The Challenges in Web Information Systems Development

Klaus-Dieter Schewe
Massey University, Information Science Research Centre
& Department of Information Systems
Private Bag 11222, Palmerston North, New Zealand
k.d.schewe@massey.ac.nz

Abstract: The development of web information systems has led to new challenges regarding the scalability and expressiveness of methods. In particular, as systems become large, it becomes decisive to guarantee consistency and integrity of designs. The paper introduces the challenges in a step-by-step way and indicates how to cope with them. First, the conceptual model of a WIS as a collection of media types, i.e. extended views, is introduced. In this context the adaptivity to users, channels and devices will also be discussed. Secondly, WIS development is discussed on a higher level of abstraction dealing with tasks, roles, user profiles and storyboards. At this level equational reasoning can be applied. Finally, both levels of abstraction are combined leading to further challenges by using dynamic and deontic logics.

1 Introduction

The development of web information systems has led to new challenges regarding the scalability and expressiveness of methods. In particular, as systems become large, it becomes decisive to guarantee consistency and integrity of designs. In this article we introduce the challenges in a step-by-step way and indicate how to cope with them.

The starting point is the commonly accepted problem triplet according to which WIS design addresses the content, navigation structure and presentation issues. This can be approached in a naive way, but this will not lead very far. Thus the first step toward a more sophisticated development methodology consists in abstracting from the content using views on some underlying databases. In doing so data modelling becomes an important part of WIS design. In a second step the navigation structure can be integrated into the views, which copes with the insufficiency of treating navigation as an add-on, but adds the problem that view construction requires more powerful query languages.

The next step addresses the diversification of the views by allowing hierarchies, order, measures, etc. In the same line of thought views can be further extended by presentation options. Then a WIS can be abstractly described by a collection of extended views. In order to cope with various user preferences, and restrictions arising from end-devices or access channels it is further desirable to provide adaptivity. We discuss how cohesion can solve this challenge.

If WISs are to become active, users have to be offered more than navigation. This challenge can be dealt with by providing operations, which can be associated with the extended views. Taking all these together we obtain the conceptual model of *media types* for WIS design [ST05].

However, media types already reside on a very fine level of detail. In order to fully capture the intentions and the anticipated usage of a WIS we provide abstraction through story spaces and show how story algebras can be used to reason about story spaces. In particular, they will enable personalisation already on a high level of abstraction.

As preferences relate to user profiles we briefly discuss user profiles. Then we extend the usage model by roles and tasks focussing on the goals of the WIS and which classes of users have to cooperate to achieve these goals. The role model immediately leads to deontic constraints expressing obligations and rights, which can be modelled by deontic logic.

In doing so, the challenge arises to reason about consistency, personalisation with respect to preferences and goals and user cooperation also on the level of media types. In doing so, we need proof obligations that can be expressed in dynamic and deontic logic. We will indicate these proof obligations.

A final challenge concerns compositionality, i.e. that user profiles should be formulated within the logic. However, this adds the request to deal with non-monotonic reasoning.

2 Related Work

Since the late nineties a lot of methodologies for the design of WIS have been published such as ARANEUS [AGS98], OOHDM [SR98], WebML [CFB+03], HERA [HBFV03], WSDM [DL98], and our Co-Design Approach [ST05]. By now there seems to be a common understanding that using extended views to model WISs is a good idea. Nevertheless there are a lot of differences between the various approaches. For instance, in WebML or WSDM navigation links are added, once the views have been defined, whereas in the co-design approach the navigation structure is generated as part of the view construction, which, however, requires much more sophistication for the query language. Furthermore, the extended views in the co-design approach also cover operations.

While extended views capture in one way or the other aspects of content, functionality (at least navigation) and presentation, the intention and usage of a WIS is only addressed explicitly in WSDM [DL98] and the Co-Design Approach [ST05], and only the latter one provides an explicit model for contextual information, which is based on an integration of media types with the approach to contextual information basis [TACS98].

With respect to personalisation, i.e. customisation to preferences of users, the commonly adapted approach is to label parts of a WIS specification with eligible user types. However, the co-design approach applies propositional reasoning techniques instead exploiting Kleene algebras with tests (KATs). KATs have been introduced in [Ko97], and since then they have been intensively studied. In addition, another line of propositional reasoning

about WISs on a high level of abstraction using propositional deontic logic to reason about obligations and rights is supported.

In [Sc04] a first step towards reasoning on the level of media types has been made. This exploits higher-order dynamic logic [HKT00] and deontic logic [El97] to set up proof obligations for personalisation and consistency.

3 Content and Functionality Modelling

Basically, if a user encounters a WIS, s/he navigates through a collection of web pages. Thus, the ultra-naive approach would be to write HTML- or XHTML-code for each of these pages. As soon as the system becomes larger, i.e. it exceeds a handfull of pages, most of which contain similar content, this approach is no longer feasible. Thus, the first challenge is to seek database support for content modelling: set up a database schema and model the content of pages by views. So, in an abstract sense each page corresponds to an object, called *media object* in [ST05]. Formally, a media object is no more than a pair (i, v) consisting of an abstract identifier, e.g. a URI, and a value v of some type.

There are two different ways to obtain such media objects through views: Build a separate view for each media object. That is, if $\mathcal{S}$ is the underlying database schema, then use a query q, which transforms an instance of $\mathcal{S}$ into a value v of some type t_q. In addition, create an identifier to obtain the media object. Alternatively, build a view with a target schema $\mathcal{S}_V$ and a query q_V that transforms each instance of $\mathcal{S}$ into an instance of $\mathcal{S}_V$. So we obtain a set $\{v_1, \ldots, v_n\}$ of values. Creating identifiers results in a set of media objects.

The co-design approach uses the the second alternative, which gives a set of possible media objects of some type, while the actual media object that will be presented to a user, is determined at run-time.

With respect to content modelling some methods stop here, take the collection of views, preferably in the second sense, and manually associate navigation links. Again, if the number of navigation links becomes large, the approach is not feasible anymore, so we discover a second scalability challenge. Worse than scalability, the navigation links should be links between individual media objects, i.e. the actual links are determined on instance level. They may even become optional.

In order to cope with this problem, the co-design approach adopts an idea from object oriented databases, and simply considers the collection of all possible media objects. By incorporating the identifier j of some media object (j, w) into the value v of some other media object (i, v), we set up a link between these two media objects. In fact, this view was already present from the very beginning in semi-structured data [ABS00].

The challenge is then that the views that are used to define the media objects are no longer independent from each other. Hence the query language must be powerful enough to create complex values that contain identifiers, while at the same time these identifiers appear in the result of other queries. In [ST05] it has been shown that IQL-like languages [AK89]

or a rather complex query algebra can cope with this challenge.

Summarising this approach to link generation through views, we obtain the notion of an interaction type:

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 *URI* for all pairs $\ell : M'$.

Finite 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 URIs 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 $\mathcal{D}$.

Note that this definition of interaction type also specifies operations. This aims at addressing another challenge, the one of providing *active* WISs, where a user does not just navigate between media objects – to be realised by pages – but performs also actions, which may involve entering data into the system. Thus, interaction types already address functionality beyond navigation.

Formally, 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.

Let us finally say a few words about the implementation of content schemata. Following [SS00] it is no problem to define and implement views, and to maintain the active set of interaction objects at any time. This has been used in industry projects, though not yet in the complicated setting that links have to be maintained. So, we get another challenge here. Yet, interaction objects can be mapped onto XML documents, while the operations give rise to script specifications. We will see later that this rather simple view of the implementation has to be enhanced, when the notion of interaction object will be extended to media objects.

Once interaction types have been defined, we also have to take care of their presentation. Doing this in an efficient way presents another challenge to WIS design. According to experiences made in several large web site development projects it turns out that page layout is rather standardised. In particular, we may exploit *web page grids*, which result a tiling of the envisioned pages. Then we may associate *style options* with the interaction types. Basically, a style option takes content data type $cont(M)$, breaks it into components, and associates each of these components with some tile in the grid. Furthermore, there may be constant layout elements such as pictures associated with grid components. Finally, a style option defines how the component is to be represented. Taking all these together we have to design XSLT templates that can be used to translate the XML documents resulting from interaction objects to actual HTML-pages.

4 Hierarchies and Adaptivity

While the use of interaction types abstracts nicely from the page level and provides adequate means for coupling WISs with database technology, it is still a rather static approach, as the information contained in an interaction object is independent from preferences and changing needs of users. Similarly, it still ignores the fact that a WIS may be accessed through various channels including TV and mobile nets, and that users may use a lot of different devices such as PCs, PDAs, cellphones, TVs, etc. At least two challenges arise from this diversity:

We have to allow for a presentation of the data at different levels of granularity, and allow a user to switch between different presentations. For this we use *hierarchies* as an extension to interaction types. We have to be support *adaptivity* to users, channels and end-devices, i.e. that in accordance with identified needs the system will adapt the content automatically. For this we use *cohesion* as an extension to interaction types.

Both extensions exploit the concept of subtyping, which is defined by a partial order $\leq$ on the set of all types. In particular, if $cont(M)$ is the content data type of an interaction type M, we may consider the set $\mathcal{S}(cont(M))$ of all its supertypes. Then $cont(M)$ is the smallest element in $\mathcal{S}(cont(M))$.

For hierarchies we select a subset $\mathcal{H}(cont(M)) \subseteq \mathcal{S}(cont(M))$ with $cont(M) \in \mathcal{H}(cont(M))$, which forms a tree with respect to the order $\leq$. We call $\mathcal{H}(cont(M))$ a *set of hierarchical versions*. Then each interaction object of type M gives rise to several versions, one for each element in $\mathcal{H}(cont(M))$. The data content results from projections that are induced by the subtyping relationship. Formally, if (i, v) represents the interaction object as defined in the previous section, then v is a value of type $cont(M)$. For $t \in \mathcal{H}(cont(M))$ we have $cont(M) \leq t$, which induces a projection $\pi_t^{cont(M)} : dom(cont(M)) \to dom(t)$, so the hierarchical version of (i, v) with respect to t is represented by the pair $(i, \pi_t^{cont(M)}(v))$.

Then a user may walk up and down the hierarchy defined by $\mathcal{H}(cont(M))$. Moving upwards means to present fewer information, which is achieved by applying a projection operation as just described. Moving downwards means to select presentation that is richer in information. By default we assume that the hierarchical version that is to be presented first, is defined by $cont(M)$. Alternatively, we may select an *initial type* $I(M) \in \mathcal{H}(cont(M))$ and present $(i, \pi_{I(M)}^{cont(M)}(v))$ instead of (i, v).

For cohesion we can either provide *proximity values* or a *cohesion preorder* [ST05]. In both cases the information provided by an interaction object (i, v) will be divided up into a sequence $(i_1, v_1), \ldots, (i_n, v_n)$ such that each v_i contains a link to a successor i_{i+1}, each v_i contains information of higher relevance than v_{i+1}, the cohesion between the information represented by v_i is larger than the cohesion between v_i and v_{i+1}, and $v_1, \ldots, v_n$ jointly represent the same information as v.

The difference between the two approaches is that for proximity values the types of the v_i are fixed a priori, whereas in the case of cohesion preorders they are determined at run-time, i.e. when the interaction object is actually created and activated.

Let us concentrate only on proximity values. So, take a maximal anti-chain $t_1, \ldots, t_m$

in $\mathcal{S}(cont(M))$. Then for each interaction object (i, v) of type M the values $w_i = \pi_{t_i}^{cont(M)}(v)$ represent a maximal split satisfying the last of the desired properties. Now define *proximity values* $p_{i,j}$ for $1 \leq i < j \leq m$. These values indicate the cohesion between the values w_i and w_j, i.e. they define, to what degree it is desired to keep w_i and w_j together.

Then for each subset $S \subseteq \{1, \ldots, m\}$ compute $p_S = \sum_{i,j \in S, i<j} p_{i,j}$ and a value w_S that joins all w_i with $i \in S$. Among these sets S choose the one with the largest value p_S, provided w_S satisfies the user preferences and the size restrictions arising from channels and end-devices. This value w_S will define v_1. Then proceed in the same way with $\{1, \ldots, m\} - S$ to determine v_2, etc., which will finally result in the desired sequence. Adding the links does not cause any problems.

An interaction type that is extended by hierarchies, cohesion, style options, measures and ordering is finally called a *media type*. Thus, in the co-design approach the conceptual model of a WIS consists of a *media schema*, i.e. a set of media types. A more formal presentation is contained in [ST05].

5 Storyboarding

While a media schema is an adequate specification of a WIS that abstracts from the page level, it still resides at a level of abstraction that is far too detailed to start development on this level, in particular, if the WIS is expected to become large. Similarly, in "traditional" information systems development conceptual data and interface modelling is embedded in an analysis process that starts from business processes and the plan how the intended system is to be used. The analogue for WISs development with the co-design approach is *storyboarding*.

Thus, we have to face the challenge to specify the usage of the WIS, i.e. to build high-level models how users will navigate through the system and which actions they will perform. This will lead to the first component of storyboarding, the *story space*. In addition, we would like to anticipate the behaviour of the users and as much as possible customise the system to their preferences and goals. This so-called *personalisation* depends on *user profiles*, thus another challenge is to define these profiles, to associate preference rules with them, and to solve the personalisation problem.

In order to model story spaces we may view a WIS is a set of abstract locations. 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*. Thus, 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. The whole multi-graph is then called the *story space*.

A *story* is a path in 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.

Looking at the whole story space from a different angle, we may concentrate on the flow of actions. 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, we must allow choice between actions, and actions can be iterated.

This leads to a *story algebra*. Thus, we can describe a story space by an element of a suitable story algebra. Such algebras have to be defined as being many-sorted in order to capture the association of actions with scenes. Furthermore, we should reserve parallelism to actions on different scenes, in which case we can replace parallel actions by commuting sequential actions. As shown in [ST05] this gives rise to Kleene algebras with tests.

Using equations we may exploit the axioms of KATs to reason about story spaces. However, we do not just want to derive equations from the KAT axioms, which would mean to exploit the equational theory of KATs, but we want to derive such equations under the assumption of other equations that describe user preferences and other constraints for the story space expression. In particular, we obtain the following types of equations:

- An equation of the form $\varphi(\alpha + \beta) = \varphi\alpha$ is a *preference rule*, as it expresses that a user, who may choose between between actions α and β under the condition φ, will prefer α. The special case $\varphi = 1$ expresses an unconditional preference.
- An equation of the form $\alpha = \varphi\alpha$ expresses that the condition φ is a *precondition* for the action α.
- An equation of the form $\alpha = \alpha\varphi$ expresses that the condition φ is a *postcondition* for the action α.
- An equation of the form $\alpha\varphi = \varphi\alpha$ is an *invariance rule*, as it expresses that the condition φ (and so its negation $\bar{\varphi}$) is invariant under the action α.
- An equation of the form $\varphi\psi = 0$ is an *exclusion rule*, as it expresses that the conditions φ and ψ exclude each other.
- An equation of the form $\alpha\beta = \beta\alpha$ expresses *parallelism*, i.e. the order of the actions α and β is irrelevant, hence they behave as if they were executed in parallel.

Then personalisation can be formalised by the following optimisation task.

> Given a process $p \in K$ that represents a story space, and a set Σ of equations on K that represents (among other constraints) user preferences and a postcondition $\psi \in B$, we look for a minimal process $p' \in K$ such that $\Sigma \models p\psi = p'\psi$ holds.

That is, the resulting process p' is a *personalisation* of p according to the *user intention* formalised by ψ. A more detailed discussion of story space personalisation including meaningful examples taken from on-line loan applications was presented in [ST05].

6 User Profiles and Roles

The preference rules used in the previous section depend on the profile of the user. Thus, the challenge is to define these profiles. For this we may ask which properties characterize a user and provide values for each of these properties. Each combination of such values defines a profile [ST05], but usually the behaviour for some of these profiles is the same.

The dimensions used in user profiles depend on the application. Sources for such dimensions can be 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, and 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*. For each dimension $\delta \in \Delta$ we assume to be given a *domain* $dom(\delta)$. If $\Delta = \{\delta_1, \ldots, \delta_n\}$ is a set of user dimensions, the *set of user profiles* over Δ is $gr(\Delta) = sc(\delta_1) \times \cdots \times sc(\delta_n)$. A *user type* over Δ is a subset $U \subseteq gr(\Delta)$.

However, in the light of the approach to personalisation based on equational reasoning a new challenge arises. Instead of first modelling user profiles and types, then assigning preference rules to them, we might directly use the information about the user type as antecedent for the preference rules. In this way we obtain implications.

The advantage is that we may dispense with classifying users, as the purpose of user profiling is to obtain preference rules, and these can be captured without classification. However, the challenging new problem is the increased complexity and non-monotonicity of the reasoning process. So far, this has not yet been investigated.

A somehow related problem arises from modelling user *roles* [ST05]. Basically, the presence of roles indicates different purposes of the WIS. 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. A role determines the set of actions that a user 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 a user in a particular role has to do. A *right* specifies what a user in a particular role is permitted to do. Both obligations and rights together lead to complex deontic integrity constraints. Co-design uses the following logical language $\mathcal{L}$ for this purpose. In this logic $\mathbf{O}\ do(r, \alpha)$ means that a user with role r is obliged to perform action α, $\mathbf{P}\ do(r, \alpha)$ means that a user with role r is permitted to perform action α, and $\mathbf{F}\ do(r, \alpha)$ means that a user with role r is forbidden to perform action α.

7 Reasoning about WISs

The high-level abstraction through storyboarding that we discussed in the previous two sections leads to practical means to capture WIS requirements. The media schemata that we introduced at the beginning then become means for reification in the sense that the details of each scene in the story space will be modelled by some media type. However, we have also seen that the introduction of story spaces opened a wide field for propositional reasoning about WISs. To that end many quality criteria can be verified already at the level of storyboarding.

Turning back to media types another challenge arises, i.e. to achieve a more detailed level of reasoning by combining the ideas for personalisation, obligation and rights with the media types. Of course, in doing so we leave the grounds of propositional reasoning. Instead of KATs we have to apply higher-order dynamic logic; instead of propositional deontic logic we have to deal with higher-order deontic logic.

The major addition comes from adding assignments to the operations. Thus, we may specify the body of operations on media types using *abstract programs* that are defined by the following constructs:

- 1 and 0 are abstract programs meaning `skip` and `fail`, respectively.
- An *assignment* $x := exp$ with a variable x and an expression of the same type as x is an abstract program. The possible expressions are defined by the type system. In addition, we permit expressions $\{\mathcal{P}\}$ with a logic program $\mathcal{P}$ that expresses a query, assuming that $\mathcal{P}$ contains a variable ans. The expression $\{\mathcal{P}\}$ is interpreted as the result of the logic program bound to ans.
- If p, p_1 and P_2 are abstract programs, the same holds for the *iteration* p^*, the choice $p_1 + p_2$ and the sequence $p_1 \cdot p_2 = p_1 p_2$.
- If p is an abstract program and φ is a condition, then the *guarded program* φp and the *postguarded program* $p\varphi$ are also abstract programs.
- If x is a variable and p is an abstract program, then the *selection* $@x \bullet p$ is also an abstract program.

There are a few subtleties regarding variable scoping and restrictions to assignments that have to be taken into account for operations on interaction types (see [ST05]), but for our purposes here it is not relevant to discuss them.

With respect to the connection to the story space, the propositional conditions φ now have to be refined to conditions on $\mathcal{S} \cup \mathcal{C}$, while each action α on a scene s is refined by an operations associated with the media type that supports s.

With the introduction of media types to support scenes we can no longe rely on simple equational reasoning using KATs. Therefore we introduce a higher-order dynamic, where the order comes from the intrinsic use of the set constructor and the logic programs in

queries. In fact, instead of using logic programs with a semantics defined by inflationary fixed-points, we could use directly higher-order logic enriched with a fixed-point operator.

As a consequence, we may consider a logic program $\mathcal{P}$ as a representative of a higher-order logical formula, say $\varphi_{\mathcal{P}}$. If $\{\mathcal{P}\}$ is used as the right-hand side of an assignment, then it will correspond to a term $\mathbf{I}ans.\varphi_{\mathcal{P}}$ denoting the unique ans satisfying formula $\varphi_{\mathcal{P}}$. That is, all conditions turn out to be formulae of a logic $\mathcal{L}$, which happens to be a higher-order logic with an inflationary fixed-point operator. From the point of view of expressiveness the fixed-point operator is already subsumed by the order, but for convenience we do not emphasise this aspect here.

Furthermore, by adding terms of the form $\mathbf{I}x.\varphi$ with a formula φ and a variable x all assignments in operations are just "normal" assignments, where the left-hand side is a variable and the right-hand side is a term of $\mathcal{L}$.

We now extend $\mathcal{L}$ to a dynamic logic by adding formulae of the form $[p]\varphi$ with an abstract program p and a formula φ of $\mathcal{L}$. Informally, $[p]\varphi$ means that after the successful execution of p the formula φ necessarily holds [HKT00]. In addition, we use the shortcut $\langle p\rangle\varphi \equiv \neg[p]\neg\varphi$, so $\langle p\rangle\varphi$ means that after the successful execution of p it is possible that the formula φ holds. Rules for $[p]\psi$ for a complex abstract program p and complex conditions ψ are given in [HKT00].

With these preparations we can rethink the reasoning about the story space. In the sequel we will discuss three applications of dynamic logic. We take a look at proof obligations for the operations that result from the the specification of the story space. We take a look at proof obligations for the operations that arise from static and dynamic integrity constraints on the underlying database schema. We reconsider WIS personalisation in the light of dynamic logic as opposed to KATs.

First let p denote the KAT expression that represents the complete story space. If all conditions in p are replaced by conditions on the pre-site and all actions are replaced by the abstract programs defining the realising operations, we obtain an abstract program, which by abuse of notation shall still be denoted p.

As a WIS has a general purpose, this can be formalised by some post-condition ψ. Thus, $[p]\psi$ describes the weakest condition, under which the purpose of the system can be achieved. If φ characterises a precondition that should be sufficient for the achievability of the WIS purpose, then we obtain $\varphi \rightarrow [p]\psi$ as a *general story space proof obligation*. In most cases we should expect $\varphi \equiv 1$.

Similarly, we may concentrate on fragments p' of the story space expression of the form $\varphi p\psi$, which corresponds to a Hoare triplet $\{\varphi\}p\{\psi\}$ and thus gives rise to a *special story space proof obligation* $\varphi \rightarrow [p']\psi$.

A *static constraint* on the underlying database schema $\mathcal{S}$ is a condition ζ, in which the free variables are among the $R \in \mathcal{S}$. Such constraints give rise to the request that whenever an operation is started in a database satisfying ζ, then the database reached after successfully completing the execution of the operation, must necessarily satisfy ζ, too.

That is, for all operations p that are defined on a pre-site and all static constraints ζ we obtain a *static consistency proof obligation* $\zeta \rightarrow [p]\zeta$.

A *dynamic constraint* on the underlying database schema $\mathcal{S} = \{R_1, \ldots, R_n\}$ is a condition ζ, in which the free variables are among in $\mathcal{S} \cup \mathcal{S}'$ with $\mathcal{S}' = \{R'_1, \ldots, R'_n\}$ and each R'_i having the same type as R_i. The additional variables R'_i are used to distinguish between $\mathcal{S}$-databases db, on which an operation p is started, and $\mathcal{S}'$-databases db' resulting after p has been successfully executed.

Obviously, a dynamic constraint ξ has to be interpreted on a pair (db, db') of databases. Following a standard approach to dynamic consistency we associate with ξ an abstract program

$$p(\xi) = @R'_1, \ldots, R'_n \bullet \xi \; R_1 := R'_1 \ldots R_n := R'_n.$$

Then dynamic consistency of an operation p with respect to ξ means that p must "specialise" $p(\xi)$, i.e. we require that $[p(\xi)]\psi \rightarrow [p]\psi$ for all conditions ψ on $\mathcal{S}$. Fortunately, this proof obligation can be rephrased using a renaming p' of $p(\xi)$ given by

$$@R'_1, \ldots, R'_n \bullet \xi\{R_1/R''_1, \ldots, R_n/R''_n\} \; R''_1 := R'_1 \ldots R''_n := R'_n.$$

If this is denoted p', the *dynamic consistency proof obligation* for p with respect to ξ becomes

$$([p']\langle p\rangle(R_1 = R''_1 \wedge \cdots \wedge R_n/R''_n))\{R''_1/R_1, \ldots, R''_n/R_n\}.$$

Finally, the general approach to personalisation that was outlined in Section 5 is still the same, i.e. we can assume a set Σ containg general constraints on $\mathcal{S} \cup \mathcal{C}$ and specific constraints that refer to preferences of a particular user type. Furthermore, personalisation assumes a postcondition χ that expresses the goals of the particular user. Then personalisation of story space p aims at a simpler story space p' such that $[p]\chi \leftrightarrow [p']\chi$ holds.

8 Conclusion

In this article we presented a brief overview of the co-design approach to WIS development focussing on the challenges that arise from large systems with unknown users in diverse environments, and to show how the approach copes with these challenges. In particular, we outlined that it is not sufficient to take only a data-driven approach, i.e. to model some underlying database schema, from which views are derived that capture the content of the WIS. We argued that this is only the first step, and that generation of navigation, hierarchies that enable different granularity for the presentation, and adaptivity to users and environment have to be handled as well.

We also argued that in order to capture the mission of a WIS and to match the anticipated usage by its users, an approach on a higher level of abstraction is needed. This is dealt with by storyboarding. One of the advantages of the co-design approach is the tight coupling of storyboarding with the data-oriented modelling of a WIS using media types. This also enables sophisticated reasoning about personalisation, task consistency, and obligations and rights of users.

A remaining challenge for future work is to extend the reasoning approach and to combine it with the data-oriented level of abstraction. First steps in this direction have been made in [Sc04].

References

[ABS00] Abiteboul, S., Buneman, P., und Suciu, D.: *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers. 2000.

[AGS98] Atzeni, P., Gupta, A., und Sarawagi, S.: Design and maintenance of data-intensive websites. In: *Proceeding EDBT'98*. volume 1377 of *LNCS*. pp. 436–450. Springer-Verlag. Berlin. 1998.

[AK89] Abiteboul, S. und Kanellakis, P.: Object identity as a query language primitive. In: *Proceedings SIGMOD 1989*. 1989.

[CFB+03] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., und Matera, M.: *Designing Data-Intensive Web Applications*. Morgan Kaufmann. San Francisco. 2003.

[DL98] De Troyer, O. und Leune, C.: WSDM: A user-centered design method for web sites. In: *Computer Networks and ISDN Systems – Proceedings of the 7th International WWW Conference*. pp. 85–94. Elsevier. 1998.

[El97] Elgesem, D.: The modal logic of agency. *Nordic Journal of Philosophical Logic*. 2:1–48. 1997.

[HBFV03] Houben, G.-J., Barna, P., Frasincar, F., und Vdovjak, R.: HERA: Development of semantic web information systems. In: *Third International Conference on Web Engineering – ICWE 2003*. volume 2722 of *LNCS*. pp. 529–538. Springer-Verlag. 2003.

[HKT00] Harel, D., Kozen, D., und Tiuryn, J.: *Dynamic Logic*. The MIT Press. Cambridge (MA), USA. 2000.

[Ko97] Kozen, D.: Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*. 19(3):427–443. 1997.

[Sc04] Schewe, K.-D.: The power of media types. In: *Proceedings WISE 2004: Web Information Systems Engineering*. LNCS. Springer-Verlag. 2004.

[SR98] Schwabe, D. und Rossi, G.: An object oriented approach to web-based application design. *TAPOS*. 4(4):207–225. 1998.

[SS00] Schewe, K.-D. und Schewe, B.: Integrating database and dialogue design. *Knowledge and Information Systems*. 2(1):1–32. 2000.

[ST05] Schewe, K.-D. und Thalheim, B.: Conceptual modelling of web information systems. *Data and Knowledge Engineering*. 2005. to appear.

[TACS98] Teodorakis, M., Analyti, A., Constantopoulos, P., und Spyratos, N.: Context in information bases. In: *Proceedings CoopIS '98*. pp. 260–270. 1998.