Applying mSpace interfaces to the Semantic Web

Nicholas Gibbins
School of Electronics and Computer Science
University of Southampton
Southampton, United Kingdom
nmg@ecs.soton.ac.uk

Stephen Harris
School of Electronics and Computer Science
University of Southampton
Southampton, United Kingdom
swh@ecs.soton.ac.uk

monica schraefel
School of Electronics and Computer Science
University of Southampton
Southampton, United Kingdom
mc@ecs.soton.ac.uk

ABSTRACT

Ontologies can represent large, multidimensional spaces: classical music, research in computer science in the UK, health care for breast cancer are examples of rich domains. There have been no easy ways to represent meaningful slices through these multidimensional spaces to privilege the parts of the domain that are of interest to a given user. mSpace, an interaction model we describe here, is particularly suited to ontology-based interaction because it is designed to expose and support exploration of relations in a domain. In this paper we propose the formalism for this interaction model to support mapping this kind of user-determined interaction onto a high dimensional space represented by an ontology. The model provides semantic web designers with a means for rapidly prototyping and interrogating the data represented by an ontology. It also and provides a fast, effective UI alternative to keyword search and browsing for users to explore the domain space while maintaining domain context.

Categories and Subject Descriptors

I.2.1 [**Applications and Expert Systems**]; H.5.2 [**User interfaces**]: Prototyping

Keywords

Semantic Web, Human factors

1. INTRODUCTION

The most popular method for searching the Web at present is by keyword search; it has proven to be an effective way to *retrieve* information from the Web. The effectiveness of the technique improves with the precision of the keywords used. As such, keyword searches rely on one's domain expertise to retrieve appropriate information. When domain expertise is less certain, one can use category resources like Yahoo or the Internet Project to search by subject and subcategories of subject, and narrow one's search in this way. An advantage of category search is that it communicates a sense of the range of data available in a given domain. Numbers besides category labels also help communicate the scope of the instances within that part of the domain. Indeed, there have been efforts to apply categorization to keyword search results [26], but these alert users to the categories only after the fact - based on the results of the retrieved information. One of the limitations of popular Web-based representations of category searches is that they usually rely on fixed hierarchies of categories. That is, both the order and the number of subcategories in the tree is fixed. This means that a user cannot reorient the space to support their focus. They must approach the domain from the organizational bias of the designers.

The Semantic Web's use of ontologies to structure domains presents new opportunities to represent domains to users for their interaction with the domain space. This means that users can have greater options in how they engage with Web-based information, thus affording more diverse approaches than keyword or category search for building knowledge from the information they discover. The semantics of ontologies, for instance, privilege relationships; ontologies also afford multiple paths to view the same data instance from multiple perspectives. Exposing these relationships and paths provides knowledge seekers with additional information to inform their knowledge building tasks. Within the Semantic Web community, however, there has been little research to date on how to exploit ontologies to formalize and automate such interaction design opportunities. The closest work in this space has been in information visualization. Understandably, in that case, most work has emphasized how extant visualization techniques can be applied to the Semantic Web [9]; interaction issues have been, again understandably, secondary. Our interest, however, is specifically how interaction affordances can be enhanced by leveraging the properties of the Semantic Web. To that end, we present a formal representation of an interaction model informed by the Semantic Web. Though we present one visualization to describe the model's interaction characteristics, the model itself is visualization agnostic. Designers can use whichever visualization they choose - tree maps [2], self-organizing maps [4], polyarchies [21], lists. Our interest here is to formalize the behaviours of the data given the set of interactions defined.

In this paper, therefore, we propose a formalism for mSpace [13], an interaction model which leverages the advantages of the Semantic Web in order to facilitate user-determined exploration of a domain. In brief, the model lets users arrange an n-dimensional information space such that they can determine both a slice through the space, and then the scope, orientation, and arrangement of the attributes in that slice. A slice is determined first by the selection of attributes (effectively, class expressions) within the ontology. This selection acts like a projection through an n-dimensional space, which is then flattened. The result is a hierarchical representation of the dependencies of attributes in that hierarchy, based on the ordering of the selections. The first attribute in the order represents a query for all the instances matching that attribute/expression. Selection of instances within that listing then act as a constraint in populating the instances of the next attribute in the hierarchy and so on. The model therefore supports two levels of user interaction: manipulation of the ontology representation itself and selection of the instances of the data associated with those configurations. The logic of the model also provides for automatic reasoning across the domain to ensure that only meaningful attribute orderings/selections can occur. Which of these affordances of the interaction model a system designer wishes to implement are up to the designer.

There are two key advantages to this interaction model for domain interaction. First, mSpace provides system designers with a way to support fast visual data inspection in the domain from numerous perspectives. It also gives designers an automatic way to leverage their ontologies to present the domain to a user such that users first, can readily perceive the scope and relations within the domain from the available attributes, and second can then explore the domain from an orientation of the information that suits their interests.

In the following section, we describe the mSpace interaction model in terms of the interaction design itself. We situate the related work within this discussion. Next we describe the formal model for the behaviours associated with each interaction. We follow this with a service-level characterization of the model. We conclude with our plans for future work, including a brief sketch of an API instantiating the mSpace model.

2. INTERACTION DESIGN OF THE MSPACE MODEL

The goal of an mSpace is to support users' *exploration* of a domain. By exploration, we mean something other than the Web sense of browsing or surfing. Surfing or browsing Web pages suggests moving among discrete Web pages which potentially have only very loose associations between them. The only association several pages may have in common, for instance, is that a user has visited each page within the past thirty seconds. By exploration, we mean that the user is making information selections within a structured information domain, based on ontological associations among its parts. The interaction design for exploration would provide mechanisms for the user to take advantage of these associations in exploring the domain. In exploration, context and the availability of contextual information to support users' exploration is a critical component of the interaction design. An mSpace, designed to support exploration, therefore privileges associations and contexts in the domain interaction and representation.

While mSpace was initially designed as an abstract interface model for exploring information domains, it seems particularly well suited to the Semantic Web, since its semantics themselves foreground the relations within a domain's ontology. Foregrounding these relations within an interaction design in turn provides users with an alternative method for interrogating an information source: while keyword searches focus on retrieving a set of matching instances from a information source, exploration focuses on representing the context of that information. Within the Human Computer Interaction community, this dichotomy between instance and relations is generally referred to as Focus + Context [20, 22]. The Web's emphasis on keyword searches has so far privileged the Focus part of the equation, if for no other reason than the unstructured nature of Web data is best searched against keyword indexing, which returns in a list of discrete instances/Web pages. The affordances of the Semantic Web raise the possibility of facilitating the Context side of this interaction equation. mSpace is designed to leverage such semantic affordances. The remainder of this section describes the specific interaction affordances of an mSpace.

2.1 mSpace Representation and Interaction

An ontology can represent a vast, multidimensional domain that can support many ways of focusing on its data. an mSpace facilitates both the representation of the space and the interaction with this representation in a variety of ways. These are:

slices: projections through the n-dimensional space,

context: spatial rather than temporal layout of a slice

dimensional sorting: the organization of attributes within a slice, and how the organization constrains the data associated with each attribute.

substitution: the revision of a slice by changing one attribute in a slice with another not previously in the slice

expansion/contraction: the addition of a new attribute to a slice; the removal of a previous attribute from a slice.

In the following sections we use the example of the Classical Music domain to ground our description of these features.

2.1.1 *Slices, Columns and Context*

As described above, slices represent projections through an n-dimensional space that are flattened. The flattened projection results in a hierarchical ordering of the attributes in slice. Slices mean that certain views of the domain are privileged over others. For example, the classical music domain may be sliced to privilege information about compositions in a variety of ways. One approach is to foreground information about compositions through the attributes Period (Romantic or Baroque) — Composer (Mozart or Bach) — Form (symphony or serenade) — Arrangement (solo instrument, orchestra) — Composition. Likewise the domain might be sliced to privilege information on compositions through recordings: Record Company — Orchestra — Conductor — Artist — Year Recorded — Composition. We refer to each of these attributes as Columns, in which the label of the column is the attribute name and the body of the column is the list of instances which are associated with the column - depending on its hierarchical location within the slice

Rationale: Flattened projections are not uncommon treatments for managing representations of n-dimensional spaces. Beyond making an n-dimensional space representationally manageable, flattened projections offer other benefits over visualizations attempting to approximate more than two dimensions. First, the resulting hierarchies can be readily represented in a variety of 2-dimensional views easily mapped onto the native affordance of a 2D computer screen. Three dimensions can at best only be simulated on such screens. Research has shown that 25% of the population cannot conceptualize information in 3-dimensions [17]. The same study showed that users performed significantly better in information retrieval tasks when navigating text-based hierarchies rather than either 2D map views of information clusters of hierarchies or 3D representations of these clusters. Other research has also challenged claims about the value of 2.5 or greater dimensional views for information organization and access [6]. Therefore 1-2D representations of slices through n-dimensional spaces provide both accessibility and performance benefits over more complex dimensional simulations.

While hierarchies can be represented in a variety of visualizations, our default visualization is a spatial, multipane or multicolumn view. An example of a spatial multicolumn view for the first Classical Music slice is shown in Figure 1.

A spatial view in general means that the associated context of the information is persistently maintained in the interface. The complement of spatial views are temporal views where context is

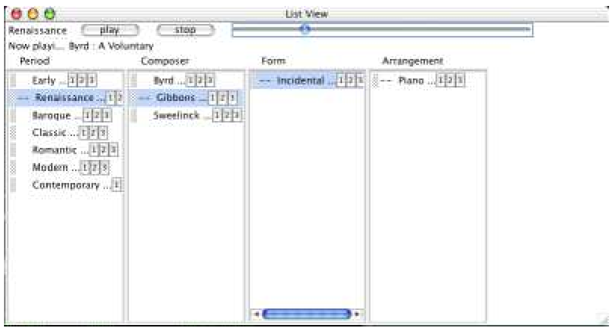


Figure 1: Spatial multicolumn view for classical music slice

maintained in memory. For instance, the standard Web representation of information is temporal: a click on a link generally replaces the originating page with a new page in the browser window, erasing the previously visible information. While mSpaces are visualization-neutral, we strongly recommend that mSpace visualizations support spatial context: in previous work we have shown that supporting spatial rather than temporal context results in better navigation performance by users [14]. Indeed, in the same study, we found that temporal interfaces had a significant negative correlation with age. With our simple default multipane view, we get a variety of benefits for low implementation cost: there is little overhead in rendering what are effectively contiguous list views of a data space.

2.1.2 Dimensional Sorting.

In each of the above slices, the organization of the slice imposes an ordering on the hierarchy. Thus, In a domain slice, the arrangement of the attributes within the slice act as constraints on the information associated with the attributes presented. In the first slice, if Composer is placed after Form, the selected form will determine which composers are listed: only those who set works in the selected form will be listed under composer. Putting Composer before Period allows one to see quickly which period is associated with a given composer. This arrangement of the slice turns out to be an interesting opportunity for learning more than a single association of a period with a composer, since in certain cases, the composer is associated with multiple periods (Bach with Classic and Baroque, for instance). Thus, this one selection retrieves both the requested information, and potentially reveals meta-information about the construction of the classical music domain.

Hierarchical rearrangement is not to be confused with sorting order within a fixed hierarchy. That is, in a fixed hierarchy such as Period — Composer — Form — Arrangement — Composition, one might choose to sort the composer column alphabetically or by date of birth. This sorting has no effect on the current instances associated with each attribute. Reordering the attributes in a hierarchy, as we explain in more detail in the formal model description below, causes instances associated with each attribute to be reassessed, both in terms of the data associated with that attribute, as well as how that rearrangement effects it neighbours on either side of it, since a dimensional sort - a rearrangement of attributes in the hierarchy - effectively creates a new query within the slice.

Rationale: Dimensional sorting lets the user organize the domain in a manner that suits their current interests/knowledge. Providing multiple perspectives on an information space has long been valued by hypertext researchers not only for access for but the value of learning more about a domain [18, 24] of building knowledge by seeing associations that would not otherwise be apparent. It also lets the explorer move into the domain from a locus of familiarity. For someone less familiar with a domain (a domain-naive user), dimensional sorting may provide an approach into a domain that would not otherwise be available. Someone who knows little about classical music, but who once took piano lessons may find it easier to access the domain by privileging Instrument as the first attribute in the hierarchical slice. By supporting dimensional sorting to improve access from multiple perspectives, we are not explicitly modelling users for a particular version of a domain, but are letting users determine the domain version for their needs/tasks.

2.1.3 Domain Substitution, Expansion and Contraction

Substitution, Expansion and Contraction are variations on Dimensional Sorting. The key difference conceptually is that unlike Dimensional Sorting, these other techniques alter the slice through the space. Substitution can fundamentally alter the slice by potentially replacing each attribute with another in the domain. Expansion/Contraction can also have a considerable effect on the slice by increasing or decreasing the scope of the slice.

Rationale: The refinements of substitution, expansion and contraction provide one more lightweight mechanisms by which users can explore a domain. Substitution provides ways to engage what-if scenarios within a domain space: what if the user extends the slice Period — Composer — Form — Arrangement — Composition with Recording — Instrumentation — RecordingYear. From exploring this extended version of the data, users may learn that period instrumentation for Bach's keyboard work would not include its most popular form of representation, the piano. Similarly, by substituting dimensional values one can gradually reformulate the slice to support new perspectives on an attribute. By translating an Historical slice gradually into a Recordings slice, one might also see that some very familiar works by Bach, like the solo cello sonatas, have been rearranged for a variety of instruments repeatedly over the centuries, into one of the latest for five string electric bass but have not been frequently recorded, whereas the same arrangement of Beethoven's Fifth symphony has been rerecorded by countless orchestras. In other words, by facilitating manipulation of the slices through substitution, expansion and contraction, we facilitate more ways to explore and therefore provide more ways to understand the domain and its inter-relations - and perhaps the biases informing its organization.

3. RELATED WORK

mSpaces are largely informed by work in Human Computer Interaction in direct manipulation interfaces [5] and query previews [19]. Direct Manipulation interfaces support immediate feedback of manipulations in an interface. Thus, moving a slider up in an onscreen volume control and hearing the volume level increase is an example of direct manipulation. Changing the attribute in an mSpace column and immediately seeing the change and its effect is an example of direct manipulation. This style of interaction design is in contrast to user interfaces which change based on inferences about user interactions. Adaptive Interfaces [25], for instance, might adjust the ordering of elements in a menu based on frequency of use. Evaluations of such adaptive models have rarely been positive [15].

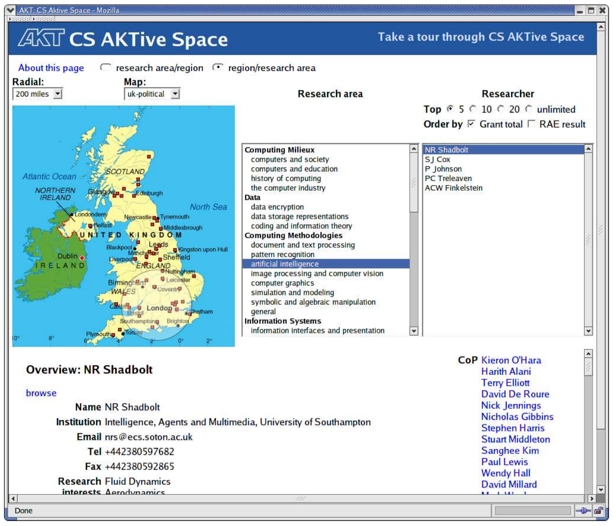
Query previews are a branch of direct manipulation user interfaces specifically designed for query formulation. They allow users to construct queries visually rather than textually. An interface on recipes, for instance, may present a variety of sliders with values associated with each so that one can select the degree of spiciness in a dish, whether the dish is fish, meat, or vegetarian; how long it takes to prepare and so on. Based on these constraints, the search engine returns a list of appropriate recipes. The design of available query attributes values eliminates the possibility of returning an empty set. In this way, users gain an easy method to quickly construct potentially more complex queries than a keyword search would allow, and equally can evaluate the results, tweak the parameters of the query, and run the revised query again just as quickly

Like query previews, mSpaces are direct manipulation, visual queries. Unlike most instantiations of such queries, mSpaces extent the preview query space, foreground domain manipulation affordances. Not only can users determine the attributes in the domain to query, they can organize them to privilege their evolving explorations of the domain.

Most direct manipulation interfaces are hand-crafted for a given application: the application adopts the heuristics for direct manipulation or for preview queries and creates an implementation to support that approach. In the following section, we present the formal model for the interaction design described above. Our goal in doing so is to provide Semantic Web designers with an interaction model they can first formally evaluate, and next instantiate to rapidly provide the described interaction affordances.

Our description of mSpace using Semantic Web technologies bears some similarity to work in the Formal Concept Analysis community, particularly in the area of ontology-aware browser interfaces for information retrieval and discovery [11, 8]. These systems construct a concept lattice which is used to provide support for navigation through an information space; this concept lattice represents the different hierarchies that would result from ordering entities according to their attributes. mSpace also provides such a hierarchical view of the entities in the system, but in a more structured manner that affords the user greater control over their exploration of the information space.

The CS AKTiveSpace (CAS) Semantic Web application [23] which recently won the Semantic Web Challenge for best Semantic Web Application [1] was very much informed by the interaction design expressed above. The goal of the application was to support user-determined exploration of the domain Computer Science Research in the UK. Underneath the UI, the application exploits a wide range of semantically heterogeneous and distributed content. The content currently comprises around ten million RDF triples, with information constantly being harvested and updated. The content in the application is mediated through an ontology constructed for the application domain, the AKT Reference Ontology [1], and incorporates components from other published ontologies as well. The challenge for the UI which an mSpace approach addressed, was to make the space managable enough to support meaningful interrogation by users. The UI would afford users the ability to explore the domain such that they could readily compare, for instance, what areas of the domain are researched in different areas of the country, who the top rated researchers are in a given domain; who do these researchers work with the most, what have they published. These are each results of queries that would be complex and potentially too cumbersome to construct to facilitate real-time shifting between one formation of a query and another. By modeling the CAS UI on the mSpace model, we were able to hide the complexity of the queries from the user, and support real-time query manipulation and ontology exploration, where data instances are revealed in context.


AKT: CS Aktive Space - Mozilla

CS AKTive Space — Take a tour through CS AKTive Space

About this page ( ) research area/region (•) region/research area

Radial: 200 miles

Map: uk-political

Research area

Computing Milieux
computers and society
computers and education
history of computing
the computer industry
Data
data encryption
data storage representations
coding and information theory
Computing Methodologies
document and text processing
pattern recognition
artificial intelligence
image processing and computer vision
computer graphics
simulation and modeling
symbolic and algebraic manipulation
general
Information Systems
information interfaces and presentation

Researcher

Top (•) 5 ( ) 10 ( ) 20 ( ) unlimited

Order by [✓] Grant total [ ] RAE result

NR Shadbolt
SJ Cox
P Johnson
PC Treleaven
ACW Finkelstein

Overview: NR Shadbolt

browse

Name	NR Shadbolt
Institution	Intelligence, Agents and Multimedia, University of Southampton
Email	nrs@ecs.soton.ac.uk
Tel	+442380597682
Fax	+442380592865
Research interests	Fluid Dynamics Aerodynamics

CoP Kieron O'Hara
Harith Alani
Terry Elliott
David De Roure
Nick Jennings
Nicholas Gibbins
Stephen Harris
Stuart Middleton
Sanghee Kim
Paul Lewis
Wendy Hall
David Millard

Done

Figure 2: CS AKTiveSpace: an example of an mSpace interface

A large goal of the mSpace work has been to formalize the interaction model in order to create a generalizeable model for semantically informed information spaces. CAS is the first application of the mSpace model mapped into a Semantic Web back end. Through the development process of implementing mSpace affordances, such as slices and dimensional sorting, against the AKT ontology, we have been better able to determine where the issues would be in formalizing the model for general application deployment. Indeed, the effort to deploy the mSpace interaction design has reinforced the value of formalizing the model. It will make building mSpace-informed applications like CAS more principled. This is important on a practical level, since based on the demonstrations we have given of CAS, we have been approached to design other CAS-inspired applications for granting councils, engine manufacturers and a space agency to name a few. Below, we touch on two of the ways that the formal mSpace model will improve mSpace-informed application development.

Query generation: The query generation performed by the CAS application is based on programmatic query constructions performed in the application logic of the program. This can make extracting and debugging the queries unnecessarily complex, whereas the constraint matrix suggested by this model, described below, will provide a cleaner and more general abstraction for the query generation algorithm.

Inter-column constraints: The current version of the CAS application is based on adjacent constraints, which has the consequence that as columns are rearranged (one of the more common interactions) the semantics of the column positions changes, causing undesirable and confusing changes in the contents of the columns. With the benefit of the model we see that privileged column constraints would have been more appropriate for this application (the user is attempting to narrow down selections, approaching a target group of individuals). This approach would have the advantage that re-arranging the columns would not affect the semantics of column selection.

In the next section, therefore, we describe the formal representation of the interaction model described above.

[1]http://challenge.semanticweb.org/

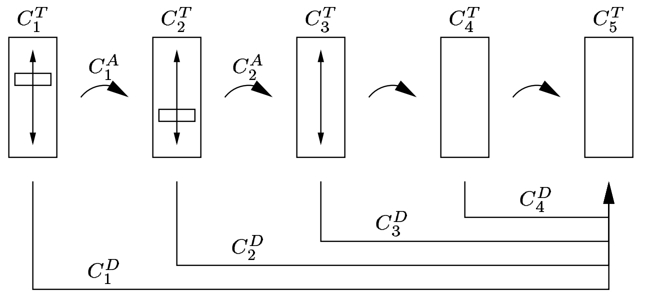


Figure 3: mSpace column constraints

4. MODELLING MSPACE INTERFACES

As described in the previous section and in [13], an mSpace interface consists of a series of user interface controls, which we call *columns*, each of which presents a selection of objects for the user to choose from. A user's interaction with these column progresses from the left to the right. The selection that they make in a column affects the selections which are offered in the subsequent columns that lie to its right. The selection offered by each column is a collection of objects of the same conceptual type; a list of people, or of institutions, or of publications, for example. Each of the columns represents some facet of the objects being searched, and a selection within a column can be thought of as specifying some dimension in the sparse multidimensional space of a knowledge base (where each class is treated as a dimension).

In this respect, the interface resembles the menu-based interfaces used for exploring databases; a user chooses from lists of options that lead the user through the formulation of a query. The mSpace method differs in that a context (or *column layout*) may be dynamically reconfigured so as to place one dimension at a higher importance than others.

4.1 Constraints

Our basic approach in constructing this model is to consider each column as representing some set of objects. We treat this set as a class which is specified by a class expression in a description logic-based language such as OWL [16] or DAML+OIL [7]; the objects presented in a column are the extension of the class expression for that column. We choose to model the columns and the interactions between them in terms of constraints which determine the choice being presented in each column and the effect that a selection has on other columns. These constraints are expressed as class expressions in the description logic language. The overall class expression for a column is the intersection of the separate constraint class expressions that apply to that column.

We identify two broad classes of constraint in the above approach, type constraints and selection constraints.

4.1.1 Type constraints

Each column has an intrinsic constraint which governs the type of the objects that are listed in that column. This may vary from a simple named class to a more complex class expression (for example, 'all higher education institutions within the UK that were rated 5* in the 2001 Research Assessment Exercise'). In Figure 4.1, type constraints are denoted by C_n^T, where n is the constrained column, and are invariant with selection.

4.1.2 Selection constraints

A selection in a column affects the choices that are presented in other columns. Selection constraints represent the relation between a particular column and the other columns in a layout (or between one class of objects and another class of objects in the ontology), and are applied to a column in addition to any other constraints it may have.

Selection constraints take the form of a restriction on a property which relates the two columns, to the value which was selected These restrictions are typically (but not always) `owl:hasValue` restrictions, written in description logic terms as $\exists R.\{s\}$, where s is the selected value and R is the relating property). Multiple selections in a column are handled by forming the union of the constraints for the individual selections (or by adding both selections to the enumerated class, as in $\exists R.\{s_1, s_2\}$). Selection constraints are particular to a given pair of column types; a constraint which works between publications and authors will not work between publications and publishers, because the underlying relations which link these types are different.

The set of constraints for a layout must provide a means to relate each column to every other column, either directly, or indirectly by chaining more than one constraint together. For a layout with n columns, the *basis set* of constraints contains $n-1$ constraints. It is possible to construct systems with more than this minimum number of basic constraints, but this raises the possibility of inconsistency when there is more than one way to relate one column to another, possibly by chaining constraints, and those relations are incompatible or have different meanings. If we restrict ourselves to basis sets of selection constraints, there are two common approaches to forming constraints which we can use in mSpace systems, each with its advantages and disadvantages.

4.1.3 Adjacent selection constraints

In the first approach, each column provides a constraint, which we call an *adjacent selection constraint*, to its immediate successor in the layout. In Figure 4.1, this type of constraint is denoted by C_n^A where n is the column from whose selection the constraint is derived (the constraint is applied to column $n+1$). Since this constraint set is a basis, we can determine the pairwise effect of a selection on any other column by composing the basis constraints. For example, in Figure 4.1 the constraint on column three from column one is $C_1^A \circ C_2^A$.

In practice, these composed constraints are not used in implementations (where the constraints are evaluated column by column from left to right); we describe them in order to illustrate the parallels between the two approaches to building a basis set.

4.1.4 Distinguished column selection constraints

The second approach to constructing basis sets for selection constraints is concerned with the case where the display of 'dead end' selections is suppressed. A dead end selection in a column is one that, when made, will produce no options to choose from in the final column. For example, if we have a column containing a list of countries that includes the Vatican City, and our final column contains a list of computer science researchers, but we have no information about the state of computer science research in the Vatican, then we remove Vatican City as a choice in the country column. In effect, we are limiting the system in such a way as to only let the user answer questions that it can answer). In this approach, we introduce a basis set of selection constraints between the final column (in the general case, any distinguished column, but we assume that the distinguished column is the final column in order to clarify this explanation) and each of the other columns. We call these constraints *distinguished column selection constraints*; in Figure 4.1, they are denoted by C_n^D, where n is the column from whose selection the constraint is derived.

This set of constraints is slightly more involved, and consists of a separate constraint from each column to the final column (for that between the penultimate column and the final column, this may well be the same as the adjacent selection constraint). As with adjacent selection constraints, these constraints depend on the types of the columns involved; two layouts whose final columns differed in type would have different sets of constraints.

As with the previous constraint scheme, constraints may be composed in order to give the effect of one column on another; in Figure 4.1, the constraint from column one to column three is $C_1^D \circ (C_3^D)^-$, where $(C_n^D)^-$ is the inverse of the constraint C_n^D. Unlike the previous scheme, we construct the complete set of these composed constraints and use them to augment the type constraints on each column in order to produce the no-dead ends behaviour described above. If no selection has been made in a column, the class expression in a selection constraint restriction is that used for the type constraint of the source column, otherwise it is an enumerated class containing the selected individuals.

For example, if we have two columns with intrinsic types C_1 and C_2, and column two is a distinguished column, the selection constraint from column one to column two might be a restriction of the form $\exists R.C_1$, where R is the relation which relates objects of type C_2 to objects of type C_1. The overall constraint on column two is therefore $C_2 \sqcap \exists R.C_1$ (the intersection of the intrinsic type constraint and the selection constraint from column one). Conversely, this constraint is also applied in reverse from column two to column one in order to suppress the display of dead end selections; this constraint is $C_1 \sqcap \exists R^-.C_2$. When a selection is made in column one, the constraint from column one to column two is modified by changing the class expression in the restriction from C_1 to $\{s_1, s_2, \ldots\}$, where s_1, s_2 and so on are the selections that have been made in column one.

The exception to this rule in implemented systems is that the replacement of type constraint class expressions with selection enumerated classes happens only in columns which succeed the column in which the selection was made. In this way, a selection in a column does not affect the options offered in its predecessors; the constraint from column two to column one is still $C_1 \sqcap \exists R^-.C_2$.

Due to the issues of consistency in the constraint set, either adjacent or distinguished column constraints may be used as a basis, but not both at the same time. The choice of which is used depends on the desired behaviour of the mSpace; in some circumstances, the information that a particular path through the space terminates in a dead end is useful knowledge, so adjacency selection constraints would be more appropriate. In addition, the choice of selection constraints changes the behaviour of an mSpace system under some operations, as described in the next section.

4.1.5 Constraint example

As an example of the above constraint schemes, consider a mSpace interface for exploring countries, the institutions located in those countries, the projects in which those institutions are involved, and the people who work on the projects. We choose an initial layout that consists of a column of institutions, then of projects, and finally of people, as shown in Figure 4.1.5. The ontology on which the interface is based defines the four named classes *Country*, *Institution*, *Project* and *Person*, each of which is used as the type constraint on the relevant column. Institutions are related to projects by the *contributes-to* property, people to projects by the *works-on* property, people to institutions by the *works-at* property, institutions to countries by the *located-in* property and people to countries by the *born-in* property.

The adjacent selection basis constraint from countries to institutions is of the form $\exists$ *located-in*.*Country*, while that from institutions to projects is $\exists$ *contributes-to*$^-$.*Institution*, and that from projects to people is $\exists$ *works-on*.*Project* Initially, the only constraints applied to each column are the type constraints and the initial forms of the selection constraints given above. As selections are made in columns, the selection constraints are modified so that the (named) class in the local range restriction is replaced by an enumerated class whose members are the selected instances (we refer to such post-selection constraints as *concrete selection constraints*).

The distinguished column selection constraint basis set consists of $\exists$ *works-at*.*Institution*, which runs from institution to person, $\exists$ *works-on*.*Project*, which runs from from project to person, and $\exists$ *born-in*.*Country*, which runs from country to person. As in the adjacent case, these constraints are modified as selections are made. The concrete selection constraint between the institution and project columns, which takes into account the existence of people which satisfy the selections made in these columns, is:

$$\exists \textit{works-on}^-.(\exists \textit{works-at}.\{s_1, s_2, \ldots\})$$

where s_1, s_2 and so on are the individuals which have been selected in the *Project* column.

In this example, where the selection constraints are straightforward `owl:hasValue` restrictions on single properties, the pairwise selection constraint is an `owl:hasValue` restriction on the composition of one of the properties with the inverse of the other (eg. $\exists$ (*works-on*$^-$ $\circ$ *works-at*).$\{s_{inst}\}$).

So far, we have described layouts which use `owl:hasValue` restrictions as selection constraint class expressions, but these are not the only possibility. We can also form constraints of the form $\forall R.\{s_1, s_2 \ldots\}$ (in OWL terms, an `owl:allValuesFrom` restriction to an enumerated class defined using `owl:oneOf`), and can incorporate other restrictions which are not parameterised by selections.

4.2 Layout operations

The description of intra- and inter-column constraints above assumes a fixed layout of columns, whereas an mSpace interface allows the user to modify the layout at will. We define two primitive operations on a layout: the *addition* of a column within the layout, and the *removal* of a column from the layout. As they are, these primitive operations are unlikely to be supported in a given mSpace interface because they are at too low a level to fit well within the paradigm of dimensional exploration introduced by mSpace. Instead, we define three distinct types of higher-level operation which can be expressed as a composition of the primitive operations, and which an mSpace interface would be expected to support (illustrated in Figure 5):

Expansion: A new column may be added to the right of the existing columns

Substitution: A column within the layout may be switched with a column of a different type

Transposition: The positions of two columns within a layout may be exchanged. This is a specific example of a dimensional sorting operation – there are others.

The operations that may be performed on a column affect the constraints on a given layout to a greater or lesser extent depending on the selection constraint scheme being used. In turn, the change

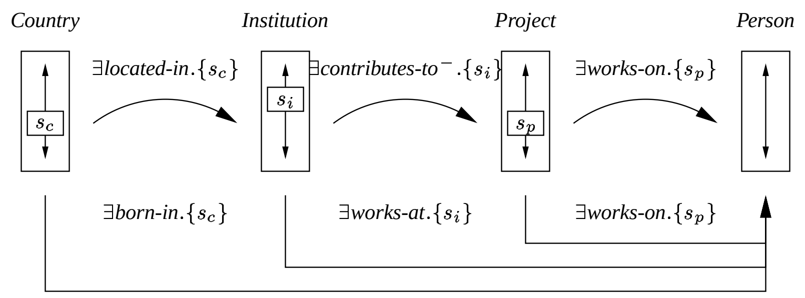

Country
Institution
Project
Person
s_c
s_i
s_p
$\exists$located-in.$\{s_c\}$
$\exists$contributes-to$^-$.$\{s_i\}$
$\exists$works-on.$\{s_p\}$
$\exists$born-in.$\{s_c\}$
$\exists$works-at.$\{s_i\}$
$\exists$works-on.$\{s_p\}$

Figure 4: mSpace example

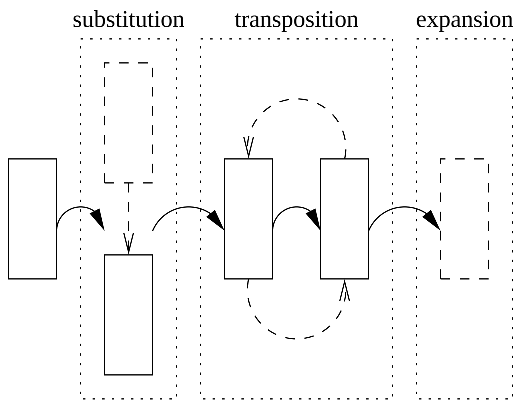


Figure 5: mSpace operations

	Adjacent		Distinguished column	
	first/last	other	distinguished	other
add	$+1$	$-1, +2$	$\pm(n-1)$	$+1$
remove	-1	$-2, +1$	$\pm(n-1)$	-1
expand	$+1$	—	$\pm(n-1)$	$+1$
transpose	± 3	± 4	$\pm(n-1)$	0
substitute	± 1	± 2	$\pm(n-1)$	± 1

Table 1: Operation cost on column based on constraint scheme

in the constraint set caused by the operation may affect the selections which the user has made (by affecting the set of options from which they made the selection), which in turn may change more instantiated constraints, and so on.

We describe operations which cause the invalidation of selections as *disruptive*. For example, if a selection has been made in a column, and the column is substituted for one of a different kind, the selection is lost, which affects any presented options or selections in subsequent columns. Similar behaviour is to be expected in the event that two columns are transposed, although it may be possible to retain the selections in the columns provided they do not violate the new selection constraints.

In the general case, we cannot easily characterise this disruptive propagation of changes, but we can determine what the change to the constraint set will be in response to the initial operation for the two selection constraint schemes. We describe the changes brought about by an operation in an abstract fashion, by counting the number of selection constraints which must be changed. These are listed in Table 1; $+n$ indicates that n constraints are added to the set, $-n$ that n are removed and $\pm n$ that n are changed.

Since the observed flow of information in an mSpace interface runs from left to right, an operation which changes a column in a layout using adjacent selection constraints is likely to change those columns which lie to its right. For example, the expansion of an existing layout under the adjacent scheme by the addition of a new final column is a non-disruptive operation which does not involve the removal or replacement of existing constraints (shown as $+1$ in Table 1).

Conversely, under a distinguished column scheme (where the final column is the distinguished column), all the selection constraints are replaced with ones suitable for use with the new final column. In this case, the change in the type of the final column leads to the introduction of a new set of selection constraints, which may be violated by the existing selections. Since constraints are applied in a pairwise manner under this scheme (going from one column of a pair to the distinguished column, and then back to the other column in the pair), a change in the type of the distinguished column affects all selection constraints, which raises the prospect that the selections in all existing columns may be invalidated (shown as $\pm(n-1)$ in Table 1).

Similar behaviours exist for the operations of transposition and substitution under either of the constraint schemes described above. In general, adjacent selection constraints yield more disruptive operations for small layouts with less than five columns (there are no distinguished columns, so the cost is mostly constant across the columns in a layout, the first or last columns being an obvious exception).

In contrast, distinguished column selection constraints designate one column as special and express all selection constraints in terms of that column. Operations on columns other than the distinguished column are less disruptive than their counterparts under adjacent selection constraints (note the values for transposition in Table 1). However, operations on the distinguished column become significantly more expensive because they change the entire constraint set.

Thus, a rule of thumb which informs a designer's choice of selection constraint scheme is this. For interfaces where dead end suppression is considered essential, or where layouts have fewer than five columns, or where the expansion operation is not to be supported, distinguished column selection constraints are the appropriate approach, otherwise adjacent selection constraints should be chosen. One proviso to this is that expansion under distinguished column constraints, where the final column is not the distinguished column, are no more expensive than expansion under adjacent constraints.

4.3 Permitted layouts

In certain circumstances, an arrangement of columns may not make sense. For example, a column which required the user to pick from a list of university departments should not be followed by a column which requires the user to pick from a list of universities; there is a dependency between the types of these columns which makes the second column irrelevant. This is particularly true of layouts which use the adjacent selection constraint scheme, where a column directly depends on its predecessor.

We express this behaviour in a successor matrix which relates each column type to the column types after which it may appear; the column types used in this matrix are the type constraint class expressions described in the previous section. When a user is presented with a layout, the set of permissible operations which they may use to rearrange the layout is constructed by consulting this matrix. For example, the user is presented with a list of column types which may be used to expand the current layout that is generated by taking the set of all possible column types and removing those which are prohibited by the columns already in the layout.

Similarly, the transposition of two columns is a two part operation. The user selects the first of the columns to be transposed, then the columns with which that column may be swapped (which would yield a permissible layout when swapped) are indicated. The user then selects the second column from those indicated, and the layout is rearranged.

4.4 Column presentation

The model of mSpace outlined above concerns itself primarily with the generation of the abstract choices offered to the user in each column and the propagation of the effects of making a selection through the system, but this is not the only concern. If each column represents an set of objects from which the user must make some choice, the manner in which that set is rendered for presentation to the user is also of importance.

In our prototype mSpace system, CS AKTiveSpace, we have implemented a column view for selecting UK higher education institutions which is rendered as a map of the UK showing the geographical distribution of HEIs. This column need not have been rendered in this fashion; it could have been shown as a simple multipick widget containing the names of the institutions.

In each case, the information required to render the column is obtained from the knowledge base (human-readable labels and latitude/longitude coordinates for the HEIs). The design of an mSpace interface must therefore not only include the set of column types, their permitted combinations and the constraints which relate them, but also the alternate rendering styles for each column.

Our CS AKTiveSpace system also illustrates the use of ordering and limiting constraints on columns. The final column contains a list of people (filtered by the preceding columns) which may be ordered according to various criteria (we allow ordering by total grant income and by the research rating of their institution). Similarly, the number of people shown in this column may be limited to the first five, ten, etc. This behaviour is separate from the selection-based behaviour described earlier in this section and is largely a presentational issue; we implement it as a set of filters which apply to the column after the application of inter-column constraints.

We treat the issue of column presentation as a level above that of the abstract column model, on which it has no direct bearing. In this respect, mSpace interfaces fit well within the model-view-controller paradigm, the model can be effectively decoupled from the view/controller pairs through the adoption of a clean API by which the view interacts with the model.

4.5 Detail view presentation

As with column presentation above, we consider the generation of the detail view provides the user with contextual information as they interact with the system to be a purely presentational issue that has little to no impact on the interactions between the columns described in our formal model. The detail view provides information (at a fairly coarse granularity) about the entity that was most recently the focus of the user's attention, namely the most recent selection that they made in a column.

5. IMPLEMENTATION

The characterisation of an mSpace user interface in terms of description logic expressions given above should be considered to be an abstract description of such a user interface, and not a detailed specification of an implementation itself.

In our prototype Semantic Web application, CS AKTiveSpace, we chose to limit the class expressions used as selection constraints to be `owl:hasValue` restrictions. We further chose to implement these constraints in a naïve manner which does not require recourse to a DL reasoner; constraints were represented as triple patterns containing variables which were bound as selections are made. For example, the constraint $\exists$ *works-at*.*Institution* (used as a distinguished column selection constraint in the example) would be replaced with the triple patterns $\langle$*?person*, *works-at*, *?institution*$\rangle$ and $\langle$*?institution*, *rdf:type*, *Institution*$\rangle$. These triple patterns were expressed in the RDQL language, and used as queries to generate the values with which panels were populated.

This approach has the advantage of simplicity, because a constraint and its inverse have exactly the same form. The triple patterns do not assume that the constraint will be read in the direction of a relation or of its reverse; the reading of the constraint depends on which variable is bound (by a selection), and which is free (and generates potential selection with which a column is populated). In addition, the RDQL query language is supported by a number of RDF stores and inference engines, including 3store [10] (on which we have built our CS AKTiveSpace application), Sesame [3] and Jena [12], so providing some flexibility in our choice of infrastructure.

The disadvantage of this approach is that it permits only one particular type of constraint, and that the construction and composition of disjunctive query patterns (as would be the case when multiple selections are made in a column), is clumsy in RDQL.

6. CONCLUSION AND FURTHER WORK

In this paper, we have presented a treatment of the mSpace style of user interfaces which builds on Semantic Web technologies. In doing so, we open up possibilities for more strongly correlating the interaction behaviours with the semantics inherent in the information represented by the system.

Our plans for future work include the development of a framework to facilitate the construction of mSpace-based systems (drawing on our experiences of implementing our prototype CS AKTiveSpace system), and the investigation of techniques for automating the development of mSpace systems from ontologies. From this, we intend to develop an API and toolset to make deploying mSpace in OWL-based Semantic Web applications a plug and play operation. To this end, part of our goal with this paper is to generate discussion about and feedback for our model, to better inform the development of an API.

7. ACKNOWLEDGEMENTS

This work was supported by the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC). The AKT IRC is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01 and comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

The authors would like to thank Nigel Shadbolt and Srinandan Dasmahapatra for their comments on this work.

8. REFERENCES

[1] The AKT Reference Ontology. `http://www.aktors.org/publications/ontology/`, 2002.

[2] B. B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Trans. Graph.*, 21(4):833–854, 2002.

[3] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. In *Spinning the Semantic Web*, pages 197–221. MIT Press, 2003.

[4] H. Chen and S. Dumais. Bringing order to the web: Automatically categorizing search results. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'00)*, pages 145–152, 2000.

[5] R. Chimera and B. Shneiderman. An exploratory evaluation of three interfaces for browsing large hierarchical tables of contents. *ACM Transactions on Information Systems (TOIS)*, 12, 1994.

[6] A. Cockburn and B. McKenzie. Evaluating the effectiveness of spatial memory in 2d and 3d physical and virtual environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 203–210. ACM Press, 2002.

[7] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. DAML+OIL (March 2001) Reference Description. W3C Note, World Wide Web Consortium, Dec. 2001.

[8] P. Eklund and R. Cole. Structured ontology and information retrieval for email search and discovery. In *Foundations of Intelligent Systems, 13th International Symposium (ISMIS 2002)*, pages 75–84, 2001.

[9] V. Geroimenko and C. Chen, editors. *Visualizing the Semantic Web*. Springer, 2002.

[10] S. Harris and N. Gibbins. 3store: Efficient Bulk RDF Storage. In *Proceedings of the First International Workshop on Practical and Scalable Semantic Systems (PSSS2003)*, pages 3–17, Sanibel Island, Florida, USA, Oct. 2003.

[11] M. Kim and P. Compton. A web-based browsing mechamism based on conceptual structures. In *Conceptual Structures: Extracting and Representing Semantics: Contributions to the 9th International Conference on Conceptual Structures (ICCS2001)*, pages 47–60, 2001.

[12] H.-P. Labs. The Jena Semantic Web Toolkit. Web page, Hewlett-Packard Labs, 2003. `http://www.hpl.hp.com/semweb/jena.htm`.

[13] m.c. schraefel, M. Karam, and S. Zhao. mspace: interaction design for user-determined, adaptable domain exploration in hypermedia. In *Proceedings of the AH2003 Workshop on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 217–235, Nottingham, UK, June 2003.

[14] m.c. schraefel, M. Karam, and S. Zhao. Preview cues for exploring domain hierarchies. In *Proceedings of Interact 2003*, Switzerland, 2003.

[15] J. McGrenere, R. M. Baecker, and K. S. Booth. An evaluation of a multiple interface design solution for bloated software. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 164–170. ACM Press, 2002.

[16] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. Candidate recommendation, World Wide Web Consortium, Aug. 2003. `http://www.w3.org/TR/owl-features/`.

[17] D. Modjeska and J. Waterworth. Effects of desktop 3d world design on user navigation and search performance. In *Proceedings of Information Visualization 2000*, pages 215–220. IEEE, 2000.

[18] C. Neuwirth, D. K. R. Chimera, and T. Gillespie. The notes program: A hypertext application for writing from source texts. In *Proceeding of the ACM Conference on Hypertext*, pages 121–141, 1987.

[19] C. Plaisant, B. Shneiderman, K. Doan, and T. Bruns. Interface and data architecture for query preview in networked information systems. *ACM Transactions on Information Systems (TOIS)*, 17(3):320–341, 1999.

[20] K. Risden, M. Czerwinski, T. Munzner, and D. Cook. An initial examination of ease of use for 2d and 3d information visualizations of web content. *IJHCS*, 53, 2000.

[21] G. Robertson, K. Cameron, M. Czerwinski, and D. Robbins. Polyarchy visualization: visualizing multiple intersecting hierarchies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 423–430. ACM Press, 2002.

[22] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubs, and M. Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Transactions on Computer Human Interaction (TOCHI)*, pages 162–188, 1996.

[23] N. Shadbolt, monica schraefel, N. Gibbins, H. Glaser, and S. Harris. CS AKTiveSpace: Representing Computer Science in the Semantic Web. `http://triplestore.aktors.org/tmp/www2004-cas.pdf`, 2003.

[24] F. Shipman, J. Moore, P. Maloor, H. Hsieh, and R. Akkapeddi. Spatial hypertext: Semantics happen: Knowledge building in spatial hypertext. In *Proceedings of the Thirteenth Conference on Hypertext and Hypermedia*, pages 25–34, 2002.

[25] P. N. Sukaviriya and J. D. Foley. Supporting adaptive interfaces in a knowledge-based user interface environment. In *Proceedings of the 1st international conference on Intelligent user interfaces*, pages 107–113. ACM Press, 1993.

[26] Vivisimo. Vivisimo document clustering engine. `http://www.vivisimo.com/`.