Chapter 3
Cognitive Semantic Categories as a Basis for a Prototype Adaptive Information System

Evangelos Kapros and Simon McGinnes

Abstract A software application is demonstrated which exhibits conceptual data independence. The application provides domain-specific functionality, yet its structure is domain-independent. Separation between conceptual model and structure is achieved by encoding models as data and interpreting them at run-time. The overall goal is to reduce cost and delay when conceptual models change, and to provide application functionality in new domains without constructing new applications. Several conceptual models are used, to illustrate domain-specific behavior in multiple domains. Results suggest that domain-independent application design can reduce the need for application development and maintenance effort, since each domain-independent application can function in multiple domains and adapts smoothly to changing conceptual models. This is especially meaningful for end users who usually have no development skills and rely on spreadsheet and database driven applications.

3.1 Introduction

Current best practice in software design produces applications that are domain-specific in both behavior *and* structure. For example, accounting software might be constructed from classes representing accounts and account entries, and might store data in *Account* and *Entry* database tables. The application's architecture is described as domain-specific because its class and table structures mirror the concepts (entity types and their relationships) in the application domain's conceptual model.

The use of domain-specific architecture is a familiar and relatively simple way of constructing software. But it leads to high cost and delay when software must be altered to match new or modified conceptual models. This remains a barrier to

E. Kapros (✉) · S. McGinnes
The University of Dublin, Dublin, Ireland
e-mail: ekapros@tcd.ie

S. McGinnes
e-mail: Simon.McGinnes@tcd.ie

system evolution despite long attention from researchers (Hick and Hainaut 2006; Hartung et al. 2011). It also makes it necessary to do development work when new domain-specific functionality is required.

In conventional software design, software architectures are based on the assumption that the end user's mental concepts are relatively static. *Conceptual data dependence* is the practice of embedding these mental concepts in software architectures. Our goal is to construct applications which exhibit *conceptual data independence*, such that minimal work is required in respect of new or changed conceptual models. The motivation is to reduce the cost and delay that organizations incur when they develop and maintain software applications to match new or altered conceptual models. Development work causes cost and delay which mainly affects small and medium enterprises and organizations, which employ staff with typically little or no programming skills. Thus, they face the dilemma to buy applications that match their requirements or fund the development of custom-made applications. However, this dilemma is usually avoided and organizations rely on simple tools such as spreadsheets (Chan and Storey 1996; Raden 2005).

We propose to reduce cost and delay by building an information system that is adaptive to model changes (Adaptive Information System, or AIS). We implemented this idea to show its feasibility, and present a software application which is simultaneously the authoring environment and the user interface of applications which exhibit conceptual data independence. That is, the end users can manage the model and the data through the same user interface. Moreover, the concepts of the model and the data are represented in user-friendly forms. Thus, expert help concerning change is minimized. In addition, conceptual data independence has implications for the visual design of user interfaces.

3.2 Related Work

3.2.1 Relational Databases and Object-Oriented Design

The relational model proposed by Codd (1970) provides a standard way of translating concepts into data structures. A table represents a concept, while columns represent the concept's attributes. The concepts that describe the structure of the database form its *schema*. Research in Schema Evolution focuses on the problem of adapting a database schema to changes. This research field shows that changes in schemas represent a significant cost to organizations. In Curino et al. (2008) changes in the database schema are reported to affect up to 70 % of queries, which have to be manually reconfigured. Some theoretical models to address this problem have been constructed, but real systems incorporating schema evolution functionality are hard to find (Roddick et al. 2000).

Other types of software design are subject to the same kinds of problem. Object-oriented design in programming and in databases is one example. In object-oriented design, concepts are represented as classes. Classes serve as blueprints for objects, which are specific instances of the concept. Changes to the underlying conceptual structure implemented in a class structure make it necessary to alter the classes and their relationships. This, in turn, makes it necessary to modify code which refers to the altered classes. Hence there can be a high overhead cost arising from changes to the underlying conceptual model of an application constructed using conventional object-oriented design.

3.2.2 Ontologies and the Semantic Web

An ontology provides a semantic network of predefined concepts intended to describe the universe of knowledge for a particular domain. Domain-specific applications may define new end-user concepts as sub-concepts of the existing concepts in the ontology.

It has been proposed (Berners-Lee et al. 2006; Alani et al. 2005, 2008) that web applications should use ontologies as well. The so-called Semantic Web applications would, then, be able to share data freely using as mediators these predefined concepts, without any need for prior programming. For this to work in the general case, ontologies would have to be capable of being integrated with a common ontology. Various semi-automatic tools have been developed for this task (McGuinness et al. 2000; Noy and Musen 2000).

However, this is a non-trivial challenge. A lack of standardization in end-user concepts leads to the Tower of Babel (Fonseca and Martin 2004) problem: the creation, in various ontologies, of incompatible definitions for the same entity. Moreover, since the existing ontologies are domain-specific, no large-scale cross-domain implementations exist. For this reason, it is still unclear how web meta-data would follow the conceptual vocabulary of the ontologies (Shirky 2003).

The idea of handling arbitrary schemas in software applications has not been previously directly addressed. However, work on ontologies has given useful results on change in semantics while using automatically generated interfaces (Ertl et al. 2011; Wach 2011). Similarly, work on dynamic data management has given useful results (Fein et al. 2011; Kennedy et al. 2011; Sun et al. 2011) but has not, in general, addressed user-interface or usability issues. While there have been design efforts in web browsers such as LENA (LENA—a Fresnel LEns based RDF/Linked Data NAvigator with SPARQL selector support n.d.) and Tabulator (Berners-Lee et al. 2007) that offer views that depend on semantics, they are targeted to software developers and not end users (SPARQL knowledge is essential). Moreover, they differentiate the authoring environment of the applications from the applications themselves, which serves well software developers but might be confusing to end users. However, these are useful paradigms and offer valuable ideas for exploration.

3.2.3 Spreadsheets

Research has shown that most organizations still rely on spreadsheets for their data management (Chan and Storey 1996; Raden 2005). There is a number of reasons why that happens, including failure to deliver end-user systems with usable schema evolution. End users have been reported to "shun enterprise solutions" (Raden 2005) and 70 % of them use spreadsheets on a frequent or occasional basis most commonly for "sorting and database facilities" (Chan and Storey 1996). Spreadsheets are error-prone and miss critical database functionality. There exists work on some database functionality in spreadsheets such as managing plural relationships (Bakke et al. 2011), but not on conceptual modelling. Similarly, work on semantic spreadsheets has improved modeling in spreadsheets, but still separates authoring and application (Zhao et al. 2010; Kohlhase and Kohlhase 2011). Moreover, the problem of schema evolution remains, since the practice of conceptual data dependence is still followed.

3.3 Conceptual Data Independence

3.3.1 Soft Schemas

We approach this problem by turning conceptual models into data. Current application design practice embeds conceptual models into software structures (classes, windows, tables, etc.) When building an AIS this practice is avoided. Instead, the AIS is constructed from generic, domain-independent structures. The model-as-data is termed a *soft schema*; in our prototype it is stored as XML, although any logically-equivalent way of storing data would suffice. The soft schema is read and interpreted by the AIS at run-time. The soft schema is a properly normalized relational data model, with some additions, but it is stored as data rather than being hardcoded in application structure.

To provide domain-specific functionality, yet also exhibit conceptual data independence, the AIS must meet several conditions. First, it must react at run-time to a soft schema, providing a user interface which looks and behaves similarly to those of conventional domain-specific applications. This requires the AIS to mimic the design choices of a human designer, in real time. Our approach is to implement automated user interface design heuristics which are applied based on the contents of the soft schema. We provide specialized behavior for different types of data by responding to known semantic categories embedded in the soft schema (see Sect. 3.3.2).

An AIS must also be able to store and retrieve data corresponding to multiple soft schemas with guaranteed data integrity. The AIS has no advance knowledge of the data and schemas it will be used with, and how they may change. An AIS would be of little use if altering a schema rendered previously-stored data unusable, or if it compromised data integrity. So the data corresponding to each soft schema must be able to co-exist and be used with data stored for other soft schemas, regardless of their structures. Our solution to this problem is to store data in a broadly domain-independent way, but to retain intact the conceptual structure for each instance of data. Our prototype meets that requirement by storing the data using XML and using XML tags to denote structure. XML was chosen in this instance because of its simplicity and flexibility which are desirable properties for building a proof-of-concept prototype. But, again, any logically-equivalent storage mechanism (such as RDF or others) would suffice.

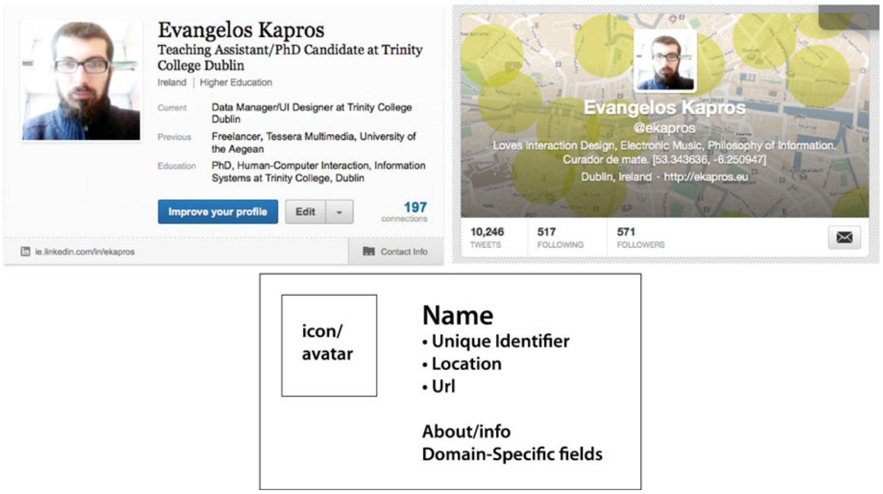
The intention in using soft schemas is to separate conceptual structure from application structure, so that change to the former does not necessitate change to the latter. But another, perhaps more far-reaching implication of this way of designing software is that an AIS could conceivably operate in many application domains, if supplied with appropriate soft schemas. Fewer applications would be required, because a single AIS could fulfill the function of many distinct (domain-specific) applications that must today be constructed separately, by hand using conventional software design practices.

3.3.2 Archetypal Categories and Differential Design

The AIS provides domain-specific behavior by responding to the currently-active soft schema. Each concept (entity type) in the soft schema represents something that data can be stored about. The AIS provides CRUD (create, read, update, delete) functionality in respect of every concept in the schema. Design heuristics are applied automatically to produce a "reasonably usable" interface directly from the conceptual model. This principle has been applied and tested in a number of web and client-server application environments (McGinnes 2005). Dialog design takes into account general rules of interaction and layout, as well as responding specifically to the data types used for attributes in the soft schema, the relationships between concepts, and so on.

However, for an AIS to offer true domain-specific functionality, it is insufficient to respond only to the conceptual model, because this provides a one-size-fits-all user interface style for every concept in the model. The AIS must instead offer a suitable interface style *for each* concept. Being able to do this depends on knowledge which is not normally present in conceptual models. For example, an application that stores data about geographical locations such as cities might offer an interface based on maps. Data about activities such as appointments might be represented using a calendar or timeline. Other interface styles are appropriate for other types of data. Normally, a software designer can choose appropriate interface styles using their own background knowledge about the concepts included in the conceptual model. The user interface designer recognizes what each concept signifies, and selects a suitable way of representing the concept and interacting with it (Liebenau and Backhouse 1990).

We therefore sought to embed this kind of general knowledge into soft schemas, so that it could be used automatically by an AIS to render more domain-specific interfaces and behavior. It is achieved by linking each concept in the soft schema with a particular *archetypal category* (major cognitive semantic category; Moore and Price 1999; Markman and Wisniewski 1997; Caramazza et al. 2003). The prototype AIS uses nine archetypal categories: people, organizations, places, documents, activities, physical objects, conceptual objects, systems and categories (McGinnes


Evangelos Kapros
Teaching Assistant/PhD Candidate at Trinity College Dublin
Ireland | Higher Education
Current Data Manager/UI Designer at Trinity College Dublin
Previous Freelancer, Tessera Multimedia, University of the Aegean
Education PhD, Human-Computer Interaction, Information Systems at Trinity College, Dublin
Improve your profile Edit 197 connections
ie.linkedin.com/in/ekapros Contact Info

Evangelos Kapros
@ekapros
Loves Interaction Design, Electronic Music, Philosophy of Information. Curador de mate. [53.343636, -6.250947]
Dublin, Ireland · http://ekapros.eu
10,246 TWEETS 517 FOLLOWING 571 FOLLOWERS

Fig. 3.1 Standard user profile design: the *upper images* represent domain-specific implementations. The generic wireframe below can load dynamically any domain-specific information at run-time. Changing its layout could result in any of the upper profile UI components

2005). Using archetypal categories allows the AIS to offer a category-specific interface style in respect of each concept in the soft schema. We refer to this process as *differential design*; it is intended to mirror the use of general knowledge by software designers (some related work exists in McGinnes 2005). An example is given in Fig. 3.1: any concept that belongs to the category people could use a standard design defined by a "user profile" visual component. This component could apply general knowledge, such as the fact that people are often identified by a name and an image, or that people usually reside at a location. This information can be required by the data structure, but everything else can be loaded dynamically in the interface and changes to the concept's definition will not break the interface.

Incidentally, the use of archetypal categories also presents advantages during modeling; for example, it allows aspects of models to be predicted, helping to speed up modeling and reduce error (McGinnes 2000).

3.3.3 Neurology and Cognitive Semantics

How much can we take these archetypal categories for granted? For many years a belief was prevalent that specific brain areas facilitate domain specific knowledge; this belief is referred to as *localizationism*. This idea has been challenged since 1891 (Freud 1953 (1891)). However, instances of damage to specific brain areas have been shown to affect unique knowledge domain. For example, some subjects have deficits in specific brain regions that prevented them from recognizing people (*prosopagnosia*) (Caramazza et al. 2003). Similar results have been proposed after

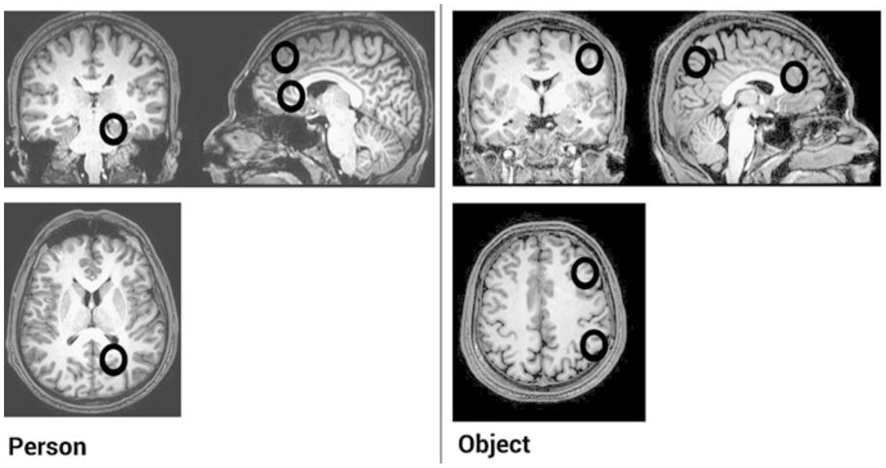


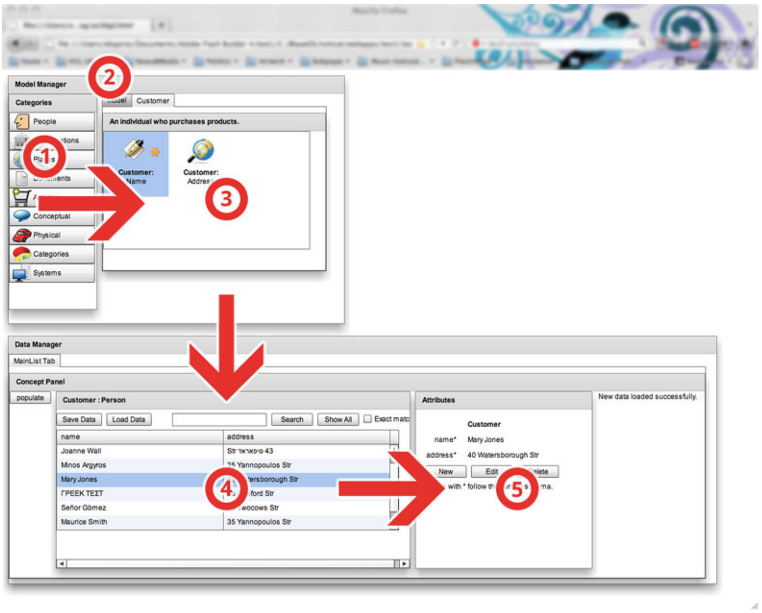
Fig. 3.2 fMRI showing approximate indicative positions of activation during Person and Object trials. Composed according to data found in Mason et al. (2004), Mitchell et al. (2002), Tyler and Moss (2001)

fMRI studies, where people, objects, and activities usually trigger signals in separate brain areas (Caramazza et al. 2003; Mason et al. 2004; Mitchell et al. 2002). Evolutionary theory has suggested that pressure from the environment resulted in dedicated neural mechanisms for each domain of knowledge, effectively creating categories that are in some sense "hard-wired" and therefore *archetypal* (Caramazza et al. 2003).

Localizationism has been challenged recently, drawing from cases where subjects have recovered from deficits of the aforementioned types. A known example of regenerated brain functionality (neuroplasticity) is the ability of blind people to substitute their visual cortex functionality with haptic input: brain areas that were formerly dedicated to one function switch to another, so that blind people can "see" what they touch (Pascual-Leone et al. 1999). However, research shows that archetypal categories still emerge, but this time in a distributed neural system rather than in brain areas, and that differences in the content of concepts drive the evolutionary categorization of cognitive semantics (Tyler and Moss 2001). There is no conflict between the fMRI results of Tyler and Moss (2001) and Mason et al. (2004), Mitchell et al. (2002) (also see Fig. 3.2).

Moreover, research has shown that cognitive semantics are formed in a *middle-out* way, in contrast with a bottom-up or a top-down one. That is, humans categorize entities using *basic level* categories first, and then generalize into more abstract entities or specialize into more concrete ones (Markman and Wisniewski 1997; Klibanoff and Waxman 2003). In simple terms, one would first recognize *a person* and then specialize it to, e.g., the particular individual Joanne Wall, or generalize it to, e.g., an abstract concept such as "animate entity".

In conclusion, given the slow pace of human evolution, we can assume it is safe to use basic level cognitive semantic categories in the construction of soft schemas.



name	address
Joanne Wall	Str סיפאראז 43
Minos Argyros	35 Yannopoulos Str
Mary Jones	ersborough Str
ΓPEEK TEΣT	ford Str
Señor Gómez	wocows Str
Maurice Smith	35 Yannopoulos Str

New data loaded successfully.

Customer
name* Mary Jones
address* 40 Watersborough Str

Fig. 3.3 An interaction map of the prototype AIS. See Table 3.1 for explanation of the layout and interaction

3.4 How the Prototype AIS Works

In this section we describe the visual and interaction design of the prototype AIS and present a technical explanation of how it deals with soft schemas and data. The present prototype implements soft schemas and archetypal categories with real-time user interface generation. Differential design (Sect. 3.3.2) and end-user modeling have yet to be implemented.

3.4.1 Visual and Interaction Design

The prototype's layout, navigation, and interaction have been designed with end users in mind, particularly given that the user interface evolves over time (O'Murchú 2009). There are two main panels, aligned vertically: the Model Manager and the Data Manager (see Fig. 3.3). The Model Manager consists of a vertical button bar and a tab bar. The buttons represent the nine archetypal categories. The tab bar allows access to panels showing the soft schema and its contents.

The first panel (shown by default) offers a top-level view of the active schema. It contains tiled icons, each denoting a particular concept in the schema. Labels help to disambiguate the meaning of icons (Evamy 2003; Whitehouse 1999) (for brevity, the term "icon" is used from this point to mean a labelled icon). The remaining

Table 3.1 Functionality of the various AIS layout elements

Element	Functionality
1. Categories button bar	Each button represents one of the archetypal categories. Clicking the button differentiates the concepts shown in the concept panel in that only concepts of the relevant category are highlighted.
2. Model tab bar	Allows the user to navigate through tabs containing the soft schema and its individual concepts.
3. Concept panel	Displays icons which represent concepts and attributes. Clicking an icon displays the tab panel and populates the datagrid for that concept.
4. Data management panel	Allows the user to load and save data, perform search/filter operations and manipulate data displayed in a dynamic grid. Clicking each row makes relevant information appear in the attributes panel.
5. Attributes panel	Offers basic data manipulation functionality; allows the user to enter, view and edit attribute values for particular concept instances and to delete concept instances. When one of the buttons is clicked a modal dialog appears, allowing the user to perform the selected function.

sub-panels represent individual concepts in the soft schema, each with tiled icons representing attributes or related concepts. For clarity, attributes have two labels: the first (in boldface) is the parent concept and the second is the name of the attribute. This way of presenting conceptual models, using icons and windows rather than boxes and lines, has been shown to substantially improve model understandability, particularly for non-experts (McGinnes and Amos 2001).

The Data Manager contains a data management panel and an attributes panel. The data management panel includes three sets of elements. Two buttons allow loading and saving of data, a set of elements facilitate searching and filtering, and a grid displays data stored by the AIS. The grid dynamically loads columns for the currently-selected concept's attributes and rows for its instances.

A text field notifies the user on the success of their actions including loading and saving data and data manipulation functions. To assist end users, action invitations are also used throughout. Hover invitations are activated for the data management panel, tabs, load/save buttons and concept icons. A cursor invitation is activated in the search input field, and a tool-tip invitation displays information about each archetypal category.

3.4.2 Handling Schemas and Data

The prototype AIS reads two types of XML file: schema files and data files. Each schema file contains a soft schema. Each data file stores data consisting of a number of concept instances. Each concept instance contains data values with structure that reflects the soft schema that the instance was created with.

Example 3.1 Schema file section describing the concept *Customer*:

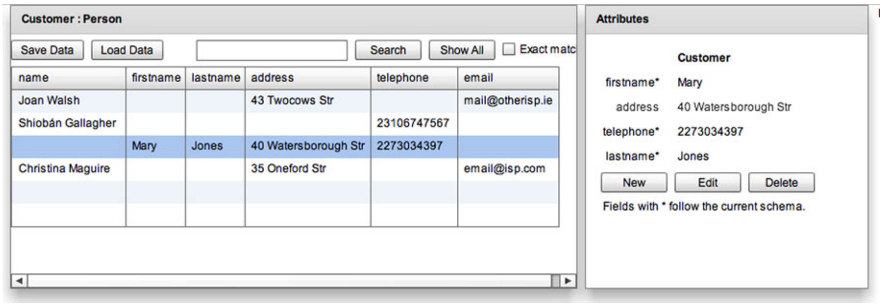
```
<concept>
    <conceptName>Customer</conceptName>
    <category>People</category>
    <attributes>
        <attribute id="1">name</attribute>
        <attribute id="2">id</attribute>
        <attribute id="3">address</attribute>
    </attributes>
</concept>
```

Once a schema file has been loaded, the AIS will enforce it for any new data instances that are entered. Data instances already stored may be retrieved and viewed, but will retain their original structure. Should the schema be altered (by loading a new schema or editing the active schema), the AIS will enforce the altered schema for any data that are subsequently entered but already-stored instances will not be affected.

Example 3.2 Data file section containing data for two customers previously entered using different soft schemas:

```
<customer>
    <name>Joanne Wall</name>
    <id>2012</id>
    <address>43 Tows Str</address>
</customer>
<customer>
    <firstname>Maurice</firstname>
    <lastname>Smith</lastname>
    <id>2002</id>
    <address>3 Yannou Street</address>
    <phone>2273034397</phone>
</customer>
```

The current schema file is not used for data retrieval and display, since any retrieved data may conform to a variety of soft schemas. Instead, the AIS interprets the data structure of each data instance, and then does its best to display the data instances together coherently, regardless of which soft schema each instance conforms to. For example, where different customers have different sets of attributes, as in the example above, the superset of the attributes is used to make up the list of columns in the data grid. Assuming that initially a concept Σ has attributes $A = \{a, b, c\}$ and later is modified to have attributes $B = \{x, y, z\}$, then the end user will be able to read instances of Σ with attributes $A \cup B$, add a new instance of Σ with attributes B, or delete an instance of Σ regardless of what attributes it has, subject



name	firstname	lastname	address	telephone	email
Joan Walsh			43 Twocows Str		mail@otherisp.ie
Shiobán Gallagher				23106747567	
	Mary	Jones	40 Watersborough Str	2273034397	
Christina Maguire			35 Oneford Str		email@isp.com

Customer

firstname*	Mary
address	40 Watersborough Str
telephone*	2273034397
lastname*	Jones

Fields with * follow the current schema.

Fig. 3.4 The data grid and the attributes panel after loading new data. Both automatically generated the columns and the text fields, thus adapting to the new data

to referential integrity constraints. Figure 3.4 illustrates the effect when a schema is changed and new data added. The columns for newly-entered instances differ from those for existing instances, yet all are displayed.

3.4.3 Applications in Reverse Engineering of Existing Data Structures

We note that it is a conceptually-simple operation to reconstruct the conceptual model underlying any database structure or XML data. Most of the semantics necessary to recreate the conceptual model implemented by a software application are implicit in, and capable of being determined by examination of, its data storage structures. This makes it possible, in theory, to use an AIS with any arbitrary dataset, regardless of whether its corresponding soft schema exists. The required soft schema can simply be reconstructed by examining the data, and this process can be automated.

The ability to reconstruct soft schemas automatically has been demonstrated in two AIS implementations to date. In the first, the AIS was capable of reading a database structure and thereby producing a corresponding soft schema. The resulting soft schema could be used to store and manipulate data with equivalent structure to that stored in the source database. But, unlike the source database, the AIS would permit the schema subsequently to be modified at will. This proved useful as a first stage in the reengineering of legacy database applications. The data structure from an existing application could be turned into a soft schema, which could then evolve relatively easily through a prototyping process to arrive at an improved structure matching client user requirements.

The second implementation is capable of reading an XML data file and reconstructing its corresponding soft schema. If the XML data file is an AIS data file, then the resulting soft schema can immediately be used to add to, and modify, the data in the file. This is useful, for example, if the soft schema for a particular data file has been lost for some reason. It is also useful where a schema has undergone substantial evolution, so that the data in the data file corresponds to multiple soft schema versions. In this case the reconstructed soft schema represents the superset of all soft schemas implied by the data. Being able to reconstruct a superset schema is useful where it is helpful to know the range of possible conceptual structures which could be considered valid.

In reconstructing a soft schema, not all elements can always be deduced. For instance, relationship cardinalities are often incompletely specified. The data may make it clear that each customer can have multiple orders, but not specify whether a customer *must* have any orders. Also it is rare, unless the data file is an AIS data file, for the data to be tagged with archetypal categories, icons, or other semantic information. Suitable categories and images can to some extent be automatically suggested by recognizing common terms. For example, for an item of data with XML tag `<customer>` it would be appropriate to suggest categories *person* or *organization*. Similarly for tag `<order>` it would be relevant to suggest category *activity*. Default images can be used according to the categories suggested. However, this process of deducing categories and images is inherently hit-and-miss, and so any suggested categories and images require review and possible modification by the user.

3.5 Discussion and Future Work

At present the prototype successfully reads schema and data files and generates suitable user interfaces, allowing basic CRUD (create, read, update, delete) functions to be performed on the data. This implementation demonstrates the feasibility of separating conceptual models from application structures, and of automatically generating user interfaces in real time from soft schemas. The next stage of our project will experimentally assess the usability of the prototype; however, related research has shown that relatively sophisticated and usable interfaces can be created this way for a variety of implementation platforms (McGinnes 2005).

Changing the schema presents no problem to the application, which continues to work effectively. Since previously-entered data can still be viewed, the user can upgrade the data to match the current schema at his or her leisure, or choose not to. We envisage that tools can be provided to assist the user in this process, identifying data which could be upgraded and automatically performing the upgrade where this is feasible. We anticipate benefits to the end user from being able to continue to use previously-entered data despite schema changes. For example, it will allow applications to grow and evolve as end user understanding improves through use. However, it also opens the possibility that data will become chaotic and unusable, particularly if many schema changes are made but data instances corresponding to earlier schema versions are not upgraded to match the new schema structure. Usability testing will reveal whether this ability to change the schema without affecting existing data is helpful for end users, or merely results in chaotic datasets which are difficult to understand and use.

At present the AIS supports only simple soft schemas, as support for relationships between concepts has yet to be implemented. We intend to add support for relationships; this will require implementation of more sophisticated user interface heuristics. Again, prior work has demonstrated that automated design can produce usable interfaces for schemas with complex relationships between concepts (McGinnes 2005). The challenge in this instance is to make the automated design occur purely at runtime rather than a mixture of design time and runtime.

In addition, functions will be added to allow the end user to visually manage soft schemas. End-user modeling using a similar schema representation has been tested in previous research (McGinnes 2000) but usability testing will help assess how easy it is for end users to do their own modeling in the context of the prototype AIS. We hypothesize that the ability to enter and retrieve data immediately upon schema change, without the need for data transformation and reloading, will facilitate understanding and learning. We also plan to implement better support for data types, with differential design, that is the dynamic selection of user interface style depending on archetypal category. For example, map views could be provided for places and calendar views for activities. It is hoped that this will improve the usability of the AIS, making it look and feel more like a hand-coded application. Again, usability testing will help evaluate and refine this feature.

Finally, the semantic categories are intended to serve as an examination ground for a potential semantic standard. This would make software more interoperable and consistent. Despite using XML at the moment, moving to OWL/RDF is an option. In this way standardization would be enforced; in any case, this option needs to be examined after adding support for relationships.

3.6 Conclusion

This chapter has presented a prototype user interface for an adaptive information system. The system handles various conceptual structures at runtime, treating these structures as data (*soft schemas*). It allows the user to handle (create, read, delete) data, as well as update soft schemas or data.

The intention is to evaluate the usability of a system with separate data and conceptual structures. Our hope is that software designed in this way could be more flexible for end users; one piece of software could have more uses than the domain-specific applications built according to current practices.

References

Alani, H., Kalfoglou, Y., O'Hara, K., & Shadbolt, N. (2005). Towards a killer app for the semantic web. In *The semantic Web–ISWC 2005* (pp. 829–843).

Alani, H., Hall, W., O'Hara, K., Shadbolt, N., Szomszor, M., & Chandler, P. (2008). Building a pragmatic semantic web. *IEEE Intelligent Systems*, *23*(3), 61–68.

Bakke, E., Karger, D., & Miller, R. (2011). A spreadsheet-based user interface for managing plural relationships in structured data. In *Proceedings of the SIGCHI conference on human factors in computing systems*, CHI '11 (pp. 2541–2550). New York: ACM.

Berners-Lee, T., Hall, W., Hendler, J., O'Hara, K., Shadbolt, N., Weitzner, D. J., et al. (2006). A framework for web science. *Foundations and Trends in Web Science*, *1*(1), 1–130.

Berners-Lee, T., Hollenbach, J., Lu, K., Presbrey, J., Pru d'ommeaux, E., et al. (2007). *Tabulator redux: writing into the semantic web*.

Caramazza, A., Mahon, B. Z., et al. (2003). The organization of conceptual knowledge: the evidence from category-specific semantic deficits. *Trends in Cognitive Sciences*, *7*(8), 354–361.

Chan, Y. E., & Storey, V. C. (1996). The use of spreadsheets in organizations: determinants and consequences. *Information & Management*, *31*(3), 119–134.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, *13*(6), 377–387.

Curino, C. A., Tanca, L., Moon, H. J., & Zaniolo, C. (2008). Schema evolution in Wikipedia: toward a web information system benchmark. In *International conference on enterprise information systems (ICEIS)*. Citeseer.

Ertl, D., Kaindl, H., Arnautovic, E., Falb, J., & Popp, R. (2011). Generating high-level interaction models out of ontologies. In *IUI SEMAIS* (Vol. 11, pp. 7–11).

Evamy, M. (2003). *World without words*. New York: Laurence King.

Fein, E., Razinkov, N., Shachor, S., Mazzoleni, P., Goh, S., Goodwin, R., et al. (2011). Using MATCON to generate case tools that guide deployment of pre-packaged applications. In *2011 33rd international conference on software engineering (ICSE)* (pp. 1016–1018). New York: IEEE Press.

Fonseca, F. T., & Martin, J. E. (2004). Toward an alternative notion of information systems ontologies: information engineering as a hermeneutic enterprise. *Journal of the American Society for Information Science and Technology*, *56*(1), 46–57.

Freud, S. (1953 (1891)). *On aphasia; a critical study*. Madison: International Universities Press.

Hartung, M., Terwilliger, J. F., & Rahm, E. (2011). Recent advances in schema and ontology evolution. In *Schema matching and mapping* (pp. 149–190).

Hick, J.-M., & Hainaut, J.-L. (2006). Database application evolution: a transformational approach. *Data & Knowledge Engineering*, *59*(3), 534–558.

Kennedy, O., Ahmad, Y., & Koch, C. (2011). DBToaster: agile views for a dynamic data management system. In *Proc. of the fifth biennial conference on innovative data systems research (CIDR 2011)* (pp. 284–295).

Klibanoff, R. S., & Waxman, S. R. (2003). Basic level object categories support the acquisition of novel adjectives: evidence from preschool-aged children. *Child Development*, *71*(3), 649–659.

Kohlhase, A., & Kohlhase, M. K. (2011). Spreadsheets with a semantic layer. *Electronic Communications of the EASST*, *10*, 1–18.

LENA—a Fresnel LEns based RDF/Linked Data NAvigator with SPARQL selector support (n.d.).

Liebenau, J., & Backhouse, J. (1990). *Understanding information: an introduction*. Basingstoke: Palgrave Macmillan.

Markman, A. B., & Wisniewski, E. J. (1997). Similar and different: the differentiation of basic-level categories. *Journal of Experimental Psychology. Learning, Memory, and Cognition*, *23*(1), 54.

Mason, M. F., Banfield, J. F., & Macrae, C. N. (2004). Thinking about actions: the neural substrates of person knowledge. *Cerebral Cortex*, *14*(2), 209–214.

McGinnes, S. (2000). *Conceptual modelling: a psychological perspective*. Doctoral dissertation, London School of Economics and Political Science (University of London).

McGinnes, S. (2005). *Systems and methods for software based on business concepts*.

McGinnes, S., & Amos, J. (2001). Accelerated business concept modeling: combining user interface design with object modeling. In *Object modeling and user interface design* (pp. 3–36). Reading: Addison-Wesley.

McGuinness, D. L., Fikes, R., Rice, J., & Wilder, S. (2000). An environment for merging and testing large ontologies. In *Principles of knowledge representation and reasoning-international conference* (pp. 483–493). San Mateo: Morgan Kaufmann.

Mitchell, J. P., Heatherton, T. F., & Macrae, C. N. (2002). Distinct neural systems subserve person and object knowledge. *Proceedings of the National Academy of Sciences*, *99*(23), 15238–15243.

Moore, C. J., & Price, C. J. (1999). A functional neuroimaging study of the variables that generate category-specific object processing differences. *Brain*, *122*(5), 943–962.

Noy, N. F., & Musen, M. A. (2000). Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the 17th national conference on artificial intelligence (AAAI-00)*. Available as SMI technical report SMI-2000-0831.

O'Murchú, N. (2009). Understanding adaptive design and user experience. In *Irish human computer interaction (I-HCI) conference 2009*.

Pascual-Leone, A., Hamilton, R., Tormos, J., Keenan, J., & Catala, M. (1999). Neuroplasticity in the adjustment to blindness. In *Neural plasticity: building a bridge from the laboratory to the clinic* (pp. 94–108). Berlin: Springer.

Raden, N. (2005). Shedding light on shadow it: is Excel running your business? *DSSResources.com*, *26*.

Roddick, J. F., Al-Jadir, L., Bertossi, L., Dumas, M., Gregersen, H., Hornsby, K., et al. (2000). Evolution and change in data management—issues and directions. *ACM SIGMOD Record*, *29*(1), 21–25.

Shirky, C. (2003). The semantic web, syllogism and worldview. In *Networks, economics, and culture*.

Sun, Y., Gray, J., & White, J. (2011). Mt-scribe: an end-user approach to automate software model evolution. In *2011 33rd international conference on software engineering (ICSE)* (pp. 980–982). New York: IEEE Press.

Tyler, L. K., & Moss, H. E. (2001). Towards a distributed account of conceptual knowledge. *Trends in Cognitive Sciences*, *5*(6), 244–252.

Wach, E. P. (2011). Automated ontology evolution as a basis for adaptive interactive systems. In *IUI SEMAIS 11* (pp. 467–468).

Whitehouse, R. (1999). The uniqueness of individual perception. In R. Jacobson (Ed.), *Information design* (pp. 103–129). Cambridge: MIT Press.

Zhao, C.-c., Zhao, L.-y., & Wang, H.-l. (2010). A spreadsheet system based on data semantic object. In *2010 the 2nd IEEE international conference on information management and engineering (ICIME)* (pp. 407–411).