Integration and Cooperation of Media Types

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

[1]Massey University, Information Science Research Centre
& Department of Information Systems
Private Bag 11222, Palmerston North, New Zealand
[h.ma|k.d.schewe]@massey.ac.nz
[2]Christian Albrechts University Kiel
Department of Computer Science and Applied Mathematics
Olshausenstr. 40, D-24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de

Abstract: Media types are a core design construct in the co-design approach for web information systems (WISs). They provide abstract means for describing content, functionality, context and adaptivity to user preferences and intentions, end-devices, and channel limitations. Basically, a media type is a view that is extended by operations and cohesion. Thus, the problem of integrating these extended views is a core design problem for WISs.

In this paper we develop formal transformation rules for media type integration that are embedded in a pragmatic method addressing how they should be applied for integration. They extend view integration rules in such a way that operations and cohesion are dealt with simultaneously.

Cooperation provides an alternative to integration in which the integrated view is only virtual, i.e. the constituating views are kept and exchange functions are designed to provide the same functionality as if the views were integrated. We show that the transformation rules can also be applied to the problem of media type cooperation.

1 Introduction

In the field of web information systems (WISs) the importance of views is commonly accepted [AGS98, CFB+03, ST04, ST05]. In [FST98, FKST00] the notion of *media type* has been introduced. A media type is an extended view over some underlying database. However, different to the approaches in [AGS98, CFB+03] the defining query already constructs the navigation structure. Furthermore, media types are prepared for dynamic WISs [LG99] by adding operations, and for adaptivity by adding cohesion preorders.

As it is likely that during the development of WISs the local data, i.e. the views, that is to be presented to the user and the local functionality is modelled before the underlying database schema is set up, the integration of media types is a decisive part of a WIS development methodology. At its core media type integration concerns the old problem of database schema and integration.

The starting point for schema integration is a set of schemata over some data models. Usually the focus is on two schemata over the same data model. If the underlying data models differ, then we may assume some preprocessing transforming both schemata – or one of them, if this is sufficient – into an almost equivalent schema over a data model with higher expressiveness. Then schema integration aims at replacing the given schemata by a single new one in such a way that the new schema dominates or is equivalent to the old ones.

If the involved schemata are the target schemata of views we talk of *view integration* [BC86, SP94, Th00]. In this case the schema transformation function defines an embedding into the new schema. If this is coupled with the defining queries for the given views we obtain a new defining query for the integrated view.

Pragmatically, we may therefore follow the framework in [LS00]. In a nutshell, we first "clean" the given schemata by removing name conflicts, then add inter-schema constraints, which leads to a single constrained schema, i.e. just the union of the given ones. To this schema we then apply formal transformation rules, which will finally take us to an integrated schema that is either equivalent to the union of the given ones or dominates this. As we deal with extended views, the transformation rules for media type integration have to extend the view integration rules in a way that effects on operations and cohesion are covered as well. In this paper we will develop these extensions.

The transformation rules can be used as well for view cooperation [Th00], which provides an alternative to view integration in which the integrated view is only virtual. That is the constituating views are kept and exchange functions are designed to provide the same functionality as if the views were integrated. In addition, media type cooperation may contribute to realising cooperative tasks in WISs.

We start in Section 2 with a look at published work on the topic of this paper. In Section 3 we briefly present media types and discuss the integration and cooperation problem in general. In Section 4 we present our pragmatic method for integration and cooperation as well as the transformation rules that are needed for the method. We conclude with a short summary.

2 Related Work

The work on view integration in [KC94, LNE89, SP94] is based on the Entity-Relationship model. Larson et al. [LNE89] consider containment, equivalence and overlap relations between attributes and types that are defined by looking at "real world objects". Equivalence between types gives rise to their integration, containment defines a hierarchy with one supertype and one subtype, and overlapping gives rise to new common supertype. The work by Spaccapietra and Parent [SP94] considers also relationships, paths and disjointness relations betwen types. The work by Koh et al. [KC94] provides additional restructuring rules for the addition or removal of attributes, generalization and specialization, and the introduction of surrogate attributes for types.

The work by Biskup and Convent in [BC86] is based on the relational data model with

functional, inclusion and exclusion dependencies. The method is based on the definition of integration conditions, which can be equality, containment, disjointness or selection conditions. Transformations are applied aiming at the elimination of disturbing integration conditions. In the same way as our work it is based on a solid theory. On the other hand, it has never been applied to large systems in practice. The approach by Sciore et al. in [SSR94] investigates conversion functions on the basis of contexts added to values. These contexts provide properties to enable the semantic comparability of values.

The work by Lehmann and Schewe [LS00] assumes that the given schemata are defined on the basis of the higher-order Entity-Relationship model (HERM) [Th00] which is known to provide enough expressiveness such that schemata existing in practice can be easily represented in HERM. The work relies on the notions of equivalence and dominance as defined for HERM in [Th00].

The design and development of WISs has attracted a lot of attention. The ARANEUS framework in [AGS98] 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. OOHDM [SR98] is quite similar to ARANEUS, but its origins are not in the area of databases but in hypertext and it explicitly refers to an object oriented approach. OOHDM emphasises an object layer, hypermedia components, i.e. links, and an interface layer. The work in [GPS93] 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 is WebML [CFB+03], which emphasises a multi-level architecture for the data-driven generation of WIS, thus takes the view aspect into account. Furthermore, it emphasises structures, derivation and composition, i.e. views, navigation and presentation, thus addresses the same problem triplet as the ARANEUS framework.

Our own work on WIS design combines a usage-oriented storyboarding methodology [TD01] and the content- and functionality-oriented theory of media types [FKST00, ST04], which are embedded in an integrated co-design methodology based on an abstraction layer model [ST05].

3 Web Information Systems

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).

The methodology for WIS design in [ST01, ST05] emphasises 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 underlying application stories. Storyboarding first describes a *story space* by scenes and actions on these scenes. Furthermore, it describes the *actors* in these scenes, i.e. groups of users of the WIS. Actor modelling leads to roles, profiles, goals, preferences, obligations and rights. Finally, the actors are linked to the story space by the means of *tasks*.

Further on in the development process the scenes in the story space have to be adequately supported. For this the methodology focusses on *media types*, which cover extended views, adaptivity and hierarchies. In a nutshell, a media type is an extended view on some underlying database schema. These views are built in a way that they capture the complex content and navigation structure that is to be presented to a user. Adaptivity to users, channels and end-devices mainly concerns the question, whether all information or only the most important part of it is to be presented to a user. By specifying on a conceptual level what these "most important" parts are and which parts have to be kept together we may then leave the technical realisation of adaptivity to an algorithmic solution. Hierachies enable the presentation of information at different levels of granularity allowing a user to switch between these levels. However, we will not deal with hierarchies in this paper.

It is likely that the content that is to be made available to users is modelled, before an underlying database schema and thus defining queries for media types have been defined. This leads uanavoidably to a view integration problem. In addition, we have to cope with the implications for operations and for adaptivity.

3.1 Media Types

At the core of a media type [FKST00, ST05] we have a view that is extended by operations, which is an idea similar to dialogue types [SS00]. First recall that a view is nothing but a stored query. For this we assume familiarity with HERM [Th00].

Definition 1 A *view* V on a HERM schema $(\mathcal{S}, \Sigma)$ consists of a schema $\mathcal{S}_V$ and a query q_V with a query mapping $inst(\mathcal{S}, \Sigma) \rightarrow inst(\mathcal{S}_V)$.

The addition of operations leads first to the notion of interaction type as defined (in a simplified form) next.

Definition 2 An *interaction type* I over a HERM schema $(\mathcal{S}, \Sigma)$ consists of a view $V_I = (\mathcal{S}_I, q_I)$, and a set $\mathcal{O}$ of *dialogue operations*.

Each dialogue operation (*d-operation* for short) in $\mathcal{O}$ consists of

- an operation name op,
- a list of input parameters $i_1 : D_1, \ldots, i_k : D_k$ with domain names D_i,
- an (optional) output domain D_{out},

- a subattribute sel of X_E, and
- a d-operation body, which is built from usual programming constructs operating on instances over $(\mathcal{S}, \Sigma)$ and constructs for creating and deleting dialogue objects.

In this definition we permit references (or roles) between the *interaction objects* of any type in $\mathcal{S}_I$ that result from applying the defining query q_I to an instance over $(\mathcal{S}, \Sigma)$. This usually requires q_I to be written in a highly expressive query language.

Furthermore, for each interaction object (i, v) in $q_I(db)$ we interpret the abstract identifier i as a surrogate for a URL address. In this way the queries used in interaction types already define the navigation structure of the WIS. This is a fundamental difference to work by others as e.g. in [CFB+03], where views are only used to extract data, whereas the navigation structure is added later on in a separate design step.

Apart from this media types extend interaction types by *cohesion* in order to enable adaptivity. *Cohesion* introduces a controlled form of information loss exploiting the partial order $\geq$ on nested attributes.

Definition 3 If X_M is the representing attribute of an interaction type M and $sub(X_M)$ is the set of all nested attributes Y with $X_M \geq Y$, then a preorder $\preceq_M$ on $sub(X_M)$ extending the order $\geq$ is called a *cohesion preorder*.

Large elements in $sub(X_M)$ with respect to $\preceq_M$ define information to be kept together, if possible. Clearly, X_M is maximal with respect to $\preceq_M$. This enables a controlled form of information decomposition [ST05]. So we obtain the following (simplified) definition of a media type.

Definition 4 A *media type* is an interaction type M together with an cohesion preorder $\preceq_M$.

See [ST05] for an algorithmic approach to adaptivity based on cohesion preorders. This algorithm is not relevant for our purposes here. In that article also alternatives to cohesion preorders, which consist of proximity values, but lead to the same results, are discussed.

3.2 Media Type Integration

As the core of a media type is defined by a view, the core problem of media type integration is that of view integration. So we start with two views V_1 and V_2 on a HERM schema $(\mathcal{S}, \Sigma)$. The result should be a new integrated view V such that $\mathcal{S}_V$ results from integration of the schemata $\mathcal{S}_{V_1}$ and $\mathcal{S}_{V_2}$, and for each instance db over $(\mathcal{S}, \Sigma)$ the two query results $q_{V_1}(db)$ and $q_{V_2}(db)$ together are equivalent to $q_V(db)$.

In particular, view integration requires precise notions for schema dominance and equivalence, which we will introduce now.

Definition 5 A HERM schema $(\mathcal{S}', \Sigma')$ *dominates* another HERM schema $(\mathcal{S}, \Sigma)$ by means of the language $\mathcal{L}$ (notation: $(\mathcal{S}, \Sigma) \sqsubseteq_{\mathcal{L}} (\mathcal{S}', \Sigma')$) iff there are mappings $f : inst(\mathcal{S}, \Sigma) \rightarrow inst(\mathcal{S}', \Sigma')$ and $g : inst(\mathcal{S}', \Sigma') \rightarrow inst(\mathcal{S}, \Sigma)$ both expressed in $\mathcal{L}$ such that the composition $g \circ f$ is the identity.

If we have $(\mathcal{S}, \Sigma) \sqsubseteq_{\mathcal{L}} (\mathcal{S}', \Sigma')$ as well as $(\mathcal{S}', \Sigma') \sqsubseteq_{\mathcal{L}} (\mathcal{S}, \Sigma)$, we say that the two schemata are *equivalent* with respect to $\mathcal{L}$ (notation: $(\mathcal{S}, \Sigma) \cong_{\mathcal{L}} (\mathcal{S}', \Sigma')$).

We may obtain different notions of dominance and equivalence. $\sqsubseteq_{\mathcal{H}}$ and $\cong_{\mathcal{H}}$ refer to the use of the HERM algebra or equivalently the HERM calculus [Th00] as the language, in which the transformations f and g are to be expressed. Analogously, $\sqsubseteq_{\mathcal{H}_{ext}}$ and $\cong_{\mathcal{H}_{ext}}$ refer to the use of the extended HERM algebra or the extended HERM calculus [Th00]. Finally, $\sqsubseteq_{comp}$ and $\cong_{comp}$ refer to the use of computable queries [CH80]. In the following we will always refer to $\sqsubseteq_{comp}$ and $\cong_{comp}$ and therefore drop the index and simply write $\sqsubseteq$ for dominance and $\cong$ for equivalence.

Now, if the schemata $\mathcal{S}_{V_1}$ and $\mathcal{S}_{V_2}$ are "cleaned", we may combine the queries q_{V_1} and q_{V_2} into one yielding a query mapping $inst(\mathcal{S}, \Sigma) \rightarrow inst(\mathcal{S}_{V_1} \cup \mathcal{S}_{V_2})$ defined by the query $q_{V_1} \cup q_{V_2}$. If we simply integrate the schemata $\mathcal{S}_{V_1}$ and $\mathcal{S}_{V_2}$ into $\mathcal{S}_V$ according to the method described above, we obtain an induced mapping $f : inst(\mathcal{S}_{V_1} \cup \mathcal{S}_{V_2}) \rightarrow inst(\mathcal{S}_V)$. As we deal with computable queries, f is the query mapping of some computable query q_f. Taking $q_V = q_f \circ (q_{V_1} \cup q_{V_2})$, V becomes a view over $(\mathcal{S}, \Sigma)$ with schema $\mathcal{S}_V$ and defining query q_V.

Finally, we have to adapt d-operations and the cohesion preorder.

3.3 Media Type Cooperation

View cooperation [Th00] provides an alternative to view integration in which the integrated view is only virtual. That is the constituating views are kept and exchange functions are designed to provide the same functionality as if the views were integrated.

Definition 6 Let $V_i = (\mathcal{S}_{V_i}, q_{V_i})$ $(i = 1, 2)$ be views (on the same or different HERM schemata). V_1 *cooperates* with V_2 iff there are subschemata $\mathcal{S}'_{V_i}$ of $\mathcal{S}_{V_i}$ and functions $f_1 : inst(\mathcal{S}'_{V_1}) \rightarrow inst(\mathcal{S}'_{V_2})$ and $f_2 : inst(\mathcal{S}'_{V_2}) \rightarrow inst(\mathcal{S}'_{V_1})$, such that both $f_1 \circ f_2$ and $f_2 \circ f_1$ are the identity function.

Basically, view cooperation expresses that part of view V_1, exactly the one corresponding to the subschema $\mathcal{S}'_{V_1}$ can be expressed by the part of view V_2 corresponding to the subschema $\mathcal{S}'_{V_2}$.

Now, if we want to obtain a cooperation between given views V_1 and V_2, we may simply apply view integration to them using the same transformation rules. This will result in an integrated view $V = (\mathcal{S}_V, q_V)$. With respect to this integrated view both $\mathcal{S}'_{V_1}$ and $\mathcal{S}'_{V_2}$ will be identified with a subschema $\mathcal{S}'_V$. In particular we obtain functions $f'_i : inst(\mathcal{S}'_{V_i}) \rightarrow inst(\mathcal{S}'_V)$ and $g'_i : inst(\mathcal{S}'_V) \rightarrow inst(\mathcal{S}'_{V_i}))$ $(i = 1, 2)$ with $g'_i \circ f'_i = id$ and $f'_i \circ g'_i = id$.

Thus, $f_1 = g_2' \circ f_1'$ and $f_2 = g_1' \circ f_2'$ define the view cooperation functions.

Consequently, if the view integration method takes care of operations and cohesion, we also obtain cooperating media types.

4 A Pragmatic Method for Integration and Cooperation

In this section we address media type integration. The underlying process for HERM schema integration follows [LS00]. Without loss of generality we assume that we are given only two media types. Before starting the integration process we assume that these types have been "cleaned", i.e. we assume that all name clashes have been removed by renaming homonyms.

4.1 Integration Process

We only have to describe a process for schema integration, as we already explained how this can be adapted for view integration and cooperation. The details are then filled by transformation rules described in the following subsection. In particular, our rules will deal with extensions to operations and cohesion in order to become applicable to media types.

1. The first step is the homogenisation of the schemata. This includes the restructuring of the schemata turning attributes into entity types, entity types into relationship types and vice versa. Furthermore, we add attributes and shift attributes along hierarchies and paths. All these individual paces correspond to the application of transformation rules. The result of the homogenisation step are schemata $(\mathcal{S}_1', \Sigma_1')$ and $(\mathcal{S}_2', \Sigma_2')$.

2. The second step consists in adding inter-schema integrity constraints that describe the semantic relationships between the two schemata. Formally, we obtain another set of constraints Σ_0, and thus the result of this step is a single HERM schema $(\mathcal{S}_1' \cup \mathcal{S}_2', \Sigma_1' \cup \Sigma_2' \cup \Sigma_0)$.

3. Step three considers the integration of types on level 0, 1, etc., i.e. we start with entity types and level-0-clusters, then proceed with relationship types and clusters on level 1, then relationship types and clusters on level 2, etc. For each level we integrate corresponding types or clusters with respect to equality, containment, overlap and disjointness conditions. Note that this step is similar to the work done in [KC94, LNE89, SP94].

4. The fourth step deals with the integration of paths using path inclusion dependencies.

5. Finally, we consider remaining integrity constraints such as (path) functional dependencies and join dependencies.

4.2 Transformation Rules

We now describe the transformation rules in detail. All rules will presented in the same way, i.e. we assume a given HERM schema $(\mathcal{S}, \Sigma)$, but we only indicate parts of it. The resulting schema will be $(\mathcal{S}_{new}, \Sigma_{new})$. The new types in the new schema will be marked with a subscript $_{new}$. With these conventions the rules will be self-explaining.

Furthermore, the presence of d-operations and cohesion preorders requires extensions dealing with the impact of the view integration on the operations and the cohesion preorder. We will discuss these rules together with the extensions.

4.2.1 Schema Restructuring.

The first group of rules addresses the aspect of schema restructuring which will be used in the homogenisation step 1 of our method.

Rule 1 Replace a tuple attribute $X(A_1, \ldots, A_m)$ in an entity or relationship type R by the attributes $A_1, \ldots, A_m$. The resulting type R_{new} will replace R. For $X(A'_1, \ldots, A'_n) \in$ *key*(R) with $A_i \leq A'_i$ we obtain $A'_1, \ldots, A'_n \in$ *key*(R').

In this case, whenever $X(A'_1, \ldots, A'_k)$ $(k \leq m)$ appears in sel, in the body of a d-operation or in $Y \in sub(X_M)$, replace it by $A'_1, \ldots, A'_k$.

This rule includes the simple case, where R is an entity type, which could be treated as a separate rule.

Rule 2 Replace a component $r : R'$ in a relationship type R by lower level components and attributes. Let the new type be R_{new}. For comp$(R') = \{r_1 : R_1, \ldots, r_n : R_n\}$ we get comp(R_{new}) = comp$(R) - \{r : R'\} \cup \{r_1^{(r)} : R_1, \ldots, r_n^{(r)} : R_n\}$ with new role names $r_i^{(r)}$ composed from r_i and r and attr(R_{new}) = attr$(R) \cup$ attr(R'). In the case $r : R' \in$ key(R) and key$(R') = \{r_{i_1} : R_{i_1}, \ldots, r_{i_k} : R_{i_k}, A_1, \ldots, A_m\}$ we obtain key(R_{new}) = key$(R) - \{r : R'\} \cup \{r_{i_1}^{(r)} : R_{i_1}, \ldots, r_{i_k}^{(r)} : R_{i_k}, A_1, \ldots, A_m\}$, otherwise we have key$(R_{new})$ = key(R).

In this case, whenever r appears in sel, in the body of a d-operation or in $Y \in sub(X_M)$, replace it by $r_1^{(r)}, \ldots, r_n^{(r)}$.

It is easy to see how to simplify this rule in the case, where R' is an entity type. Again, this could by formulated by two separate rules.

In the case of the restructuring rules 1 and 2 we can always show that the original schema and the resulting schema are equivalent. The next rule only guarantees that the resulting new schema dominates the old one.

Rule 3 Replace a key-based inclusion dependency
$R'[key(R_i)] \subseteq R[key(R)]$ by new relationship types R'_{new} with $comp(R'_{new}) = \{r' : R', r : R\} = key(R'_{new})$ and $attr(R'_{new}) = \emptyset$ together with participation cardinality constraints
$card(R'_{new}, R) = (0, 1)$ and $card(R'_{new}, R') = (1, 1)$.

There is nothing to add for rule 3, as this introduces a new type, so operations and cohesion have to be defined for that new type. The last two restructuring rules allow to switch between attributes and entity types and between entity and relationship types. These rules 4 and 5 guarantee schema equivalence.

Rule 4 Replace an entity type E with $A \in attr(E)$ by E_{new} such that $attr(E_{new}) = attr(E) - \{A\}$ holds. Furthermore, introduce an entity type E'_{new} with $attr(E'_{new}) = \{A\} = key(E'_{new})$ and a new relationship type R_{new} with $comp(R_{new}) = \{r_{new} : E_{new}, r'_{new} : E'_{new}\} = key(R_{new})$ and $attr(R_{new}) = \emptyset$.

Add the cardinality constraints $card(R_{new}, E_{new}) = (1, 1)$ and $card(R_{new}, E'_{new}) = (1, \infty)$.

In this case, omit A in sel, in the body of a d-operation and in $Y \in sub(X_M)$, whenever it appears.

Rule 5 Replace an relationship type R with $comp(R) = \{r_1 : R_1, \ldots, r_n : R_n\}$ and the cardinality constraints
$card(R, R_i) = (x_i, y_i)$ by a new entity type E_{new} with
$attr(E_{new}) = attr(R) = key(E_{new})$ and n new relationship types $R_{i,new}$ with
$comp(R_{i,new}) = \{r_i : R_i, r : E_{new}\} = key(R_{i,new})$ and $attr(R_{i,new}) = \emptyset$. Replace the cardinality constraints by
$card(R_{i,new}, R_i) = (1, y_i)$ and $card(R_{i,new}, E_{new}) = (1, \infty)$.

In this case, omit $r_1, \ldots, r_n$ in sel, in the body of a d-operation and in $Y \in sub(X_M)$, whenever it appears.

In the case of rule 5 explicit knowledge of the key of R allows to sharpen the cardinality constraints.

4.2.2 Shifting Attributes.

The second group of rules deals with the shifting of attributes. This will also be used in the homogenisation step 1 of our method. Rule 6 allows to shift a synonymous attribute occurring in two subtypes, i.e. whenever tuples agree on the key they also agree on that attribute, to be shifted to a supertype. This rule leads to a dominating schema. Conversely, rule 7 allows to shift an attribute from a supertype to subtypes, in which case schema equivalence can be verified.

Rule 6 For $comp(R_i) = \{r_i : R\}$ and $A_i \in attr(R_i) - key(R_i)$ $(i = 1, 2)$ together with the constraint

$$\forall t, t'.t[key(R_1)] = t'[key(R_2)] \Rightarrow t[A_1] = t'[A_2]$$

replace the types R, R_1 and R_2 such that $\mathit{attr}(R_{new}) = \mathit{attr}(R) \cup \{A_i\}$, $\mathit{comp}(R_{i,new}) = \{r_i : R_{new}\}$ and
$\mathit{attr}(R_{i,new}) = \mathit{attr}(R_i) - \{A_i\}$ hold.

In this case, omit A_i in sel, in the body of a d-operation and in $Y \in sub(X_M)$ associated with $R_{i,new}$, whenever it appears.

Rule 7 For $\mathit{comp}(R_i) = \{r_i : R\}$ $(i = 1, \ldots, n)$ and $A \in \mathit{attr}(R) - \mathit{key}(R)$ together with the constraint $\forall t \in R. \exists t' \in R_i. t'[r_i] = t$ replace the types such that
$\mathit{attr}(R_{new}) = \mathit{attr}(R) - \{A\}$, $\mathit{comp}(R_{i,new}) = \{r_i : R_{new}\}$ and $\mathit{attr}(R_{i,new}) = \mathit{attr}(R_i) \cup \{A\}$ hold.

In this case, omit A_i in sel, in the body of a d-operation and in $Y \in sub(X_M)$ associated with R_{new}, whenever it appears.

Note that the last two rules have no effect on R_{new} or $R_{i,new}$, as the extension of operations and the cohesion preorder has to be defined for these new types.

The next two rules 8 and 9 concern the reorganisation of paths and the shifting of attributes along paths. In both cases we obtain a dominating schema. Rule 8 could be split into two rules dealing separately with binary and unary relationship types R_n.

Rule 8 For a path $P \equiv R_1 - \cdots - R_n$ and a relationship type R with $r_n : R_n \in \mathit{comp}(R)$ together with path cardinality constraints

$$\mathit{card}(P, R_1) \leq (1, 1) \leq \mathit{card}(P, R_n)$$

replace R such that $\mathit{comp}(R_{new}) = \mathit{comp}(R) - \{r_n : R_n\} \cup \{r_{1,new} : R_1\}$ with a new role $r_{1,new}$ holds.

In this case, replace r_n by $r_{1,new}$ in sel, in the body of a d-operation and in $Y \in sub(X_M)$ associated with R_{new}.

Rule 9 For a path $P \equiv R_1 - \cdots - R_n$ with $A \in \mathit{attr}(R_n)$ and path cardinality constraints

$$\mathit{card}(P, R_1) \leq (1, 1) \leq \mathit{card}(P, R_n)$$

replace R_1, R_n such that $\mathit{attr}(R_{1,new}) = \mathit{attr}(R_1) \cup \{A\}$ and $\mathit{attr}(R_{n,new}) = \mathit{attr}(R_n) - \{A\}$ hold.

In this case, omit A in sel, in the body of a d-operation and in $Y \in sub(X_M)$ associated with $R_{n,new}$.

4.2.3 Schema Extension.

The third group of rules deal with the schema extensions. This either concerns new attributes, new subtypes or the simplification of hierarchies. These rules are needed in step 1 of our method. For this group of rules only rules 10 and 13 give rise to reasonable extension rules for media types, but they do not affect the operations.

Rule 10 Add a new attribute A to the type R, i.e.
$\mathit{attr}(R_{new}) = \mathit{attr}(R) \cup \{A\}$. In addition, the new attribute may be used to extend the key, i.e. we may have $\mathit{key}(R_{new}) = \mathit{key}(R) \cup \{A\}$.

In this case, add A to all $Y \in sub(X_M)$ associated with R_{new} and extend the now flawed cohesion preorder.

If the new attribute A introduced by rule 10 does not become a key attribute, we obtain a dominating schema.

The next two rules allow to introduce a new subtype via selection or projection on non-key-attributes. In both cases we have schema equivalence.

Rule 11 For a type R introduce a new relationship type R'_{new} with $\mathit{comp}(R'_{new}) = \{r : R\} = \mathit{key}(R'_{new})$ and add a constraint $R'_{new} = \sigma_{\varphi}(R)$ for some selection formula φ.

Rule 12 For a type R and attributes $A_1, \ldots, A_n \in \mathit{attr}(R)$ such that there are no $B_i \in \mathit{key}(R)$ with $A_i \leq B_i$ introduce a new relationship type R'_{new} with $\mathit{comp}(R'_{new}) = \{r : R\} = \mathit{key}(R'_{new})$ and $\mathit{attr}(R'_{new}) = \{A_1, \ldots, A_n\}$, and add a constraint $R'_{new} = \pi_{A_1,\ldots,A_n}(R)$.

The last rule 13 in this group allows to simplify hierarchies.

Rule 13 Replace types R, $R_1, \ldots, R_n$ with $\mathit{comp}(R_i) = \{r_i : R\} = \mathit{key}(R_i)$ and $\mathit{card}(R, R_i) = (0, 1)$ $(i = 1, \ldots, n)$ by a new type R_{new} with $\mathit{comp}(R_{new}) = \mathit{comp}(R)$, $\mathit{attr}(R_{new}) = \mathit{attr}(R) \cup \bigcup_{i=1}^{n} \mathit{attr}(R_i)$ and $\mathit{key}(R_{new}) = \mathit{key}(R)$.

In this case, define the cartesian product of the cohesion preorders and extend the resulting flawed cohesion preorder.

4.2.4 Type Integration.

The fourth group of rules deals with the integration of types in step 3 of our method. Rule 14 considers the equality case, rule 15 considers the containment case, and rule 16 covers the overlap case. Note that these transformation rules cover the core of the approaches in [KC94, SP94, LNE89]. For this group of rules, however, no reasonable extensions for media types can be defined, as we deal with new types.

Rule 14 If R_1 and R_2 are types with $\mathit{key}(R_1) = \mathit{key}(R_2)$ and we have the constraint $R_1[\mathit{key}(R_1) \cup X] = f(R_2[\mathit{key}(R_2) \cup Y])$ for some $X \subseteq \mathit{comp}(R_1) \cup \mathit{attr}(R_1)$, $Y \subseteq \mathit{comp}(R_2) \cup \mathit{attr}(R_2)$ and a bijective mapping f, then replace these types by R_{new} with $\mathit{comp}(R_{new}) = \mathit{comp}(R_1) \cup (\mathit{comp}(R_2) - Y - \mathit{key}(R_2))$, $\mathit{attr}(R_{new}) = \mathit{attr}(R_1) \cup (\mathit{attr}(R_2) - Y - \mathit{key}(R_2)) \cup \{D\}$ and $\mathit{key}(R_{new}) = \mathit{key}(R_1) \cup \{D\}$ and an optional new distinguishing attribute D.

Rule 15 If R_1 and R_2 are types with *key*(R_1) = *key*(R_2) and the constraint $R_2[$*key*$(R_2) \cup Y] \subseteq f(R_1[$*key*$(R_1) \cup X]$ holds for some $X \subseteq$ *comp*$(R_1) \cup$ *attr*(R_1), $Y \subseteq$ *comp*$(R_2) \cup$ *attr*(R_2) and a bijective mapping f, then replace R_1 by $R_{1,new}$ with *comp*$(R_{1,new})$ = *comp*(R_1), *attr*(R_{new}) =
attr$(R_1) \cup \{D\}$ and *key*(R_{new}) = *key*$(R_1) \cup \{D\}$ and an optional new distinguishing attribute D. Furthermore, replace R_2 by $R_{2,new}$ with *comp*$(R_{2,new})\{r_{new} : R_{1,new}\} \cup$ *comp*$(R_2) - Y -$ *key*(R_2), *attr*$(R_{2,new})$ = *attr*$(R_2) - Y -$ *key*(R_2) and *key*$(R_{2,new})$ = $\{r_{new} : R_{1,new}\}$.

Rule 16 Let R_1 and R_2 are types with *key*(R_1) = *key*(R_2) such that for $X \subseteq$ *comp*$(R_1) \cup$ *attr*(R_1), $Y \subseteq$ *comp*$(R_2) \cup$ *attr*(R_2) and a bijective mapping f the constraints

$$\begin{aligned} &R_2[\textit{key}(R_2) \cup Y] \subseteq f(R_1[\textit{key}(R_1) \cup X] \quad , \\ &R_2[\textit{key}(R_2) \cup Y] \supseteq f(R_1[\textit{key}(R_1) \cup X] \quad \text{and} \\ &R_2[\textit{key}(R_2) \cup Y] \cap f(R_1[\textit{key}(R_1) \cup X] = \emptyset \end{aligned}$$

are not satisfied. Then replace R_1 by $R_{1,new}$ with *comp*$(R_{1,new})\{r_{1,new} : R_{new}\} \cup$ *comp*$(R_1) - X -$ *key*(R_1), *attr*$(R_{1,new})$ = *attr*$(R_1) - X -$ *key*(R_1) and *key*$(R_{1,new})$ = $\{r_{1,new} : R_{new}\}$, replace R_2 by $R_{2,new}$ with *comp*$(R_{2,new})\{r_{new} : R_{1,new}\} \cup$ *comp*$(R_2) - Y -$ *key*(R_2), *attr*$(R_{2,new})$ = *attr*$(R_2) - Y -$ *key*(R_2) and *key*$(R_{2,new})$ = $\{r_{new} : R_{1,new}\}$ and introduce a new type R_{new} with *comp*(R_{new}) = *comp*(R_1) $\cup$*comp*(R_2), *attr*(R_{new}) = *attr*$(R_1) \cup$ *attr*$(R_2) \cup \{D\}$ and *key*(R_{new}) = *key*$(R_1) \cup \{D\}$ and an optional new distinguishing attribute D.

The rules 14-16 could each be split into several rules depending on f being the identity or not and the necessity to introduce D or not. In all cases we obtain dominance.

Rule 17 considers the case of a selection condition, in which case schema equivalence holds.

Rule 17 If R and R' are types with *comp*$(R') \cup$*attr*$(R') = Z \subseteq$ *comp*$(R) \cup$*attr*(R) such that the constraint $R' = \sigma_\varphi(\pi_Z(R))$ holds for some selection condition φ, then omit R'.

4.2.5 Handling Integrity Constraints.

The fifth group of rules to be applied in step 4 of our method concerns transformations originating from path inclusion constraints. Rule 18 allows us to change a relationship type. This rule leads to equivalent schemata. Rule 19 allows to introduce a relationship type and a join dependency. Finally, rule 20 handles a condition under which a relationship type may be omitted. Both rules 19 and 20 guarantee dominance.

Rule 18 If there are paths $P \equiv R_1 - R - R_2$ and $P' \equiv R_2 - R' - R_3$ with *comp*$(R) = \{r_1 : R_1, r_2 : R_2\}$ and *comp*$(R') = \{r_3 : R_3, r'_2 : R_2\}$ such that the constraint $P[R_2] \subseteq P'[R_2]$ holds, then replace R in such a way that *comp*$(R_{new}) = \{r_1 : R_1, r_{new} : R'\}$, *attr*$(R_{new})$ = *attr*(R) and *key*(R_{new}) = *key*$(R) - \{r_2 : R_2\} \cup \{r_{new} : R'\}$ hold.

In this case, replace r_2 by r_{new} in sel, in the body of a d-operation and in $Y \in sub(X_M)$, whenever it appears.

Rule 19 If there are paths $P \equiv R_1 - R - R_2$ and $P' \equiv R_2 - R' - R_3$ with *comp*$(R) = \{r_1 : R_1, r_2 : R_2\}$ and *comp*$(R') = \{r_3 : R_3, r_2' : R_2\}$ such that the constraint $P[R_2] = P'[R_2]$ holds, then replace R and R' by R_{new} such that *comp*$(R_{new}) = \{r_1 : R_1, r_{2,new} : R_2, r_3 : R_3\}$, *attr*$(R_{new}) =$ *attr*$(R) \cup$ *attr*(R') and *key*$(R_{new}) =$ (*key*$(R) - \{r_2 : R_2\}) \cup$ (*key*$(R') - \{r_2' : R_2\}) \cup \{r_{2,new} : R_2\}$ hold. Add the join dependency $R_{new}[r_1, r_{2,new}] \bowtie R_{new}[r_{2,new}, r_3] \subseteq R_{new}[r_1, r_{2,new}, r_3]$.

In this case, replace r_2 by $r_{2,new}$ in sel, in the body of a d-operation and in $Y \in sub(X_M)$, whenever it appears.

Rule 20 If there are paths $P \equiv R_1 - R_2 - \cdots - R_n$ and $P' \equiv R_1 - R - R_n$ with *comp*$(R) = \{r_1 : R_1, r_n : R_n\}$ such that the constraint $P[R_1, R_n] = P'[R_1, R_n]$ holds, then omit R.

The final group of transformation rules 21-24 permits to handle remaining constraints such as functional dependencies, path functional dependencies, and join dependencies. All these constraints are described in detail in [Th00]. The rules refer to step 5 of our method.

Rule 21 handles vertical decomposition in the presence of a functional dependency. Rule 22 allows to simplify a key in the presence of a path functional dependency. Rule 23 introduces a new entity type in the presence of a path functional dependency. Finally, rule 24 replaces a multi-ary relationship type by binary relationship types in the presence of a join dependency. The four rules lead to dominating schemata.

Rule 21 If a functional dependency $X \rightarrow A$ with a generalized subset X of *attr*(E) and an attribute $A \in$ *attr*$(E) - X$ holds on an entity type E, but $X \rightarrow$ *key*(E) does not hold, then remove A from *attr*(E) and add a new entity type E'_{new} with *attr*$(E'_{new}) = X \cup \{A\}$ and *key*$(E'_{new}) = X$.

In this case, remove A in sel, in the body of a d-operation and in $Y \in sub(X_M)$, whenever it appears.

The last three rules have no extensions for media types.

Rule 22 For a path $P \equiv R_1 - R - R_2$ with *comp*$(R) = \{r_1 : R_1, r_2 : R_2\}$ such that the path functional dependency $X \rightarrow$ *key*(R_2) holds for a generalized subset X of *attr*(R_1) replace *key*(R) by $\{r_1 : R_1\}$.

Rule 23 For a path $P \equiv R_1 - \cdots - R_n$ such that the path functional dependency $X \rightarrow A$ holds for a generalized subset X of *attr*(R_1) and $A \in$ *attr*(R_n) add a new entity type E_{new} with *attr*$(E_{new}) = X \cup \{A\}$ and *key*$(E_{new}) = X$.

Rule 24 If R is an n-ary relationship type with *comp*$(R) = \{r_1 : R_1, \ldots, r_n : R_n\}$ and *attr*$(R) = \emptyset$ such that the join dependency $R[r_1, r_2] \bowtie \cdots \bowtie R[r_1, r_n] \subseteq R[r_1, \ldots, r_n]$

holds, then replace R by n new relationship types $R_{1,new}, \ldots, R_{n,new}$ with $comp(R_{i,new}) = \{r_1 : R_1, r_i : R_i\} = key(R_{i,new})$ and $attr(R_{i,new}) = \emptyset$.

5 Conclusion

In this paper we applied schema and view integration and cooperation to web information systems. The key concept is that of a media type, which is a view extended by operations and cohesion in order to facilitate adaptivity.

We presented a method for media type integration following the framework in [LS00], i.e. we first "clean" the schemata of the given types by removing name conflicts, then we add inter-schema constraints, and to this schema we then apply formal transformation rules. The transformation and augmentation rules are correct in the sense that they will always result in a new view that is equivalent to the original one or dominates it. Furthermore, most of the transformation rules require additional attention for the operations and cohesion. In addition, media type cooperation may contribute to realising cooperative tasks in WISs.

References

[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.

[BC86] Biskup, J. und Convent, B.: A formal view integration method. In: *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*. pp. 398–407. Association for Computing Machinery. 1986.

[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.

[CH80] Chandra, A. und Harel, D.: Computable queries for relational data bases. *Journal of Computer and System Sciences*. 21. 1980.

[FKST00] Feyer, T., Kao, O., Schewe, K.-D., und Thalheim, B.: Design of data-intensive web-based information services. In: Li, Q., Ozsuyoglu, Z. M., Wagner, R., Kambayashi, Y., und Zhang, Y. (Eds.), *Proceedings of the 1st International Conference on Web Information Systems Engineering (WISE 2000)*. pp. 462–467. IEEE Computer Society. 2000.

[FST98] Feyer, T., Schewe, K.-D., und Thalheim, B.: Conceptual modelling and development of information services. In: Ling, T. und Ram, S. (Eds.), *Conceptual Modeling – ER'98*. volume 1507 of *LNCS*. pp. 7–20. Springer-Verlag. Berlin. 1998.

[GPS93] Garzotto, F., Paolini, P., und Schwabe, D.: HDM - a model-based approach to hypertext application design. *ACM ToIS*. 11(1):1–26. 1993.

[KC94] Koh, J. und Chen, A.: Integration of heterogeneous object schemas. In: Elmasri, R., Kouramajian, V., und Thalheim, B. (Eds.), *Entity-Relationship Approach - ER'93*. volume 823 of *LNCS*. pp. 297–314. Springer-Verlag. 1994.

[LG99] Ludäscher, B. und Gupta, A.: Modeling interactive web sources for information mediation. In: Chen, P. P.-S. (Ed.), *Advances in Conceptual Modeling*. volume 1727 of *LNCS*. pp. 225–238. Springer-Verlag. 1999.

[LNE89] Larson, J., Navathe, S. B., und Elmasri, R.: A theory of attribute equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering*. 15(4):449–463. 1989.

[LS00] Lehmann, T. und Schewe, K.-D.: A pragmatic method for the integration of higher-order Entity-Relationship schemata. In: Laender, A. H. F., Liddle, S. W., und Storey, V. C. (Eds.), *Conceptual Modeling - ER 2000*. volume 1920 of *LNCS*. pp. 37–51. Springer-Verlag. 2000.

[SP94] Spaccapietra, S. und Parent, C.: View integration – a step forward in solving structural conflicts. *IEEE Transactions on Knowledge and Data Engineering*. 6(2):258–274. 1994.

[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.

[SSR94] Sciore, E., Siegel, M., und Rosenthal, A.: Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM TODS*. 19(2):254–290. 1994.

[ST01] Schewe, K.-D. und Thalheim, B.: Modeling interaction and media objects. In: Bouzeghoub, M., Kedad, Z., und Métais, E. (Eds.), *Natural Language Processing and Information Systems: 5th International Conference on Applications of Natural Language to Information Systems, NLDB 2000*. volume 1959 of *LNCS*. pp. 313–324. Springer-Verlag. Berlin. 2001.

[ST04] Schewe, K.-D. und Thalheim, B.: Structural media types in the development of data-intensive web information systems. In: Taniar, D. und Rahayu, W. (Eds.), *Web Information Systems*. pp. 34–70. IDEA Group. 2004.

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

[TD01] Thalheim, B. und Düsterhöft, A.: SiteLang: Conceptual modeling of internet sites. In: et al., H. S. K. (Ed.), *Conceptual Modeling – ER 2001*. volume 2224 of *LNCS*. pp. 179–192. Springer-Verlag. Berlin. 2001.

[Th00] Thalheim, B.: *Entity-Relationship Modeling: Foundations of Database Technology*. Springer-Verlag. 2000.