Conceptual interfaces

Ján Šefránek
Institute of Informatics, Comenius University
Bratislava, Slovakia
e-mail: sefranek@fmph.uniba.sk

October 19, 2000

Abstract

This paper is devoted to an aspect of man-machine interaction (MMI), called *conceptual interface* in the sequel. By a conceptual interface we understand a set of concepts used in the communication (between man and machine) together with the relationships among the concepts.

We discuss the basic features of conceptual interfaces. The discussion is based on some examples. The impact of a conceptual interface on context-dependent and dynamic aspects of man-machine interaction is emphasized.

Special attention is devoted to a kind of conceptual interface – to Structured Maps (SM). A structured map defines a database schema and its instance over an information universe. Therefore, an additional structure is superimposed over a world (of documents) and the structure extends the possibilities of querying and searching.

We generalize the concept of SM. We will consider a Society of Structured Maps (SSM, a set of alternative SMs, they represent a set of alternative views on the information universe). Additionally, a set of transformations between the Structured Maps is considered. The transformations provide a dynamic view of the information universe. Finally, it is outlined how a society of SMs can be formalized in terms of Dynamic Kripke Structures.

1 Introduction

Each dialogue is based on a conceptual background. In this paper we reflect the obvious fact that an interaction between human and computer relies on some conceptual background, too. The basic starting point of this paper is trivial: for each program and each user (of the program) there is a set of concepts used in man-machine interaction. The concepts range from "click here" to – for example – the natural join of two relations.

First, we explain what we mean by a *conceptual interface*. The interaction between man and machine has to be described from a variety of viewpoints. In this paper we restrict ourselves to a viewpoint that does not usually attract a suitable attention. We devote our attention to the basic conceptual background underlying the interaction (between man and machine). In order to express this conceptual background we shall use a notion of the *conceptual interface.*

Definition 1 *By a* conceptual interface *we understand a set of concepts used in the communication between man and machine together with a set of interrelations between the concepts.*

Second, our goal is to discuss a crucial importance of the conceptual interface. The trivial fact, the existence of a conceptual interface between man and machine yields some interesting consequences. Our claim is that features, qualities of MMI are dependent on features of the conceptual interface.

The quality of MMI can be assessed by such features as flexibility, adaptability to changes, to context. Flexibility and adaptability are supported by an appropriate conceptual interface. For example, a suitable conceptual interface is of decisive value for a flexible and intelligent access to relevant parts of huge volumes of information (stored in data bases, document bases, World Wide Web).

A proper level of abstraction of the conceptual interface is important. The appropriate abstraction supports an embedding of the conceptual interface constructs into various contexts and enables a more flexible, dynamic interaction.

The problem of context-dependence and dynamical change of MMI is of central importance, and until now only partly solved. The ability to adapt a running dialogue to the current context, is inherent in the dialogue between humans. To improve the dialogue between man and machine, it is essential to capture at least part of this ability in the machine. The present paper is an attempt to analyze the problem and to outline a first step to a formalization.

The paper is organized as follows. First, we motivate our approach by an analysis of some characteristic types of conceptual interfaces. The basic features and merits of conceptual interfaces are discussed on the example of relational data model and relational languages (and some of their generalizations). It should be emphasized that the topic of this article are not databases – the topic is the conceptual interface and databases provide a good example of an appropriate conceptual interface.

A special attention is devoted to Structured maps (SM), see [6]. An SM defines a database schema and its instance over an information universe. An additional structure in terms of SM is superimposed over a world (of documents) and the structure extends the possibilities of querying and searching (therefore, the possibilities of MMI).

We generalize the concept of SM – a set of alternative structures can be superimposed over the information universe and some transformations between the structures provide a kind of dynamics. Finally, it is outlined how a society of SMs can be formalized in terms of Dynamic Kripke Structures.

Each of the sections will be concluded by two statements, indicated by A and B. The A statement summarizes the basic properties of the conceptual interface, and the way it is specified. The B statement characterizes the dynamic and context-dependent aspects of the conceptual interface type under discussion. A list of abbreviations is given at the end of the paper.

We do not assume that a reader is familiar with relational databases, data mining, and semi-structured databases . Therefore, we need to explain some basic facts from the above mentioned domains. Nevertheless, the main accent is on the conceptual interface of databases etc., and on dynamic, context-dependent features of the conceptual interface. The reader with a clear idea of the conceptual interface in relational databases etc. can skip to the section 3.

2 Examples

In this section we illustrate and justify (in a sense) the restriction of our attention to the conceptual interfaces. We present some examples of appropriate conceptual interfaces.

Let us recall that by a *conceptual interface* we understand a set of concepts used in the communication between man and machine together with a set of relationships among the concepts.

A level of abstraction of the used conceptual apparatus is of crucial importance. The level should be appropriate for human thinking and also for a transformation of the conceptual structures to a representation on the machine level.

2.1 Relational database languages

An excellent example of a suitable conceptual interface between man and machine are relational database languages.

Let us compare properties of the conceptual interfaces underpinning the traditional DBMSs and the relational DBMS. The fruitful properties of the relational data model (and its languages) [4] are clearly visible in contrast to the database management systems (DBMS) of sixties.

Data models of the late sixties provided a view of the data very close to the physical representation of data in storage, see [5, 11]. As a consequence, they did not develop sufficiently strong language tools for casual end users and also for programmers, (i.e. tools for MMI). According to Codd, the traditional systems "burdened application programmers with numerous *concepts* that were irrelevant to their data retrieval and manipulation tasks, forcing them to *think* and code at a needlessly low level of structural detail . . . The needs of end users for direct *interaction* with databases, particularly interaction of an unanticipated nature, were inadequately recognized – a query capability was assumed to be something one could add on to a DBMS at some later time . . . There was no sharp distinction between the programmers (logical) view of the data and the (physical) representation of data . . . In addition, it was not possible to make slight changes in the layout in storage without simultaneously having to *revise* all programs that relied on the previous structure" [5].

On the other hand, among the goals of Codd's research work that resulted in the relational model were

- to provide a sharp and clear boundary between the logical and physical aspects of database management (the *data independence* objective),
- to make the model structurally simple (the *communicability* objective),
- to introduce high level language concepts to enable users to express operations upon large chunks of information at a time (the *set-processing* objective).

Codd has introduced an abstract (logical and algebraical) view of data and of operations on data. Relational calculus and relational algebra provide a set of concepts useful for defining databases and for querying databases (some examples: relation, attribute, functional dependence, selection, projection, join, union, etc.).

What are the crucial properties of the relational data model and the relational database languages? They provide a suitable *conceptual representation* of databases (of data and of operations on data):

- the representation is indeed a proper abstraction of the physical organization of the data as well as an abstraction which supports *thinking, expressing requirements*, and *communicating ideas*,
- all these essential properties (support of thinking, expressing, and of communication) are mediated through the relational database languages; they are easy to learn, they have a clear semantics, and there are well established methods of their efficient implementation,
- the abstraction enables the separation of implementation problems from the specification and design problems.

Other important properties are consequences of the essential features discussed above. We can mention a support of *unanticipated ad-hoc queries*, *declarativeness*, a data *independence*.

The view construct, a closure property and a possibility of embedding queries into general purpose programming languages deserve in the context of this article a special attention. Let us discuss them:

Usually, only a part of data is interesting from a viewpoint of a group of users (or from a viewpoint of an application program). This part of data is called a *view*.

The view construct defines what is relevant (for a group of users) in a database. Of course, the relevancy is dependent on the context – the context of a group of users or the context of an application program. If the context (user or program, her/his/its roles and tasks) is changed, the view has to be changed, too.

The *closure property* is a characteristic feature of relational languages: a query is defined on some relations (the relations provide the input for the query), but the result of the query is a relation again, and it can provide an input to another query etc. etc.

The expressions of the relational language can be *embedded* into a general-purpose *host language* (as Cobol or C^{++}, for example), through the *application programmer interface* (API).

The closure property and the capability of embedding the expressions of the relational language into a general-purpose host language provide – in a sense – an ability of a dynamic embedding of a query into a new context (the context of a program or the context of another query).

Finally, one cannot use a relational database language without a knowledge of a database conceptual schema. This implies that the relational interface between human and a database is completely determined by the given database schema and by the relational language. This type of conceptual interface can be characterized as the interface requiring complete knowledge of the interaction domain.

Observation 1 *We can summarize a conceptual interface to a database (and its dynamic and context-dependent aspects) as follows:*

A *A conceptual interface to a relational database consists of*

- *the constructs of a relational database language,*
- *a database schema, defined in the language (in its data definition part),*
- *the primitives provided by API.*

B *Context-dependent and dynamic features of conceptual interfaces to relational databases are based on the view construct, on the closure property, and on the application programmer interfaces.*

Remark: Let us discuss the observation 1. The constructs of a relational database language are conceptual tools – they enable to view a database in terms of relations, selections, projections, joins etc. Similarly, a database scheme, defined in the language, supports a view of data in terms such as "employee, manager, department, address, supplier, state, city", etc. etc. (instead of fields of records or sequences of bytes). The primitives provided by API enable to place statements of a more abstract conceptual structure – of a database relational language – into a code written in less abstract setting, for example in Cobol.

However, at the beginning of seventies it was not yet obvious that the appearance of the relational model on the database scene changed the situation in a radical way. For example, Jeffrey Ullman, the author of one of the most important monographs devoted to the relational database theory [11], wrote: "Codd's paper was so profound, that it took five years before its implications could begin to be understood in either theoretical or the practical areas. I remember reading the paper when it first came out and sneering at its foolish attempt to throw some mathematical terminology at what it claimed was a real problem. Well, we live and learn, I guess."[12] The slogan "it's theoretical, it cannot be practical" [5] represented a common attitude towards the relational data model in the seventies.

Now we present an example illustrating the crucial role of a suitable conceptual interface. One could expect that a pictorial language essentially improves the qualities of an interface to databases.

A pictorial interface over hierarchical and network databases is described in [3]. But the interface did not influence their capabilities in a decisive way. Hierarchical and network data models lost their position (because of the properties of their conceptual interfaces) and the relational data model became the dominant data model of nineties. For example, a relational interface to the version 12 of the IDMS, the best known and successful network database, was developed at the beginning of nineties, in order to increase its competitive power.

It means that a clear and simple conceptual interface between man and machine is of decisive importance (at least for database applications), the strength and usefulness of a data model (and its language) cannot be essentially improved by a pictorial language. The pictorial interface can be helpful, it is comfortable, but it does not improve the expressiveness of a language used in a man-machine communication. It can be said that a pictorial interface to a database does not change the possibilities and facilities offered by a conceptual interface in a substantial way.

In the next two subsections we discuss some generalizations of relational languages. They use more general and more complicated schemata, more general queries, more general views of data and they relax the knowledge requirements of the interaction. We are interested in consequences of these generalizations for the idea of *conceptual interface*.

2.2 Data mining

First, we generalize the query concept.

A relational query takes a set of relations (as an input) and produces a relation (over some attributes of the given database schema). An important generalization of this query concept is used in data mining (or knowledge discovery in databases, KDD).

Intuitively, a relation represents a set of facts. But a deeper knowledge is contained implicitly in the set of facts. We can discover regularities, associations, and frequently occurring patterns in the facts. It is natural to ask questions about such regularities, associations, rules (in more technical terms: about KDD-objects). KDD-objects are: rules, classifiers, and clusters. It means, KDD is interested in more general objects than relations. Each KDD-object can be a response to a query.

A distinction between first-generation and second-generation KDD is important from the viewpoint of MMI. The current situation in KDD is according to [7] very similar to the situation of DBMSs in the sixties, when each application had to be built from scratch, without the benefit of dedicated database primitives provided later by the relational language SQL and relational database API's. The first-generation programs for discovering knowledge in databases are dedicated to isolated applications, they cannot be used in a variety of contexts – in a variety of applications, in a variety of computations.

The crucial features of a second-generation KDD query language are:

- the closure principle: a result of a KDD query may be an argument of another KDD query,

- KDD queries can be embedded into a host programming environment to provide application programming interfaces for knowledge- and data-discovery applications, just as SQL queries can be embedded in host languages.

Therefore, the second generation KDD can be embedded into new contexts. Moreover, the interface of KDD is a more strong one (when compared with the DBMSs): while a query in a DBMS is mediated through a relational conceptual schema, in the case of the KDD the conceptual interface contains (besides the database schema) a knowledge about possible associations among data, about clustering, and classifying.

Embedding of KDD-queries through an application-programmer interface into a new context supports a full use of data analysis methods. KDD-objects generated by queries can be embedded in more complex applications that may involve spreadsheets, statistical packages, graphical tools etc.

The above-mentioned features are very similar to the relational database facilities (as discussed in the subsection 2.1). Now we will pass to some specific features of KDD.

The KDD process is inherently interactive and iterative [8]. According to Mannila, the KDD systems should be seen as an interactive tool, not as automatic-analysis systems. An iteration of the basic steps in the KDD process is necessary: The discovered patterns can show that some changes should be made to the data-set formation step. Post-processing of patterns can cause a modification of pattern types etc. If some pattern looks interesting, the user often wants to evaluate patterns that closely resemble that pattern.

Interactiveness and iterativeness are natural components of each intelligent dialogue which reacts to a given context and to its own dynamics. Both components assume an intervention of a user into the KDD process: the user selects the interesting rules from the huge amount of the generated rules. Discovering methods and tools for selecting, ordering, and grouping of rules are challenging research problems in the area. A kind of meta-predicates can be introduced in order to formalize the concept of interestingness.

Computational problems caused by huge volumes of data in databases can be solved by sampling. Small samples from the database can give good approximations to the association rules. Knowledge implicit in a selected sample provides an approximation of the knowledge implicit in the whole database. Of course, the results of sampling can be wrong. It is not excluded that results obtained in one sample (context) cannot be obtained in another. A process of revision should be a part of a KDD process based on sampling.

"The user can take advantage of the previous query, by slightly modifying it or by taking a rule which is a part of the answer and mining around it ... queries themselves may require the system to perform complex search tasks that may lead to *nondeterministic computations*. Thus, queries serve only as general and often loose specifications for the system to provide varying degrees of user control over the discovery process. (In this way even a fully automated discovery can be viewed as a query evaluation for a very lose and broadly specified query.)" [7]

Observation 2 *Let us summarize:*

A *KDD uses a more expressive conceptual interface and provides more powerful tools for querying than relational query languages. A conceptual interface to a KDD-system consists of*

- *a conceptual interface to a relational database,*
- *a conceptual apparatus used in the generalized querying process (specification of KDD-objects: classifiers, clusters, and rules expressing associations)*
- *the primitives provided by the API of KDD*

B *Context-dependent and dynamic features of KDD-conceptual interfaces are based on the closure property and on the application programmer interfaces. Moreover, they are supported by an interactive and iterative character of the KDD-process. A kind of dependency on the context is introduced into KDD by sampling and by an iterative experimentation of the user with the data stored in the machine.*

In the sequel we do not use the generalized concept of query. Therefore, we take no more account of the aspects of knowledge discovery, learning, and induction. Nevertheless, we should note that the above-mentioned aspects are of key importance for an intelligent dialogue. We intend discuss the problem in an abstract setting, in Dynamic Kripke Structures (DKS), see 5. DKS provide a semantics of hypothetical reasoning with such significant features as context-dependency and dynamics. Of course, hypothetic reasoning is a substantial aspect of knowledge discovery, learning, and induction.

2.3 Semi-structured databases

In this section we proceed to a further generalization of the conceptual interface to relational databases. We relax the complete knowledge (of the interaction domain) condition.

Semi-structured data are not completely described by a rigid structure. The structure may be irregular or incomplete. Only a part of the document can be described by the structure, but a large part of information cannot be described by the structure. Such data are for example in HTML, SGML, or BIBTEX format (see the list of abbreviations, if needed).

The structure can be represented by a structuring schema as in [1]. The schema defines the structured part of a database of semistructured objects (documents).

The query languages for semi-structured data, see [1, 2, 9], exploit a structure when it is present, but it does not require a uniform structure for meaningful answers. The languages contain:

- search based on the structure of the data,
- information retrieval facilities (searching a string, a pattern),
- browsing and navigation.

Querying data without exact knowledge of their structure is enabled by the information-retrieval facilities, by browsing and navigation capabilities. Moreover, the structure alone can be a subject of querying. For example paths and attributes are possible responses to some queries. Path and attribute variables of OQL-doc, see [1], enable to obtain information about a path in the hierarchical structure of a document and about an attribute (containing relevant information). If we obtain the information about the path or about the attribute, we can pose queries directly to the content of the node reachable on the path, or to the content of the attribute.

Control of the navigation for WWW-querying is incorporated into an SQL-like language, WebSQL, see [9]: only a limited neighborough (not too far from a specified source) can be specified for a task of navigation. Moreover, WebSQL takes advantage of multiple index servers without requiring users to know about them.

Observation 3 *Let us summarize:*

A *A conceptual interface to a semi-structured database consists of two parts:*

- *a part specified by a database structure (this part of the conceptual interface is similar to the conceptual interface to a relational database),*
- *an interface to the unstructured rest of the database established on concepts usually used in information retrieval, browsing, and navigation.*

B *In order to be complete, a context-dependency and dynamics in semi-structured databases can be of the same type as in the relational databases. Context-dependent features and a kind of a dynamics of the conceptual interface to the unstructured part of the database rely on browsing and information retrieval facilities. Browsing enables to react to the given context and to reflect the changes and states reached in the history of browsing. The techniques of information-retrieval enable stepwise refinement of a query, interactiveness, and iterativeness.*

In the next section we will discuss a way to define a (meta)-structure for data with no explicit structure. The way is called Structured Maps. The idea of structured maps is a promising one, it helps to overcome some bottlenecks of information retrieval-techniques and of browsing. In the case of information retrieval, the occurrence of a string or pattern is often not significant, it does not characterize the relevance of the text in a sufficient manner. In the case of browsing, the search is often "too blind" and the user often loses the orientation in the data.

3 Structured Maps

Sometimes we recognize which information is implicit in a database (in an information store) only if we superimpose a (meta-)structure over the database. Let "the database" be a set of clinical findings about a patient. If a physician superimposes a right medical meta-structure over the findings and accepts a diagnosis explaining the findings, he may discover the information implicit in the findings.

The unstructured part of a semi-structured database can be, in a sense, described by a new conceptual (meta-)structure superimposed over the data. The structure introduces a new information.

The idea is elaborated in a systematic way in Structured Maps of [6]. Structured Maps (SM) provide a kind of (meta-)structure that supports a better orientation in the data. An analogy between SM and a conventional map is emphasized in [6]. The framework is divided into three layers. At the bottom is an Information Universe (IU), a set of documents. The documents are assembled in general from many heterogeneous sources. There are two layers over the information universe. They provide a kind of an organization of the information in the IU. The top level contains a definition of an entity-relationship database schema. The middle level consists of an instance of the schema. The entities of the instance contain links to elements of the information universe.

Let us use the example from [6]: The database schema may contain attributes as *painter, painting*. On the instance level there are names of many painters and titles of many paintings (and information about the authors of paintings). Let us consider some facts recorded in the database instance, for example, Gentille Bellini (a painter) and "Miracles of the Relic of the Cross" (his painting). SM contains links from these two database facts to the Information Universe, i.e. to the documents about Gentille Bellini and to the documents about his painting "Miracles of the Relic of the Cross".

Therefore, a connection between the superimposed structure and the information universe is defined. Structured Maps are implemented by using relational-database technology.

We proceed to a characterization of:

- the content of an Information Universe,
- and to the selection of information capabilities.

There are no limits on the structure of documents in the (heterogeneous) IU. The documents in IU are collected from various sources. SM need not be complete nor exhaustive. SM contain only the information explicitly recorded by the user (or by the tools the user employed).

A selection of information is mediated by the (meta-)structure. The database schema (and the complete database machinery) is used for querying. Moreover, the schema serves as well as a basis for navigation. When a navigation process is running, the schema can be used also as a tool for orientation: in each step of browsing one can see the visited part of the schema.

Observation 4 *Let us summarize:*

A *the conceptual interface inherent in a structured map provides a meta-structure over an information universe; the conceptual interface exhibits more structure than is present directly in the information universe,*

B *the context of the meta-structure is of decisive importance for information search in the given set of documents.*

Of course, we can define more (meta-)structures over one IU. Possible transitions among Structured maps are not discussed in [6]. In the next section we discuss this type of transitions.

4 A Society of Structured Maps

The SM-approach permits a set of structures superimposed over the information universe. A natural generalization of this approach is based on a system of transitions among the (meta-)structures. Let us summarize basic components of Structured Maps (slightly generalizing):

- a schema (a view on the information universe – in a simple case a database schema, as in [6]; in a more general case it can be a set of predicates together with a set of postulates defining the arguments of the predicates and some relations between the predicates),
- an instance of the schema (a description of the information universe in terms of the schema),
- information universe.

Let us imagine that we are querying the information universe via a given SM. It is not excluded that the current results of querying are not relevant and interesting for us. It may be that the view defined by a given SM is not the proper one with respect to the interests and requirements of a user.

Example 1 *Let a unary predicate seminal_paper*(ID) [1] *be among the predicates in the conceptual schema of a given SM. But the query of the form "Give me all seminal papers!" returns too many not relevant responses.*

Let us assume we would like to transfer to a binary predicate seminal_paper$(ID, Topic)$ *and to introduce a new predicate topic*$(Topic)$.

If we specify a topic (or some topics) in a query, the response should be more relevant.

We can modify the conceptual schema of the given SM and implement the changes on the instance level. Simultaneously we should modify links to the elements of the information universe. But there are several reasons for a transfer to a new SM (instead of a modification of the given SM):

[1] The intended meaning of the unary predicate *seminal_paper* is that the paper ID is a seminal one. The intended meaning of the binary predicate *seminal_paper* is that the paper ID about a topic $Topic$ is a seminal one. The predicate *topic* enables to define a set of topics.

- sometimes (in some contexts) the original SM could be more useful than the new one,
- the original SM may be a "private property" (we imagine a world-wide community of SMs with a public access via Internet), [2]
- we don't want to require an apriori knowledge of an "SM-definition" (the set of defined predicates, their arguments, and their interrelationships); a huge conceptual schema (an expected result of a series of modifications) is not an appropriate one, a small schema is better, it should be graspable in one view.

Therefore, we will consider a society of SMs. The society consists of a set of schemata. For each schema there is an instance (or more instances!), connecting the schema with the information universe. Connections between the top levels of the structure and the information universe are like in the structured maps. Of course, the description of the information universe in terms of the given schema is not necessary complete (to the contrary, it usually is not complete, only a part of the items in the information universe is described in terms of the schema).

An essential aspect of this approach is a set of transformations from one schema to another. If we do not take account of the problems connected to a development and maintenance of a society of SMs, there is only one important problem: how to design and implement the transition from one SM to another.

If a given context is changed or if a change of the given context is required, a dynamic reaction is needed – a change of the view of data, a change of the superimposed schema (and its instance). There is a variety of possible changes.

If a "world of SMs" is given and known, there are two main problems:

- how to define and create a transformation,
- what transformation to be used in a situation.

A solution is dependent on the disposable information about SMs.

If only a list (of addresses) of SMs is known, we can use some blind transitions among SMs or some transitions based on the string- (or pattern-) retrieval capabilities.

If a kind of knowledge about SMs [3] is accessible a kind of informed transformations can be proposed.

The transformations can be called manually or automatically. If the user manually chooses a transformation (from a variety of transformations) she/he can browse blindly or

[2] The idea is not too unrealistic. There are private bookmarks accessible for anybody on the Internet. (Bookmarks are files containing preselected links to some WWW-pages.) Of course, the amount of work required for an implementation of a society of SMs (SSM) is dramatically larger, nevertheless we may receive a non-trivial benefit in some circumstances.

[3] The knowledge is on the meta- meta- level with respect to the Information Universe.

she/he can deliberately select a promising [4] SM. If the machine choose a transformation, the selection can be based on some superficial information (for example on an occurrence of a string) or on a more deep knowledge of the SMs (and of the given state of querying the Information Universe via current SM). For example, if you are interested in articles about analogy and the current SM does not satisfy you, the informed "transformations server" present another SM to you and this SM contains relevant information (not only some documents containing the string "analogy").

The technical details of the proposal will be described in another paper.

Observation 5 *Let us summarize:*

A *the conceptual interface of a society of structured maps consists of a set of structured maps (see Section 3) and of the transformations between the structured maps,*

B *the transformations between structured maps support a dynamics, they enable a change of the view on the data stored in the information universe, and an adaptability of searching and querying with respect to a given context (problems, interests, tasks, responses).*

Of course, the idea of a society of structured maps introduced above has a natural social content. We can assume that structured maps are created by members of a community. Each structured map is characterized by a vocabulary (created and used by members of the community).

Similarly, transformations among structured maps are defined by various members of the community. A definition of a transformation may contain a proposal of triggering conditions for the transformation. The triggering conditions may be dependent on membership of the user in a group.

5 Conclusions

Importance of conceptual interfaces was stressed in the paper. A special attention was devoted to Structured Maps. They form an interesting case of conceptual interfaces. We have introduced and discussed a society of SM's. A kind of dynamics, a change of contexts, and social dimension are characteristic features of SSMs.

Finally, we are preparing a paper devoted to a generalization of SSMs. The generalization is based on a concept of Dynamic Kripke Structures (DKS).

DKS was introduced in [10]. A goal of an ongoing research is to confirm an assumption that a semantics of various kinds of non-monotonic, hypothetical and context-dependent reasoning can be given in terms of DKS.

[4] Let us imagine that I use often Joe's bookmarks, http://www.cs.uni.world/ joe/bookmarks. As I can Joe, a "transformation" to his bookmarks is for me a promising one, when I try to find something on the Internet.

Dynamic Kripke Structures consist of a static part and of a dynamic part. The static part is a usual Kripke structure (it is sufficient here to imagine a Kripke structure as a generalization of a database). The dynamic part is a set of transformations from one element of the Kripke structure to another (from possible worlds to possible worlds).

DKS can be viewed as a formalization (and generalization) of a society of SM's (a semantic specification of the conceptual interface inherent in the society of SM's).

We emphasize role of DKS in cognitive modeling: in a sense they integrate a logical and a psychological approach.

6 List of abbreviations

API – application programmer interface
BIBTEX – a format for bibliographical databases, used in TEX (and its derivatives), a sophisticated program designed to produce high-quality typesetting
DBMS – Database Management Systems
DKS – Dynamic Kripke Structures
HTML – HyperText Markup Language, a collection of styles (formats) used to define the various components of a World Wide Web document
IU – Information Universe
KDD – Knowledge Discovery in Databases
MMI – Man-Machine Interaction
SGML – Standard Generalized Markup Language, a collection of style (formats) used to define style sheets
SM – Structured maps
SQL – Structured Query Language, a standard of the relational database language

Acknowledgements:
Many thanks for valuable comments and recommendations to J.G.Dalenoort. I am indebted also to P. Forbrig and to an anonymous referee.

References

[1] Abiteboul, S., Cluet, S., Christophides, V., MIlo, T., Moerkotte, G., Siméon, J. *Querying documents in object databases.* Int. J. Digit. Libr. (1997) 1: 5-19

[2] Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J., *The Lorel language for semistructured data.* Int. J. Digit. Libr. (1997) 1: 68-88

[3] Barker, P.G., Najah, M., *Pictorial interfaces to data bases,* Int J. Man-Machine Studies (1985), 23, 423-442

[4] Codd, E.F., *A relational model of data for large shared data banks.* Communications of the ACM, June 1970, vol.13, No. 6, 377-387

[5] Codd, E.F., *Relational database: a practical foundation for productivity.* The 1981 Turing Award Lecture, Communications of the ACM, Feb 1982, vol. 25, No. 2, 109-117

[6] Delcambre, L., Maier, D., Reddy, R., Anderson, L., *Structured maps: modeling explicit semantics over a universe of information.* Int. J. Digit. Libr. (1997) 1: 20-35

[7] Imielinski, T., Mannila, H., *A database perspective on knowledge discovery,* Communications of the ACM, vol. 39, No. 11, Nov 1996, 58-64)

[8] Mannila, H., *Methods and problems in data mining.* Proc. of Int. Conf. Of Database Theory, 1997

[9] Mendelzon, A., Mihaila, G., Milo, T., *Querying the World Wide Web.* Int. J. Digit. Libr. (1997) 1: 54-67

[10] Šefránek, J., *Dynamic Kripke structures.* CAEPIA'97, Actas de la VII conferencia de la associaci Española para la inteligencia artificial, Malaga 1997

[11] Ullman, J.D., *Principles of database and knowledge-base systems*, Computer Science press 1989

[12] Ullman, J.D., *Database theory: past and future.* Proc. of PODS, 1987