereas the use of a cohesion (pre-)order would determine such a split. This means that cohesion (pre-)orders are more flexible for the price of being more costly with respect to the determination of the split.

5.3.3 Content Scaling through Hierarchies

Another possibility to tailor the information content of interaction types is to consider dimension hierarchies as in OLAP systems. Flattening of dimensions results in information growth, its converse in information loss. Such a hierarchy is already implicitly defined by the component or link structures, repectively.

Formally, flattening can be defined by operators flat_r. If E' is a component of E corresponding to the role r, we may replace E by $\mathit{flat}_r(E)$ defined as follows:

$$comp(\mathit{flat}_r(E)) = comp(E) - \{r : E'\} \cup comp(E') \text{ and}$$
$$attr(\mathit{flat}_r(E)) = attr(E) \cup attr(E')$$

The new key is $id(E) - \{r : E'\} \cup id(E')$ in the case $(r : E') \in id(E)$; it is $id(E)$ otherwise. We may extend this definition and flatten occurrences of links $\ell : M'$ in content data types cont(M'). We simply substitute cont(M') for $\ell : M'$. The resulting interaction type will be denoted as $\mathit{flat}_\ell(M)$.

The converse operator $raise_P$ for $P \subseteq comp(E) \cup attr(E)$ is defined analogously. Any subset $P \subseteq comp(E) \cup attr(E)$ allows to replace E by $raise_P(E)$ and a new database type E_{new}

defined as follows:

$$\begin{aligned}
comp(raise_P(E)) &= comp(E) - P \cup \{r_{\text{new}} : E_{\text{new}}\} \quad , \\
attr(raise_P(E)) &= attr(E) - P \qquad , \\
id(raise_P(E)) &= \begin{cases} id(E) - P \cup \{r_{\text{new}} : E_{\text{new}}\} & \text{for } P \cap id(E) \neq \emptyset \\ id(E) & \text{else} \end{cases} \\
comp(E_{\text{new}}) &= P \cap comp(E) \quad , \\
attr(E_{\text{new}}) &= P \cap attr(E) \qquad \text{and} \\
id(E_{\text{new}}) &= \begin{cases} id(E) & \text{for } id(E) \subseteq P \\ P & \text{else} \end{cases} \quad .
\end{aligned}$$

Again, this may be generalized to $raise_{exp}(M)$ for a content expression occurring within $cont(M)$. This will introduce a new link expression replacing exp.

Definition 5.10. For an interaction type M let $\bar{H}(M)$ be the set of all interaction types arising from M by applying a sequence of flat-operations or their converses to interaction types or underlying database types. A *set of hierarchical versions* of M is a finite subset $H(M)$ of $\bar{H}(M)$ with $M \in H(M)$. Each cohesion order $\preceq_M$ on M induces an cohesion order $\preceq_{M'}$ on each element $M' \in H(M)$.

5.4 Modelling Contextual Information

Within the framework of media types the problem of contextual information is not yet well supported. In [15] it has been emphasised that "escort information" is required in each scene. The problem is to provide in a condensed form the information the user has already seen since entering the WIS. This means to place each scene of the story space into a context. The work in [15] claims that introducing a subtyping mechanism between media types is useful for modelling escort information, thus contexts. However, subtyping is a static concept, which is useful for tree-like navigation structures, whereas in some cases the more complex navigation and story structures will require a dynamic solution. We therefore ask, whether the work on contextual information bases (CIBs) [3, 45, 46] can be exploited for enhancing our methodology. We generalise and tailor the CIB-approach in order to integrate it with the theory of media types. This provides the formal conceptual means for context modelling.

According to the theory of CIBs a context is a set of objects, each having several names, and each of these names may be coupled with a reference to another context. There may be names for objects that are not referencing to other contexts. Here, the term "object" is used in the sense of "object identifier", i.e. a unique abstract handle to identify objects.

More formally, a *context* C is a finite set of triples (o_i, n_i, r_i), where o_i is an object identifier, i.e. a value of some base type ID, n_i is a name, i.e. a value of type $STRING$, and r_i is either a reference $\rightarrow C'$ to a context C' or nil, the latter one indicating that there is no such reference.

We write $C = \{n_1 : o_1 \rightarrow C_1, \ldots, n_\ell : o_\ell \rightarrow C_\ell\}$. If there is no reference for the i'th name, i.e. we have (o_i, n_i, nil) we simply omit $\rightarrow C_i$ and write $n_i : o_i$.

The idea of working with contextual information bases is that a user queries them and thus retrieves objects. In order to describe these objects in more details she or he accesses the context(s) of the object, which will lead — by following the references — to other objects. In addition, a particular information encoded by the name is associated with each of these

references. The work in [46] describes a path query language for contextual information bases. Most important for our problem are the following macros of this language:

- The macro `look-up`(C, n) takes two input parameters. The first one is the name of a context C. The second one is a name n, i.e., a value of type *STRING*. The macro returns name paths $n_i = n_i^0, \ldots, n_i^{k_i}$ starting from context C and ending in n, i.e. $n_i^{k_i} = n$.
- The macro `cross-ref`(p, C) also takes two input parameters. Here the first one is a name path $p = p^0, \ldots, p^\ell$. The second one is the name of a context C. The macro returns name paths $n_i = n_i^0, \ldots, n_i^{k_i}$ starting from context C and ending in the name specified by p, i.e., $p^\ell = n_i^{k_i}$.

Let us now bring together media types and contextual information bases. The obvious questions are:

- What are the objects that are required in contextual information bases, if we are given media types?
- What are the references that are required in contextual information bases?
- Is it sufficient to have names for describing objects in a context or should these be replaced by something else?

The natural idea for generalising the notion of object in contexts is to choose the media objects. Concentrate on the raw media objects first. Evaluating the defining query for a raw media type M leads to a set $\{(u_1, v_1), \ldots, (u_n, v_n)\}$ of raw media objects. Recall that the u_i are values of type *URL*, whereas the v_i are values of the representing type t_M. As these URL-values are unique, they identify the raw media objects, and thus can be used as surrogates for them in the notion of context.

This answers our first question. The objects are the media objects. The object identifiers needed in the contexts are the (abstract) URLs of these media objects.

As we want to have access to path information, we may want to reference back to the various media objects that we have encountered so far. These media objects are placed in several contexts, one of which is the right one corresponding to our path. However, we may also have different references, which lead to different contexts. So, the contexts we asked for in the second questions are just the contexts for the media objects.

As to the third question, we definitely want to have more information than just a name. Fortunately, the theory of media types is already based on the assumption of an underlying type system. Thus, we simply have to replace the names by values of any type allowed by the type system. Having defined such extended contexts, the query macros such as `look-up` and `cross-ref` would allow to traverse back a path in the story board and to explore alternative access paths, respectively.

However, one important aspect of media types is the use of classification abstraction. Conceptually, we do not define a set of media objects, but we generate them via queries defined on some underlying database schema. Therefore, we also need a conceptual abstraction for contexts.

In order to obtain this conceptual abstraction, we assume another base type *Context*, the values of which are context names. Instead of this, we could take the type *URL*, but in order to avoid confusion we use a new type.

A *context* consists of a name C, i.e. a value of type content, a type t_C and a defining query q_C, which must be defined on the media schema, i.e., the set of media types, such that

$$(\{(\text{object} : \mathit{URL}, \text{value} : t_C, \text{reference} : \mathit{Context})\}, q_C)$$

defines a view. Thus, executing the query q_C will result in a set of triples (u_i, v_i, r_i), where u_i is the URL of a media type, v_i is a value of type t_C, and r_i is the name of a context. If this context is undefined, this is interpreted as no reference for this object in this context. Note that in particular this definition of context leads to views over views.

Let us finally reconsider the "old" definition of media types in [15], which included supertypes. In this case, all the supertypes are media types, thus depend on defining queries. They could be treated as queries defining a context. Thus, a media object of type M would be in as many contexts as there are supertypes of M. However, there are two important differences:

- In contextual information bases we want to select one context to obtain the information about the path, whereas the supertyping assumes that the combination of all supertypes defines the required context.
- If the supertypes are treated as if they are defining contexts, then there will be no references from their objects to other contexts. This omits the possibility of navigating through contexts.

Alternatively, we could take all the defining queries of supertypes of M together to define a context. Then each media object would belong to exactly one context, and as before there would not be any references to other contexts. Thus, supertyping turns out to be a simplified, static version of context modelling.

Chapter 6

Conclusion

In this article we presented two central parts of a conceptual modelling approach to web information systems. The first of these parts deals with storyboarding, which addresses the following problems:

- modelling the path a potential user may take through the WIS including the actions performed along such a path;
- modelling the information portfolios of users, i.e. the information needed at a particular location in the WIS;
- modelling obligations and rights of users in a particular role;
- modelling user profiles and preference rules that arise from them;
- modelling the intention of users;
- modelling tasks that users will perform either individually or cooperatively while using the WIS.

Furthermore, we applied Kleene algebras with tests for propositional reasoning about the storyboard. In particular, we showed how to personalise a WIS to preferences and intentions of users.

The second part dealt with media types for the support of content and functionality at an individual location in the WIS. This addresses the following problems:

- modelling views on underlying databases in a way that the content presented to a WIS user will be expressed by such views;
- extend the views in such a way that not only the content at a particular location in the WIS is generated out of a database, but also navigation links are created by these views;
- further extend the views by operations in order to model the functionality offered to a user at a particular location in the WIS;
- extend the views by supporting adaptivity to user preferences, limitations of channels, and a variety of end-devices;
- extend the views by hierarchical versions that support different granularities for the content and permit to switch between them;
- modelling context for locations by taking condensed path information into account.

We omitted to discuss presentation options, i.e. style options that can be added to the media types and used for the generation of pages.

The most challenging future research direction to be continued is to approach extensions to the propositional reasoning about the storyboard and to widen the scope towards an inclusion

of deontic constraints that are used to model obligations and rights. Furthermore, we think of widening the reasoning scope also by switching from propositional reasoning with Kleene algebras with tests to general dynamic logic. This means to take also the updates of the underlying databases into consideration.

Bibliography

1. Abiteboul, S., Buneman, P., and Suciu, D. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.
2. Agrawal, R., Gupta, A., and Sarawagi, S. Modeling multidimensional database. In *Proc. Data Engineering Conference, Birmingham* (1997), pp. 232–243.
3. Akaishi, M., Spyratos, N., and Tanaka, Y. A component-based application framework for context-driven information access. In *Information Modelling and Knowledge Bases*, H. Kangassalo, Ed., vol. XIII. IOS Press, Amsterdam, 2002, pp. 254–265.
4. Atzeni, P., Gupta, A., and Sarawagi, S. Design and maintenance of data-intensive web-sites. In *Proceeding EDBT'98*, vol. 1377 of *LNCS*. Springer-Verlag, Berlin, 1998, pp. 436–450.
5. Baresi, L., Garzotto, F., and Paolini, P. From web sites to web applications: New issues for conceptual modeling. In *ER Workshops 2000*, vol. 1921 of *LNCS*. Springer-Verlag, Berlin, 2000, pp. 89–100.
6. Binemann-Zdanowicz, A., Kaschek, R., Schewe, K.-D., and Thalheim, B. Context-aware web information systems. In *Conceptual Modelling 2004 – First Asia-Pacific Conference on Conceptual Modelling* (Dunedin, New Zealand, 2004), S. Hartmann and J. Roddick, Eds., vol. 31 of *CRPIT*, Australian Computer Society, pp. 37–48.
7. Binemann-Zdanowicz, A., Thalheim, B., and Tschiedel, B. Flexible e-payment based on content and profile in the e-learning system DaMiT. In *Proceedings of eCoMo 2003* (2003), H. C. Mayr and W.-J. Van den Heuvel, Eds.
8. Binemann-Zdanowicz, A., Thalheim, B., and Tschiedel, B. Logistics for learning objects. In *Proceedings of eTrain 2003* (2003).
9. Bonifati, A., Ceri, S., Fraternali, P., and Maurino, A. Building multi-device, content-centric applications using WebML and the W3I3 tool suite. In *ER Workshops 2000*, vol. 1921 of *LNCS*. Springer-Verlag, Berlin, 2000, pp. 64–75.
10. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., and Matera, M. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, San Francisco, 2003.
11. Ceri, S., Fraternali, P., and Matera, M. Conceptual modeling of data-intensive web applications. *IEEE Internet Computing 6*, 4 (2002), 20–30.
12. Ceri, S., Fraternali, P., and Paraboschi, S. Data-driven, one-to-one web site generation for data-intensive applications. In *Proceedings of the Conference on Very Large Databases (VLDB 1999)*. IEEE Computer Society, 1999, pp. 615–626.
13. De Troyer, O. Designing well-structured websites: Lessons learned from database schema methodology. In *Conceptual Modeling- ER'98*, T. Ling, S. Ram, and M. Lee, Eds., vol. 1507 of *LNCS*. Springer-Verlag, 1998, pp. 51–64.
14. Düsterhöft, A., and Thalheim, B. SiteLang: Conceptual modeling of internet sites. In *Conceptual Modeling – ER 2001*, H. S. K. et al., Ed., vol. 2224 of *LNCS*. Springer-Verlag, Berlin, 2001, pp. 179–192.
15. Feyer, T., Kao, O., Schewe, K.-D., and Thalheim, B. Design of data-intensive web-based information services. In *Proceedings of the 1st International Conference on*

Web Information Systems Engineering (WISE 2000), Q. Li, Z. M. Ozsuyoglu, R. Wagner, Y. Kambayashi, and Y. Zhang, Eds. IEEE Computer Society, 2000, pp. 462–467.

16. Feyer, T., Schewe, K.-D., and Thalheim, B. Conceptual modelling and development of information services. In *Conceptual Modeling – ER'98*, T. Ling and S. Ram, Eds., vol. 1507 of *LNCS*. Springer-Verlag, Berlin, 1998, pp. 7–20.
17. Feyer, T., and Thalheim, B. E/R based scenario modeling for rapid prototyping of web information services. In *Advances in Conceptual Modeling*, P. P.-S. Chen, Ed., vol. 1727 of *LNCS*. Springer-Verlag, 1999, pp. 253–263.
18. Fraternali, P. Tools and approaches for developing data-intensive web applications: A survey. *ACM Computing Surveys 31*, 3 (1999), 227–263.
19. Fraternali, P., and Paolini, P. A conceptual model and a tool environment for developing more scalable, dynamic and customizable web applications. In *Proceedings EDBT'98*, vol. 1377 of *LNCS*. Springer-Verlag, 1998, pp. 422–435.
20. Gädke, M., and Turowski, K. Generic web-based federation of business application systems for e-commerce applications. In *EFIS 1999*. 1999, pp. 25–42.
21. Garzotto, F., Paolini, P., and Schwabe, D. HDM - a model-based approach to hypertext application design. *ACM ToIS 11*, 1 (1993), 1–26.
22. Gehrke, D., and Turban, E. Determinants of successful website design: Relative importance and recommendations for effectiveness. In *Proceedings HICSS'99*. Springer-Verlag, 1999.
23. Germán, D. M., and Cowan, D. D. Formalizing the specification of web applications. In *Advances in Conceptual Modeling*, P. P.-S. Chen, Ed., vol. 1727 of *LNCS*. Springer-Verlag, 1999, pp. 281–292.
24. Hoare, C. A. R. An axiomatic basis for computer programming. *Communications of the ACM 12*, 10 (1969), 576–580.
25. Kaschek, R., Matthews, C., Schewe, K.-D., and Wallace, C. Analyzing web information systems with the Abstraction Layer Model and SiteLang. In *Proceedings of the Australasian Conference on Information Systems (ACIS 2003)* (2003).
26. Kaschek, R., Schewe, K.-D., Thalheim, B., and Zhang, L. Integrating context in conceptual modeling for web information systems. In *Proceedings WES'03* (2003).
27. Kaschek, R., Schewe, K.-D., Wallace, C., and Matthews, C. Story boarding for web information systems. In *Web Information Systems*, D. Taniar and W. Rahayu, Eds. IDEA Group, 2004, pp. 1–33.
28. Kozen, D. On Kleene algebra and closed semirings. In *Mathematical Fundamentals of Computer Science* (1990), pp. 26–47.
29. Kozen, D. A completeness theorem for Kleene algebras and the algebra of regular events. *Information & Computation 110*, 2 (1994), 366–390.
30. Kozen, D. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems 19*, 3 (1997), 427–443.
31. Kozen, D. On Hoare logic and Kleene algebra with tests. In *Logic in Computer Science* (1999), pp. 167–172.
32. Kozen, D., and Smith, F. Kleene algebra with tests: Completeness and decidability. In *Computer Science Logic* (1996), pp. 244–259.
33. Lowe, D., Henerson-Sellers, B., and Gu, A. Web extensions to UML: Using the MVC triad. In *Conceptual Modeling – ER 2002*, S. Spaccapietra, S. T. March, and Y. Kambayashi, Eds., vol. 2503 of *LNCS*. Springer-Verlag, 2002, pp. 105–119.
34. Ludäscher, B., and Gupta, A. Modeling interactive web sources for information mediation. In *Advances in Conceptual Modeling*, P. P.-S. Chen, Ed., vol. 1727 of *LNCS*. Springer-Verlag, 1999, pp. 225–238.

35. Mecca, G., Merialdo, P., and Atzeni, P. ARANEUS in the era of XML. *IEEE Data Engineering Bullettin* (1999).
36. Rossi, G., Garrido, A., and Schwabe, D. Navigating between objects: Lessons from an object-oriented framework perspective. *ACM Computing Surveys 32*, 1 (2000).
37. Rossi, G., Schwabe, D., and Lyardet, F. Web application models are more than conceptual models. In *Advances in Conceptual Modeling*, P. C. et al., Ed., vol. 1727 of *LNCS*. Springer-Verlag, Berlin, 1999, pp. 239–252.
38. Schewe, K.-D. Querying web information systems. In *Conceptual Modeling – ER 2001*, H. S. Kunii, S. Jajodia, and A. Sølvberg, Eds., vol. 2224 of *LNCS*. Springer-Verlag, 2001, pp. 571–584.
39. Schewe, K.-D., Kaschek, R., Wallace, C., and Matthews, C. Modelling web-based banking systems: Story boarding and user profiling. In *Advanced Conceptual Modeling Techniques: ER 2002 Workshops* (2003), vol. 2784 of *LNCS*, Springer-Verlag, pp. 427–439.
40. Schewe, K.-D., and Schewe, B. Integrating database and dialogue design. *Knowledge and Information Systems 2*, 1 (2000), 1–32.
41. Schewe, K.-D., and Thalheim, B. Modeling interaction and media objects. In *Natural Language Processing and Information Systems: 5th International Conference on Applications of Natural Language to Information Systems, NLDB 2000*, M. Bouzeghoub, Z. Kedad, and E. Métais, Eds., vol. 1959 of *LNCS*. Springer-Verlag, Berlin, 2001, pp. 313–324.
42. Schewe, K.-D., and Thalheim, B. Structural media types in the development of data-intensive web information systems. In *Web Information Systems*, D. Taniar and W. Rahayu, Eds. IDEA Group, 2004, pp. 34–70.
43. Schwabe, D., and Rossi, G. An object oriented approach to web-based application design. *TAPOS 4*, 4 (1998), 207–225.
44. Schwabe, D., Rossi, G., and Barbosa, S. Systematic hypermedia design with OOHDM. In *Proc. Hypertext '96*. ACM Press, 1996, pp. 116–128.
45. Teodorakis, M., Analyti, A., Constantopoulos, P., and Spyratos, N. Context in information bases. In *Proceedings CoopIS '98* (1998), pp. 260–270.
46. Teodorakis, M., Analyti, A., Constantopoulos, P., and Spyratos, N. Querying contextualized information bases. In *Proceedings ICT & P '99* (1999).
47. Thalheim, B. *Entity-Relationship Modeling: Foundations of Database Technology*. Springer-Verlag, 2000.
48. Thalheim, B., and Düsterhöft, A. The use of metaphorical structures for internet sites. *Data & Knowledge Engineering 35* (2000), 161–180.
49. Van Duyne, D. K., Landay, J. A., and Hong, J. I. *The Design of Sites*. Addison-Wesley, Boston, 2002.