The Use of Keywords by Context to Relate Items of a U-book

John Bauer Mengelberg and Nancy Hernandez Negrete
Colegio de Postgraduados, Mexico

jbauer@colpos.mx hernandez.nancy@colpos.mx

Abstract

The need to index large data collections and the different types of structures used to store and index them is presented for a particular application: U-books, which are collections of multimedia files that are presented in different ways to its readers. The descriptions of files are stored as items in a database table. Ordered subsets of these items - called sequences - are offered, instead of a single set of items of a book. Since the concept was broadened from an extension of an e-book to other types of uses of the materials, the number of files could be sufficiently large to require additional descriptors to retrieve them individually or in subsets satisfying certain search criteria. To perform these searches efficiently, the items must be indexed in some manner. RISP, a component of SPRP - the software package that implements these books - is used to relate items based on keywords by context, which are tags provided to indicate their interpretation. The data structures and methods of RISP are described, and the need to index the keywords leads to the use of KBC, a particular indexing scheme that is essentially a Generalized Inverted index but also offers the use of equivalent terms which will be indexed by a common term. An example based on a collection of multimedia teaching materials used by schools and colleges is used to explain and illustrate the features of U-books and the software. Some technical details are included as part of the descriptions; others are either included in the Appendix or omitted.

Keywords: e-book, index, generalized inverted index, keywords by context, informing process, information retrieval

Introduction

The proverbial needle in a haystack provides a good way to explain the topic of this paper. There are essentially three ways to look for the needle: you look all over the place; somebody told you it would only be in one small sector of the stacks; or you might be told exactly where it is. In many situations regarding information retrieval, the situation is complicated by the fact that the longest needle is sought but the number of needles is unknown. A search to find one or more elements of a repository or collection of information items is such a situation. A strategy to reduce the amount of work is to *index* the information so that it is not necessary to examine every item individually. Thus such indices are necessary, not just convenient, for almost any large collection of objects.

In order to index something, you have to decide on some attribute or property of the object to which the index will refer. In a book a chapter index helps find a particular topic, but to find a particular word one would use a word index. In a

This paper was accepted through blinded executive review

database, you can include several indices that will point to different elements (fields or combination of columns) of records of the same table. Thus the index itself (not its values) can be considered as a pair: what you are indexing, and the way you build, store and use it.

A client seeking knowledge or information will formulate a question and the programs should respond accordingly. Thus, the software must provide both components: the one that will accept (and understand) questions formulated in a certain way; the other used to find and deliver the results. Search engines are perhaps the best known of such products, since we all use them constantly. Of course there are other information retrieval processes: someone wishes to know what is in a collection, instead of asking for particular items. This is the case in business analytics or data mining applications, but also in many others. The software product will include the convenient indices, probably based on some type of cost-benefit considerations – especially in computer resources and response times, but also constrained by the possibility to include an adequate index: once again, you cannot index something that is not defined or specified as an attribute of the object you are indexing.

This is the topic of this paper: in a particular situation, it was not possible to include the necessary mechanisms to deliver the information so as to satisfy some of the attributes we expect from an informing process. The situation we describe concerns U-books, which can be succinctly described as follows: U-books (Bauer Mengelberg, 2007) are collections of items, which may be thought of as files containing multimedia objects, and sequences, through which ordered subsets of these items are offered to a reader. Further descriptions as well as some comments about the origin of these books will be presented in a separate section: this organization of the paper was chosen in order to introduce the need to include keywords and index them promptly.

There is a collection files which can be almost anything that can reside on a computer disk, including local devices or remote ones. A U-book allows offering these files to readers that differ regarding to what they wish to read or find out. It uses a database to describe the files. An item is a record of a table that contains certain general descriptors of one of more files that represent or refer to the same knowledge, event, fact or other contents, but in different languages or versions. Thus an item is actually a pointer to its files.

We will use a single example of a U-book throughout this paper. A group of schools and colleges decide to share their materials, including almost anything that might be useful to teachers, either for preparing a class, assigning tasks, formulating exams, writing a textbook or class notes, group discussions about teaching methods and techniques, results of certain methods, experiments, evaluations by teachers and students, whatever. They decide to include all their materials in a U-book, “Didactic material”. A name is necessary to organize the book’s data elements.

To organize the collection so that items can be found and used, they establish certain procedures to include their own files as well as web pages and describe them as items of the book. For example, they will adhere to certain classifications of the materials according to standards defined specifically for this instance. As features of U-books are described in the section dedicated to this topic, they will be included in the description of the example.

The teachers and administrators understand it is a lot of work, but it will be done by several persons as items become available or are found somewhere. Other schools and colleges from any country will be invited to join the project. They also agree on certain options offered by U-books. Initially, they only have materials in English and Spanish, but they expect to obtain files in other languages since the will invite other schools and colleges from any country to join the project. They will use versions of items, which they will specify according to a catalogue prepared for this purpose to reflect the content, presentation, types of use or other characteristics of a particular file.

An item of this collection might be an experiment in physics, and several files explain or show the experiment in different ways, using a video-clip, a presentation, a plain text or a link to a page. But versions could also indicate that the content is theoretical or demonstrative. Note that more than one item might refer to the same experiment.

The main use of the items is to assemble sequences according to their purpose. A teacher may want to look at all items that refer to similar experiments, especially those that are useful for his next lecture. He will then build his sequence for the class using some of them; of course he could add items to the collection as he creates them or obtains them elsewhere. Other users may use the book to compose class notes or even a textbook.

They also decide to make use of the restricted privileges feature, whereby certain items are only available to a subset of the users of the book. They understand that these constraints may be voluntary or not: someone wishes to hide certain materials from his students - who of course may be users of the U-book as well as their teachers - or use the constraints so as never to be offered an item that he does not need or will never use. Examples of the constraints used for the latter purpose can be found in an application of U-books for the use of delegates to a conference in (Bauer Mengelberg, 2011). Several different ways are offered to delegates to find out schedules of presentations, especially but not exclusively to select the ones they will attend. The knowledge field of every paper is included as part of its description: this allows a delegate to select the fields in which he is interested, thus preventing other papers to appear in the programs. Delegates can change these constraints either temporarily or permanently whenever they wish to do so.

The delegate U-book also provides an example of the use of several versions of an item. For every paper, the title, authors, knowledge field and the session in which it will be presented constitute one version; another version includes a biography and photograph of the main author, and still another one is the full paper. Some of the sequences prepared for delegates present the first version, and offer the others as an option. One could add *a posteriori* versions to the item, since the book could be offered to delegates as a memory of the conference: a video-clip of the presentation itself; the list of delegates that attended it; or perhaps the discussion that followed the paper. Observe that several files of this book are included in more than 1 item. For example, biographies and photos are included as items referring to authors as well as to their papers.

The teachers in our example particularly appreciate a feature of SPRP, the software product that will allow them to expand their U-book to include material from other countries: it is not only available in several languages, but is translatable (Bauer Mengelberg, 2001) to any other language as well, where this adjective indicates that the interfaces can be translated without any knowledge of programming languages, databases or other technical aspects of the programs, but especially without having to modify the actual programs or database tables.

Though initially conceived as an extension of e-books, U-books can be - and are - used for several other purposes. The example we use here illustrates one such situation, where the number of items is sufficiently large to necessitate more “pointers to items” – as one can interpret indices in this situation - than offered by the fields of the item database table. Thus, items can be furnished with additional descriptors – in the form of keywords – by RISP, a special component of the software and the topic of this paper.

In the Didactic material book, keywords by context are added as needed for an item so it can be retrieved asking for any combinations of some of them, such as the knowledge field (physics), the topic, the persons involved, the use of the items (class, notes), the type of description (theoretical or not), the type of audience (for example, to prepare an actual experiment for a particular class), or several other attributes that may limit the list of items delivered by a search. However, keywords are not enough by themselves: the term “watch” could refer to a timing device, a word in the title of Rembrandt’s Night Watch or as a verb. So the term by itself does not provide an

interpretation. In other words, keywords are examples of unstructured information, a designation that reflects precisely this circumstance, as opposed to the fields of the item's record, whose meanings are conveyed by their position in the record.

To indicate the meaning or interpretation of a keyword to differentiate it from others, we provide it with a tag – called its context. In (Tag metadata, 2010), a tag is defined as "a non-hierarchical keyword or term assigned to a piece of information. This kind of metadata helps describe an item and allows it to be found again by browsing or searching." Since a table of a relational database has fixed columns (fields) and only one value can be provided for each field, a NoSQL database is used, where in computing this term designates database management systems that differ from classic relational database management systems (RDBMS) in some way (NoSQL Concept, 2010). Amongst them, including RISP, many are based on key-value pairs, where a unique key is associated to one or more values. By adding their context, keywords will now be part of what is called semi-structured information.

The inclusion of keywords by context offers additional ways to locate an item or a collection of items that satisfy a particular request, since now the criteria can include them besides the item table's fields. For example, one might wish to retrieve all items that satisfy the condition: [Knowledge field = physics] AND [topic = pendulum] AND [language = English or Spanish] and [level = Expert]. As will be seen, contexts are numbered so this will not be the real query.

To deliver the subset of items that satisfy such a search criterion, either the entire collection must be examined or some type of indices may be available. Since the first alternative is not attractive for large data collections, RISP indexes every [context, keyword] pair that is specified for its items. For that purpose, it uses an indexing product called KBC; it was chosen since, besides being efficient, it offers equivalences, which can be limited by context or language. Observe that this feature is necessary for many applications, including our example. An item can include the keyword "FISICA" whereas another one has "PHYSICS", though they both indicate the same knowledge field. A search based on this keyword should retrieve items regardless of whether they have one value (of the context) or the other. So an equivalence is specified: FISICA will be equivalent to PHYSICS, but only in Spanish and perhaps only in certain contexts. For example, when the term appears in the context "topic" of an item, such as preparación física" (fitness), the equivalence should not apply because the term is used with a different meaning: its context is different.

The quadruplet [equivalent, context, language, basic] will cause KBC to replace the *equivalent* by the *basic* term if the context and language constraints apply (this is the terminology used by KBC). The basic value will be included in the index. Whenever the equivalent term appears, KBC will replace it by the basic value indicated in the equivalence, but the item's description will not be affected.

Besides the features of RISP and KBC, the data structures selected for both of them are discussed, as well as alternatives data models that could have been used. The paper was therefore organized in the following manner.

After U-books are introduced, the need to include keywords by context as additional descriptors of items leads to the inclusion of RISP, the software component that implements them. Equivalences are introduced as part of RISP's features. KBC is then described as the product that manages the indices. Some possible applications of U-books precede the conclusions, which include present and future research in these topics. Some topics and details were included as an Appendix, especially to allow readers to skip subjects or details in which they may not be interested.

U-book – An Informing Tool

U-books as e-books

Several years ago, people started to think about computers as a means to publish and read a book. An electronic book or e-Book is a text and image-based publication in digital form produced on, published by, and readable on computers or other digital devices (Gardiner & Musto, 2010). This definition does not - explicitly - include some of the new possibilities offered by digital books, such as the inclusion of multimedia materials, the *live* references to published papers and books, and very specially, due to its enormous impact on the way a book is now written, the possibility of including links – be they hypertexts or some other way to access the material in non-linear ways. Of course for some readers, the possibility of enlarging letters or having a software product read the texts out aloud may be the main advantage of reading the book on a computer or similar electronic device.

Many of the available software products used to publish such an e-Book emphasize the maximization of the author's revenues. For example, the description of a product called ebook88 (ebook88, 2002) enhances environmental aspects, the cost of producing them and the ease of distribution, adding the possibility of making changes or adding materials to the book. This allows a book to offer current information, a desirable attribute of an informing process (Gackowski, 2005). Of course the concept does not apply to any type of book, but it is vital for processes that offer information, precisely one of the uses of U-books.

In "E-book, an article" (2007) we found an illustrative discussion comparing several meanings of the same terms, such as e-books meaning the material as well as the device to display it, and e-texts as plain text files instead of the bundled files produced by most e-book products. E-texts were first introduced by Michael Hart in 1971, with the purpose of making all information available to everybody (Hart, 1992). This in turn implied offering the material through the most available software products, as is still the goal of many informers. However, the book industry mainly offers e-readers (or e-book devices), portable electronic devices that are designed primarily for the purpose of reading digital books and periodicals. Devices such as Kindle (2011) and Nook Color (2011) have gained acceptance far beyond expectations, so that new models and brands with expanded features, as well as products for use on mobile devices, are offered to the rapidly growing population of electronic book readers. An up-to-date comparison of available devices can be found in (Comparison of e-book readers, 2011) which is also the source of the definition of e-readers reproduced here.

Some of us regarded the above mentioned possibilities – and several others - of digital publishing as the creation of a new concept, somewhat more general than a *book* in the traditional sense of the word, which would eventually replace most printed books, not just due to distribution and ecological considerations, but precisely because they add new dimensions to the information that may be transmitted to a reader. U-books are a result of precisely such an outlook: they intend to take advantage of most of the new features made possible by the fact that they will be read on an electronic device. The concept was formulated as an informing tool, and thus, adheres to Cohen's sentence definition of Informing Science (Cohen, 1999): "The fields that comprise the transdiscipline of Informing Science provide their clientele with information in a form, format, and schedule that maximizes its effectiveness." U-books are very client-oriented; thus many of its features were defined following Birdsall (2009) and Gill and Bhattacherjee (2007).

As stated in the original paper about U-books, they complement the existing enhancements made possible by the electronic medium. Their introduction is not meant to cause a significant change in the publishing world, but to offer a different way to communicate with readers of the material who, as stated, have a much bigger role in this interaction than in traditional books. Additionally,

other uses of the concept and the resulting software product are not only possible, but probably will have greater impact than any new types of books made possible by the added features.

The concepts of U-books are not new, but after a very extensive search, both in the literature and the description of soft-ware products, no similar entity was found. For example “Build a Custom Book” (O’Reilly’s SAFARI-U, 2006) illustrates how a textbook can be created by concatenating articles, papers and excerpts from existing material. The idea is the same, but the differences when compared to U-books definitely outweigh similarities. In this context the recent announcement by Apple of their Ibooks will be mentioned in the final remarks about the applications of U-books, since some details about the latter is necessary to understand them.

The concept of a U-book was reached in three stages. The initial idea was to offer parts of a book in different orderings. If would be grossly unfair to fail to refer to Julio Cortazar’s Hopscotch (the original title is Rayuela) in this context: the book offered several readings by changing the order of its chapters (Cortazar, 1963). Instead, U-books offer items to be rearranged, not entire chapters.

Next, versions and translations of the same item were included, pursuing the goal that Gackowski (2005) formulated as “information reaching the reader should be interpretable during acquisition, a primary requirement in all informing situations”. Additionally, Gackowski (2005), referring to the use of information for a choice between alternatives, suggests that “data or information values or any combination thereof should not only be usable but also useful. It means they can be used effectively, for which they should be interpretable during presentation”. Usefulness, however, is contextual, depending heavily on the situation. For example, in the context of teaching or presenting a subject to an audience, one wishes to achieve a certain degree of success measured by the degree of comprehension, assimilation and in certain instances, belief by the members of the audience of the materials. The way the subject is presented will have a significant effect on these objectives. So offering diverse presentations of the information to different audiences constitutes one of the main tools available to teachers and other communicators.

Finally, the necessity to block some of the items from certain readers led to the inclusion of the *restricted privileges* component as part of the software product. Its primary concern is, rather than preventing unauthorized reading of the books, to bar individual readers from seeing something they should not – or do not want to - see. Hereby, we address another desirable attribute of an informing process: its relevance and interest of the client of the information, which according to Gackowski rank high in the list of such attributes. A reader of a U-book may use chapter and level constraints to bar certain non-interesting items, of course referring to his particular interests.

One may think of a child not being able to see adult material, or a student not having access to the solutions of the problems included in a textbook. On the other hand, a reader may not enjoy maps depicted in a history book or explanations of items that are superfluous for an expert on the subject. An examination of several publishing tools showed that they claim that they would significantly reduce unauthorized use of the materials. On the contrary, we subscribe to the theory of free and unlimited distribution of knowledge, central to many organizations and groups, including the Informing Science Institute which applies this to any type of published material.

U-books emerged as a result of addressing all these attributes, where the U stands for *unstructured*, since the parts (called items) of the book do not necessarily have a place or meaning within the book. Software products such as ITUNES and IPHOTO (Apple Computer, 2006), to name just two of them, allowed an ever increasing number of persons to build a list of items – in this case, a playlist or a photo book - to be enjoyed in a certain order: this is exactly the same as building a sequence of a U-book. As a matter of fact, the existence of such tools makes the explanation of our concept all the harder, since we are treading on known paths: people will not listen to added features or will tend to reject them automatically. We think of this in a peculiar manner:

one seeks analogies when presented with new concepts, and this process in turn causes differences to be ignored or questioned.

As a result of applying the concept underlying U-books to different types of texts, books or other uses, such as the preparation of class materials, presentations, shows, catalogues and many others, several features were added, mainly through changes and additions to SPRP. They were included for different but converging purposes, including mainly: to increase the variety of items of a book; enhance the way to present it to its readers; and provide additional aids to those preparing sequences. Many of the new features included in SPRP were due to shortcomings that surfaced when U-books were used in different areas, admittedly most of them associated directly or indirectly with academic activities.

The software product in its different versions once again proved that perhaps the most difficult attribute of an informing tool is to avoid excessive task complexity (Gill & Hicks, 2006). As is often the case, flexibility may result in added complexity: more and similar features may cause the product to be quite difficult to use. Both the provider and the client of the informing process were taken into account in the design of SPRP, and constant changes reflect previous failures to achieve a certain degree of simplicity as well as improvements in this direction. The authors of a book will have to know how to use certain special components of the programs, or rely on someone else - we refer to them as editors - to perform the necessary tasks. Anybody who has published a book or prepared an e-Book will relate easily to this concept. But, alas, the reader is the main stakeholder, and it is really easy to read a sequence.

The Items of a U-book

Conceptually, an item is a description of one or more files: one might think of it as a pointer to a set of files. Though the items can be used individually for many purposes, their main use is that they can be included in sequences.

Suppose an item of the Didactic materials book is refers to an experiment to explain a pendulum. It contains step by step instructions of what materiels are used and the steps to perform the experiment. However, there are several such explanations that will be included as a single item: some are in Spanish, and others are offered for High School and College students. Besides, some of the explanations were recorded as video-clips, whilst others were texts or Power Point presentations. So the item will *point* to several files. Figure 1 shows the files associated with an item that consists of explanations of the pendulum.

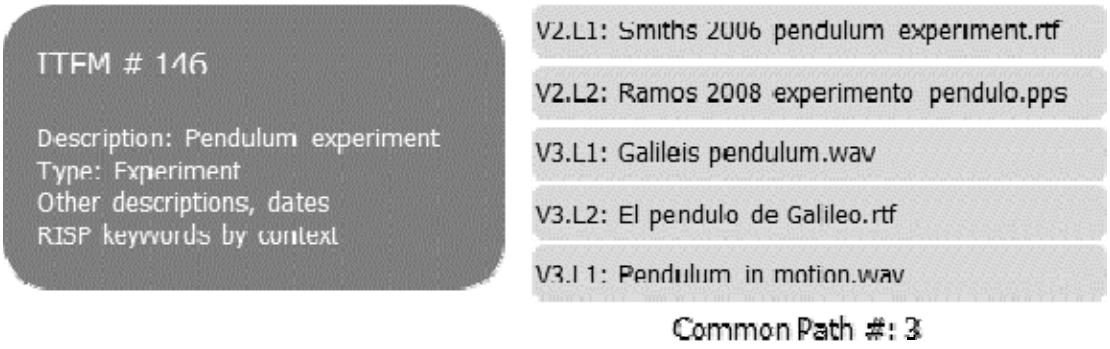

Figure 1. An item of a U-book and its files

The Versions (numeric codes) used by the book were established previously – though others can be added anytime. Here we use Version 1: the default or only version; version 2: elementary material; 3: advanced material; 4: for specialists or experts. Of course there may be others.

SPRP uses numeric codes for file directories, and refers to them as path numbers. Paths are nested in another path (the father path), meaning that the full directory name is obtained by concatenating the father's and the son's path names. This feature was included to obviate the need to change the paths of all files when they are moved to a different location (or computer) or to allow their use in different locations. For example, a user may use a book that resides on a different computer of the network or in a domain on the web. He would only have to change path # 7, and all the book's files will then be used at this location.

Table 1: The path numbers used by the Didactic Material U-book

#	Nested in	Name	Description
1	**0**	C:\	The directory of the user's computer
2	**7**	Didactic material\	the directory created for the book
3	**2**	item files\	part of the files of a book
4	**4**	SPRPfiles\	Resides in the SPRP directory
5	**1**	SPRP\	SPSP's programs and files
6	**2**	RISPdata\	part of the files of a book
7	**0**	C:\	Where the book's data is stored
8	**3**	Physics\	A subdirectory of the item files

Typical path numbers for the book of the example are shown in Table 1. Path numbers are stored in the SITEBASE database. The user of a session can change a path for the purpose of his session, but this will not update the database, which is shared by other users, not only of that book but any others that may have been created. Paths can be added whenever needed.

Sequences

A U-book offers information to its clients, whom we shall call readers of the book, though the term might not apply in some cases such as a student in a class. A reader will be presented some of the items of the book in the form of a sequence: an ordered subset of the items. Figure 2 shows several sequences of a book, though it also illustrates other features. The different shapes used to depict items are meant to show that the corresponding files may be of different (multimedia) types. Additionally, the chapter and level constraints are illustrated.

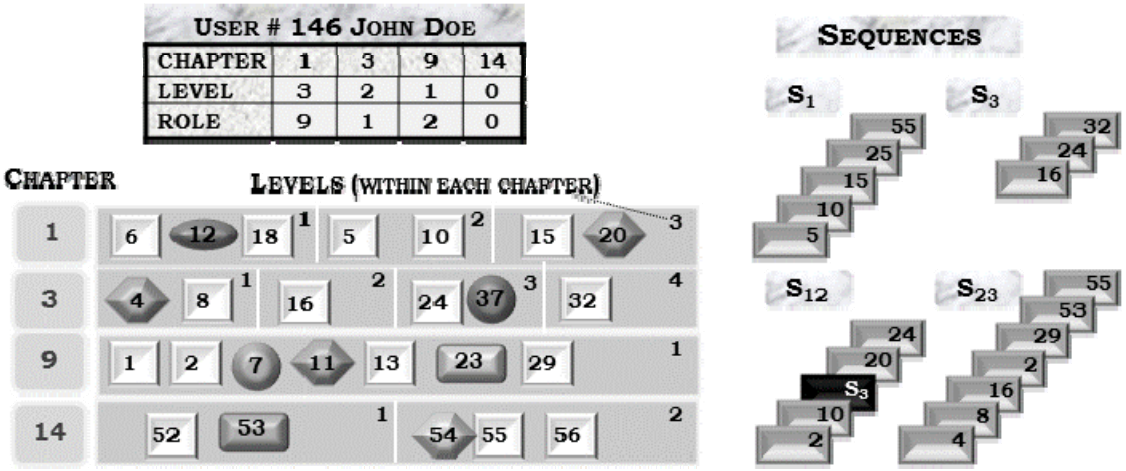

Figure 2. Sequences of items

In the sequences depicted in Figure 2, sequence S12 has sequence S3 as one of its elements, so we refer to the latter as a subsequence of the other one. Thus, S12 will present items 2, 10, 16, 24, 32, 20, 24 in that order. Notice that item 24 occurs twice in the same sequence: there is no reason

to limit an item to appear only once in a sequence. Subsequences can simplify the construction of sequences, especially long ones such as a text-book, or even a presentation. They may serve to divide the task into a number of separate stages, but also save a lot of effort since they are reusable.

A user has a level in every chapter of the book: he may not see items that have a higher level. In the example depicted, user #146 may not see items 24, 37, 32 since he has level 2 in chapter 3, or any of the items of chapter 14. Thus, User #146 would see only items 2, 10, 16, 20 when he reads S12. Once again, we use the terms read or see items of a sequence, though actually the reader may be listening to an audio clip.

Observe that the term *chapter* is used with a totally different meaning than the corresponding term in a book. It was adopted because e-Books are often offered to readers (buyers) in releases: one chapter is sold or offered free, and subsequently the interested reader will purchase the other chapters. This misnomer was caused by our inability to find an alternate designation for such groups of items. Since the term is pretty much transparent to most readers, users of U-books will soon overcome the initial confusion caused by this terminology.

Additionally, in every chapter, a user has a *role* that indicates which functions he may perform with items of that chapter. Roles are: 0=nothing; 1=only read; 2=update items; 9=assign roles of that chapter to other users. For example, User #146 is practically the “owner” of chapter 1 (he can do everything) whereas he is just a reader of chapter 3.

Building a Sequence

Elements of a sequence are either items or other sequences included as subsequences. This feature is recursive, that is, a subsequence may in turn have its own subsequences. (SPRP detects *loops* that may result from this feature.) Elements of the sequence are presented as a list, and deletion and reordering are performed on it by selecting an element and indicating the desired operation. There are essentially three ways to add elements to a sequence.

- An item or a sequence can be added (or inserted anywhere) individually indicating its item number.
- One or more items are selected from a list of items, and added to the sequence.
- A list of items is prepared, using a filter or relations based on their descriptors, and the resulting items added to the sequence. The filters are based on combinations of conditions formulated using the main classifications of items, such as the type of item, one or several chapters and a minimum level of each of them, and the fixed fields that are used to indicate other attributes of an item.
- An existing (previously defined) sequence can be added as a subsequence.
- Elements of an existing sequence can be added as a special case of the filtered lists of items, but changes made to the source subsequence will not be reflected on the new one.

As will be described below, the RISP feature was included mainly but not exclusively to provide additional ways to add elements to a sequence: it will use operations based on the keywords by context to build a result list of items which can be added to the sequence.

From a list of items, there are once again three alternatives:

- All the items of the list are added to the sequence. The user may indicate the position in which he wants to insert them.
- Elements of the list are selected and added to the sequence.
- The file corresponding to each item of the list is displayed one after another to the user, so he can decide if it should be added or not.

Besides its elements, a sequence has attributes of its own, which are introduced using an interface similar to that illustrated in Figure 3. "Default values" for the version and language may be indicated. If not equal to zero, they will override the reader's preferences. As shown below, a version or language can be specified for an element as well; these values will override any previous specifications. Notice that understanding all these details is in no way essential for the comprehension of the main topics of this paper (relating items via RISP).

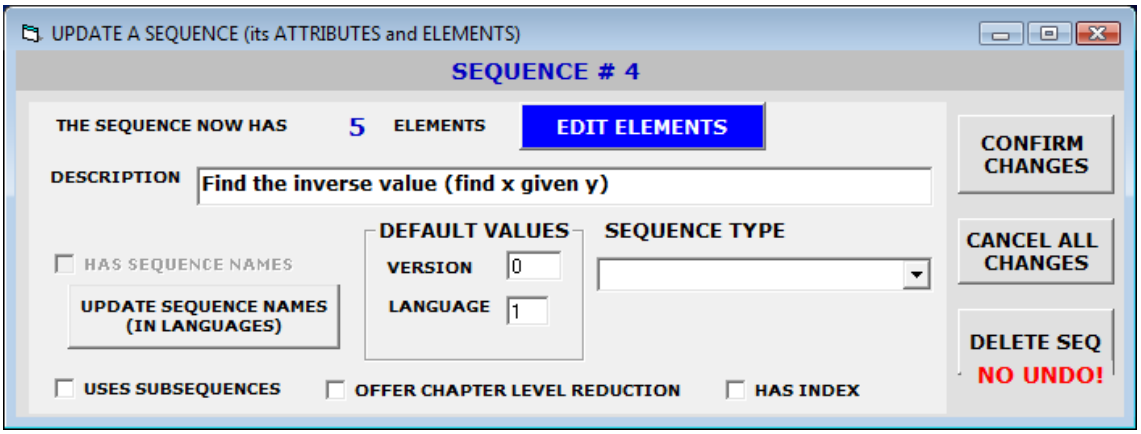

Figure 3. The interface to update options and attributes of a sequence

To add an item as an element of a sequence, a list of items can be invoked. Since this list may be too large to be of any use – think of a list of a few hundred thousand items – a filter is offered so that only items that may be of interest for this purpose are offered. Figure 4 shows a simplified version of the interface to define such a filter.

Observe that RISP can be used to limit this list. We use this function to point out that a user will not see any components of the system he does not need or even know, to avoid confusing users that do not know the concept. For example, in a book where chapter constraints are not used (an option of a book) all references to chapters and levels will disappear from the screens. This partially explains the design of screens, which took this feature into account for an additional reason: a user should not notice *missing* elements.

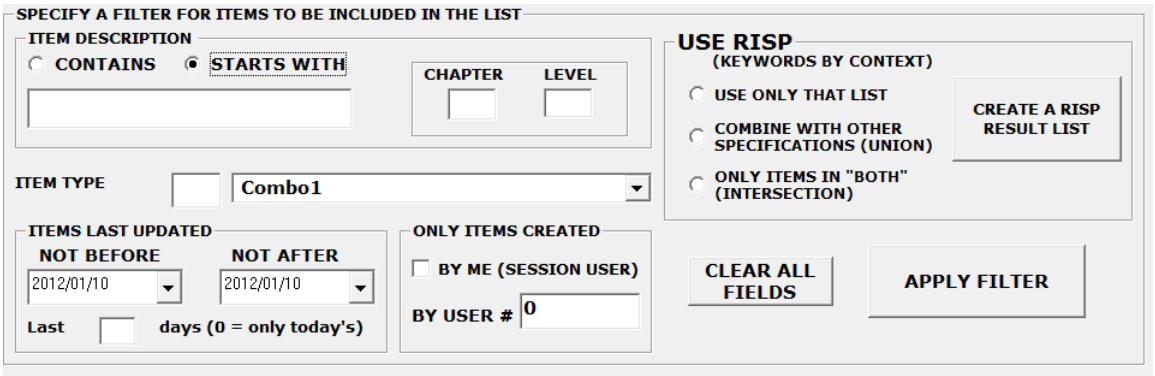

Figure 4. Filtering the list of items offered to the builder of a sequence

An item can be included as an element of a sequence by – first - indicating its number and using the ADD 1 ITEM feature, which will be disabled unless he provides the number. The same is true for subsequences; in both cases, the system will reject the request if the corresponding object is not already in the database. Figure 5 depicts some of the components of the interface used to update the elements of a sequence. Other objects appear when they are invoked, and still others

were left out since they would only complicate explanations and are not needed to understand the essence of the function.

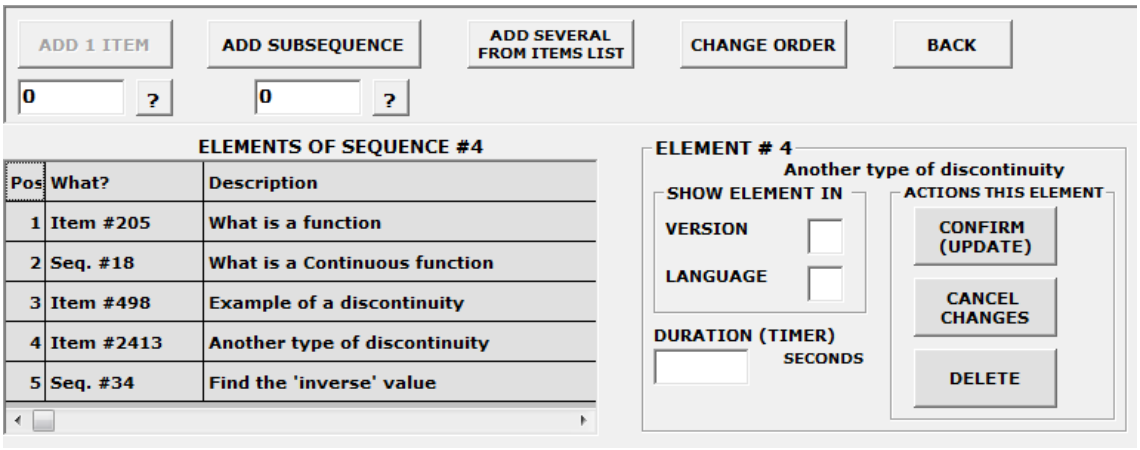

Figure 5. An interface to add and edit elements of a sequence

The "Change order" button will activate a set of fields that allow such operations. The timer feature was included to allow the builder of a sequence to determine the duration of the display of the item as part of a sequence. Actually, one can also indicate the duration for a particular file; the difference is that in this case, the same duration will apply to the item in any sequence, which may or may not be desirable. We use this timer to synchronize certain items when presented simultaneously, either on two (or even three) separate devices or on a split screen, a topic that was not included in this paper but will be mentioned briefly as part of our current and future research.

To round off this section, let us add that a sequence can be used as a source for another one. A new sequence duplicates the other one, typically to provide a user with the possibility to eliminate or add elements.

Reading a Sequence

The reader of a U-book is a user, that is, he has gained access to the book. When he starts a reading session, he will choose one of a list of available sequences. There might be sequences that are reserved for certain users only, for example, somebody built a sequence only for himself. If there are many sequences, he may *filter* this list in several ways, typically by their name or description, but also by the author of the sequence (who built it).

Once he has chosen a particular sequence, he may change certain reading options, especially the choice of a language and version: he will see the items of the sequence in the chosen version whenever they are available; otherwise items will be offered in their default versions. SPRP also *remembers* readers of the book: at the start of a session, even on a different computer, a reader will be offered the last sequence he used, and he may even indicate to restart where he left off.

The elements of the sequence are presented one after another. He may invoke a navigator, including an index of the sequence which can be built for it, to choose the next element he wishes to see. However, he will never be shown or offered an item which he may not see due to a chapter or level constraint.

Finally, a reader can comment elements of a sequence, both for his own use or – optionally – also for the use by others. It is important to note that these comments will not refer to the items themselves, so that they will not apply to the same item as an element of another sequence.

Due to length considerations of an already long paper, several topics related to sequences and therefore to reading sessions of a U-book, are not mentioned, since they have no impact on RISP.

The Data Model Used by SPRP

We decided to limit this section to the main objects of a U-book: Users, chapters and levels, items and sequences, though there are several other tables. The main purpose of this description is to state why keywords had to be stored separately.

SPRP uses 3 main relational databases (SITEBASE, ABOOKBSE AND SEQUENCEBASE) though there are others to store additional and temporary data. For example, the content of files can be stored as fields of a special database, instead of including a pointer to its location. Though this has several usages, the reason it was included in SPRP was to offer encoding of files: this is one of the security features of U-books not described here. There is also a CONSTANTSBASE for every language: it contains data that enables translating the names of fields, message texts and help features. A reading session will use the version according to the language selected for that session. Observe that a translation of the software consists of creating a new instance of this database; separate software products are used to translate the data elements.

SPRP offers books as part of a *site:* think of a publishing house that offers several books. The list of books (instances) is stored here: essentially a title and the location of the databases for each book. Users are registered and assigned a site-role in the site's database, so they will have the same password for any book these use. They also have a role in the site: they may add or delete books, add other users, change the options of the site or update certain catalogues that contain data used by the programs.

For every book of their choice, they become users of that book, and additional data is stored in the USERBOOK and USERCHAPTER tables for that book (in the book's database). A user has a role in every book (for example, he may or not create chapters, include other users or build *public* sequences) and, as explained before, a role in every chapter of the book, once again reflecting if he can use or see items of that chapter or assign chapter-roles to other users.

Items are also stored in this database. The 2 main tables are THEITEMS and ITEMVERSLANG. THEITEMS contains a record for every item where it is described by certain fields. Pointers to the actual files that constitute its content are stored in the other table, where an item has a record for every combination of version and language for which there is a file. The main fields of those records house the location of the corresponding file indicated by the triplet [path-number, filename, extent] described previously.

Sequences are stored in a separate database, to allow a user to use his own copy if this were convenient. It has 2 main tables: SEQUENCES and SEQELEMENTS. The former contains the sequence's options; the latter has a record for every element of the sequence (they are ordered through a principal key formed by the sequence number and the position of the element in the sequence (the system renumbers them automatically when an update occurs). There are other tables but their description would not add anything to the paper: some of them can be found in the previous papers about U-books, though several changes have occurred in the newer versions of the software.

The Need for Additional Descriptors of an Item

A U-book may easily have many hundreds of thousands - or millions - of items, since many elements will be divided into several items to enable their partial use or to be included in several versions or languages. Thus the task of assembling a sequence for a particular purpose may involve either knowledge of *everything* in the repository or just a lot of work.

An item is included in a U-book as a record of a relational database table, and has certain attributes provided by the – fixed - fields of the table. Some of them offer certain flexibility, as is the case with 6 classification fields which can be used as needed by the particular book, or even in a different manner according to the type of item. But even though they provide certain flexibility, only one value of an attribute can be furnished. For example, if one wished to specify several authors of an item, the fields of the table would be insufficient.

So SPRP needs additional descriptors of items, without any constraint on their number or what they indicate. This necessity was implemented by RISP: relate items in structured publishing. The authors (there may be many, as in our example) define what we call contexts. The term was chosen carefully, after considering many alternatives, including ontology. “What is an ontology? Short answer: An ontology is a specification of a conceptualization” (Gruber, 2001). Alternatively, Wikipedia reproduces a definition also due to Gruber (1993) as a "formal, explicit specification of a shared conceptualization" but complements it thus: “An ontology renders shared vocabulary and taxonomy which models a domain with the definition of objects and/or concepts and their properties and relations” (Arvidsson and Flycht-Eriksson, 2012). Since U-books were not conceived as knowledge management or sharing systems (though they can be used for such purposes), we did not deem it appropriate to use those fields’ terminology. So the decision was that for most U-books it suffices to “know what you are talking about” when somebody utters or uses a term, so we called it a context.

RISP: A Way to Relate Items of a U-book

RISP was described briefly in the original paper about U-books, but the way they used this component has changed in the new version of the software. The essence remained the same: provide items with additional descriptors - keywords by context - that allow relations between them to be formulated based on these additional attributes.

We start with a brief history of this product, and then describe it fully. Technical details are provided only when deemed necessary or useful to a reader.

The foregoer of RISP was MARTE (mark and retrieve texts) a product developed several years ago, partially as a Master’s Thesis of a student. Its information units were texts, individual words of which could be marked by context. Thus, they became keywords by context. Actually, colors assigned to each context were used to “tag” the keywords. The keywords by context were included in convenient structures (indexes) which allowed retrieval of the texts that contained them. Boolean operations between subsets of texts that contained a particular keyword provided a very flexible and efficient way to find whatever was sought.

When U-books were developed, the methods of MARTE were used to relate items of the books, but quite independently from the items themselves. Thus, an item of the U-book could correspond to a RISP text (the new name assigned to this feature) or not, and vice-versa, there could be RISP texts that did not (yet) correspond to an item. The same RISP text could describe several items. These features were offered to be able to prepare certain materials for their later inclusion as items of the book. This feature was not successful: it caused confusion, so it was eliminated; in new version of SPRP keywords are added directly to the item.

RISP is described in separate sections for each of these topics: RISP’s functions and how they are used; the data structures used to store the keywords; the indexing method; and finally, certain remarks about the task of furnishing descriptors for large collections of items.

It is important to point out that though the software that implements RISP is described as such (that is, for use in U-books) it is actually available to any application that has similar needs, and therefore, has its own nomenclature. For example, RISP uses the term “item” to refer to the in-

formation units, but the software calls them instances or basic information units (BIU). Therefore RISP is implemented as a separate module (a DLL), and not as part of the application. For example, a U-book may not require RISP at all: its users won't even know there is such a thing available.

The main functions performed by RISP (there are several support functions, such as reorganization of files and recovery of loss of data) include:

- Define and update the context catalogue for the particular U-book;
- Add, delete or show keywords by context of a particular item;
- Request a list of items – a result list, in RISP terminology - satisfying certain conditions, which are formulated based on the keywords by context;
- Perform massive updates or changes (explained in a separate section).

The application – in this case, SPRP, will invoke RISP's functions, essentially:

- Display the keywords of a particular item;
- Update the keywords of an item;
- Build and deliver a result list. The criteria are indicated through a component of RISP, not by SPRP itself;
- Add one of more keywords (provided by SPRP) to an item.

The communication between SPRP and RISP is performed by RISP methods, since they do not have access to the other's data. When SPRP creates an instance of RISP it provides the name of the directory used by the book for the RISP data, and certain options that may limit or enable features of RISP. For example, SPRP may tell RISP that it will not need equivalences, or that it does not use the concept of "language of items" at all.

Most of the interaction is based on parameters. However, result lists may be delivered as disk files or as arrays of integers made available to SPRP. The methods will only receive relevant fields. For example, RISP has the following method:

Public function update-keywords (item-number by value, item-class by value, has-n-marks by Ref, pointer-to-list by ref) as integer

The function will return a negative result if something failed (according to a coded value), the value zero if no changes were made, or 1 if the record should be updated with the new values. SPRP provides the necessary information: the item number, the item type or RISP-class, and the previous values of the two necessary fields: how many isolated marks (as keywords by context are called) and the pointer to the array of these on the disk file. The latter are passed "by reference" so RISP will update them and inform SPRP that the record corresponding to the item has to be updated to reflect the new values.

The Data Model Used by RISP

The data model stores the contexts used by the book and the actual keywords of the items.

The catalogue of contexts for a book

There is a list (catalogue) of contexts used by the book. We decided to include the details in the Appendix for interested readers. Here we only describe fields of the main table, CONTEXTS.

Context-num: Every context is assigned a number between 1 and 999 (this is the maximum supported by RISP).

Type-of-context: 1=unconstrained; 2=unique; 3= "catalogue", where "unique" means there can only be one instance for every value of the context (this corresponds to a unique index). "Cata-

logue" indicates that marks will be rejected if there is no previous instance for the value. Of course there is a way to insert the first instance that occurs.

Names: up to 8 names (descriptions) are stored in every language used by the book.

Actually, the specification of a context includes 2 additional components: the contexts that are applicable to a particular class (in RISP it is the type of item, but in other application it might be determined by other attributes), and a grouping of contexts. These features were added so that in a U-book with a large number of contexts (such as is the case in the example used in this paper), the list of contexts offered when keywords are updated can be limited either by the class of the item or by a selection of a group of contexts. There are also certain technical fields, used by the indexing processes to determine the appropriate data structures it will use for a particular context.

How keywords by context are stored by RISP

A relational database model for keywords by context could have served the purpose: its Table would have the fields Context-number, value and item-number. Since an index would be essential, if clustered indices were not available, the table would be duplicated (though with a different structure). A clustered index is an index that stores the actual data, whereas a non-clustered index is just a pointer to the data (Shapiro, 2006). However, we used an altogether different data structure.

RISP stores the keywords of every item in an array of [context-value] pairs. Such a list has the following elements: The item-number (included to enable recovery and efficient reorganization of the files); a counter of how many keywords compose the list; and finally an array of [context-value] pairs, where the context is a 2 byte integer and the value is a 16 character string: KBC does not handle longer values, so the same constraint was included as part of RISP. Thus, the size (in bytes) of a list is: 4 + 2 + N * 18, where N is the number of pairs in the list.

They are included in a single - logical - plain file, though it may be actually stored as several physical files to avoid extremely large files. When the array is modified (by additions or deletions) the previous version is replaced by the new one. Should the size of the array increase, it will be stored in a different location – since it probably will not fit in the previous one. A B-tree (or order 5) is used to provide reuse of vacated locations, where the key is the number of elements of the list it could contain. Whenever an array is to be stored, its location will be the first free location in which its number of elements is greater or equal than the key. Vacated locations are added to the B-tree. If the largest size available – stored specifically besides being in the last leaf of the tree - is insufficient, the array is added at end of the file. B-trees are described in the APPENDIX for reference.

How Keywords for an Item are Updated

Items of a U-book are introduced individually or in lots or batches. The latter would be the case if several files are to be added as items based on their filenames or other selection criteria. They may already reside in other data structures (databases, lists of files, other U-books or others). The system will prompt users to furnish descriptions and certain attributes of the items, either individually or once again, applicable to a subset of items.

To add keywords to an item, the list of its current keywords (if any) is updated. Figure 6 depicts a screenshot of a simplified version of the interface because the real one is dynamic, meaning objects appear or disappear as needed either by the user of the system itself. Notice that it is part of the "update an item" function, where all attributes of an item are updated, so one can see other attributes of the item as well as display its files.

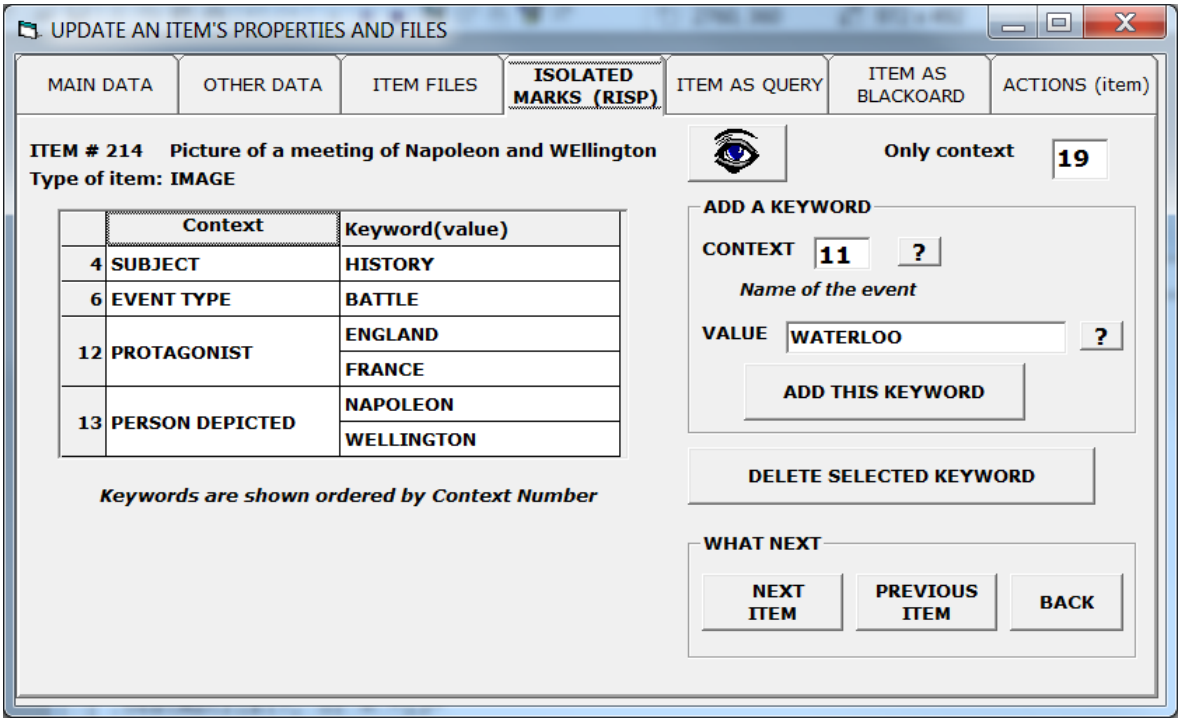

Figure 6. Update of the keywords by context of an item

The "eye" button will display the item in its default version and language (remember there may be others). In SPRP buttons with a "**?**" label indicate that the user wishes to select the value from a list of alternatives. To delete a keyword, select it and use the Delete button. Change of a value is performed with a delete-add combination. After every operation, the list of keywords is updated and presented in the correct order (by context number and value).

To add a keyword, the context can either be provided by its number or invoking a list of applicable (according to the class of the item) contexts. The new value can be keyed in or chosen from a list of all keywords used (previously) by items in that context. Many other features (not described in detail here) were included to alleviate the task of describing items. For example, a "source" item can be indicated and its keywords added to the item being updated.

The "only context" feature: if a context is indicated, only keywords for that context will appear in the list, and values are added or deleted without having to indicate the context: for applications where items can have a large number of keywords, other marks may "get in the way".

The "Next Item" button will show the following item, which depends on the way the function was invoked. A user may build a list of items to be marked, and in this case the following item of the list will be presented. Note that this is particularly useful when the items will inherit keywords from a given (source) item indicated for that purpose.

The Need to Establish Equivalences

One of the main features of KBC - and therefore, RISP – is that it offers equivalences, described in the introduction. An applicable quadruplet [equivalent, context, language, basic] indicates that a keyword with an *equivalent* value will be replaced by the *basic* term if the equivalence is *applicable*, meaning that both the context and language constraints of the equivalence are satisfied. Here we have italicized the actual terminology used by KBC.

When the equivalence is constrained to a particular context, it will not apply to the value when used in other contexts, though it may occur in several contexts. Though equivalences are an essential part of RISP, details about them and the data model used by KBC to store them were once again deferred to the Appendix: the paper is too long!

The Use of Keywords to Retrieve Lists of Items

KBC will build and deliver a *result list* - an array of the numbers of the items that satisfy a formula, which consists of a set of logical operations based on operands. The operands can be [context-value] pairs or other previously obtained result lists. Operations are: AND; OR; XOR; AND NOT; NOT, and parentheses can be used as needed. (Actually it also implements what is known as UNION ALL: elements common to more than one operand are duplicated in the result list.)

Therefore RISP has a module to define formulae: these can be very simple or as complex as necessary. Users are offered several different interfaces according to different levels of expertise using this or other software products. A prototype of one such interface to define a formula is depicted in Figure 7. We ask the reader to ignore the actual formula, result lists and other fields which are neither consistent nor totally applicable to our example.

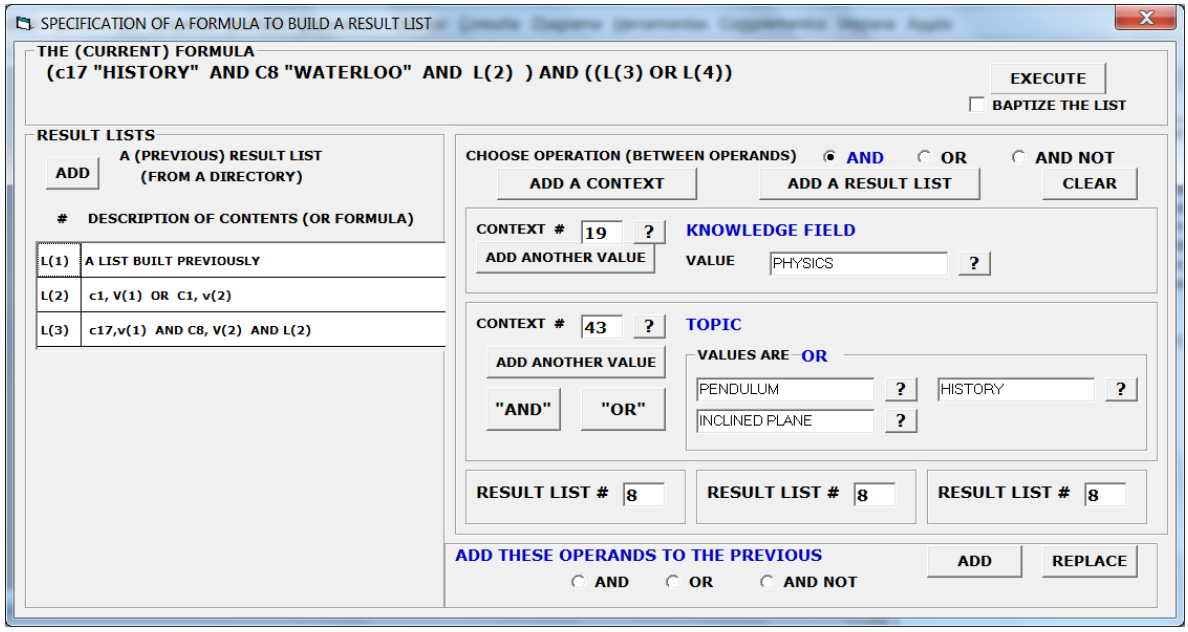

Figure 7. An interface to define a formula in RISP

The operations can be performed in stages (stepwise) to allow back-and-forth queries. After obtaining a list of items resulting from one or more previous operations, another constraints may be added as ("AND") or other items included ("OR"). If this does not satisfy the user for some reason, he can try some other change of the list, and this will not imply performing the same queries again. Note that result lists built previously and stored on disk may also be used: they are added to the list shown in the interface, either specifying their location or choosing them from a file-picker type object. RISP has a special directory used to store temporary result lists so only the name of the file would have to be provided.

The possibility to use result lists in other operations may also greatly simplify the task of formulating a query. Instead of having to formulate an awkward relation, for example involving several levels of parenthesis, the operations to be included in each of these are performed separately and

the result lists are then combined using the appropriate operations. Interfaces were carefully designed to make these tasks easy to understand and use.

KBC to Satisfy the Need to Index Keywords

RISP has the methods to include and store keywords of an item, but there must be a way to build subsets of items that have a common mark, that is, a certain keyword in a given context. Browsing all items and determining if they contain that mark is – though feasible - not a good idea, so an indexing scheme is used. As stated, RISP uses the indexing features offered by a separate software product called KBC: it will store the indices, and provide the methods to use them to build subsets of items based on the presence or a keyword by context or by operations based on such lists.

According to Rob and Coronel (2009) an index is an orderly arrangement used to logically access rows in a table. Every element of the arrangement will be a pair (key, position) where the key is the value of the index. The value is usually that of one particular field or a concatenation of fields. The position is an identification of the row involved, which may be the row number or an indirect reference, for example, the key of a clustered index. Though this definition of an index refers to tables of a database, it may be applied to many other types of data collections, as long as the key points to an element of the collection. A discussion of different types was included in the Appendix, especially Generalized Inverted Index (GIN) since it is the type used by KBC. These indices have become popular for very large databases, especially in Key-value stores.

We had used these indices in the past, though the name had not been coined, especially to index values of several tables (not just one as in databases) and especially to allow several values of the "same field" to occur in the same record. Since this is the case in RISP, the choice of an appropriate indexing method was quite straightforward, though several variations were introduced in KBC.

Every keyword furnished for an item is included in the index corresponding to the context of the keyword. KBC receives triplets [instance number, context, value] from RISP or any other application. The instance number is an identifier provided by the application: RISP uses the (unique) item-number as this identifier. KBC forces applications to provide such a unique identifier. The context is a 2-byte integer and the value is a string with no more than 16 characters (this is a constraint imposed by KBC).

KBC stores keywords and uses then to build result lists. Several generations of this product were produced, actually with different names (KBC is the last and probably final name). The main difference in these generations – one might call them versions, but it would be a misnomer because the functionality changed – is twofold. The structures used to store the indices varied significantly, and the first version did not include equivalences.

KBC serves applications with widely differing cardinalities (number of items, contexts, values or instances). A typical U-book will not have a huge number of items, but the DBB database - a NoSQL database not totally implemented and therefore not yet published - may contain many billions of records. This in turn caused certain data structures used by previous versions of KBC to be insufficient as far as providing efficient retrieval and updates of indices. KBC stores the triplets using the following model. For every context, it builds a B-tree of its values (the ones for which triplets have been added). The leaves of the B-tree point to the location of another B-tree formed by the instance numbers for that value. Thus, for every context, it is a GIN that stores [value, instance] pairs.

The version of KBC currently used by RISP will be described here. The newest version is in the final stages of development, and as soon it is ready it will replace the previous one. This will not

only be a simple operation (a DLL is replaced by another one) but also transparent to RISP - and therefore to SPRP - users.

Due to a limit on the programming time allocated to the project (a few days) instead of using its own B-trees, these were implemented using relational database tables, though admittedly a very non-normal one, which is described in the following section. Experience (and many simulations) proved they were very efficient even for large collections of data. A formula requesting certain operations based on keywords for a set of 400,000,000 sales records, for example, was delivered in approximately 100 milliseconds for a short formula on a laptop computer; the longest such duration – about 5 seconds – was due to a formula with many operands and extremely long result lists. The description of this simulation, conducted as part of the DBB project, was never published, but is available per request to the authors of this paper.

KBC: The Product that Manages the Indices

The two main functions of KBC are:

- Update an index value: include or exclude a triplet in the index corresponding to that context; if there is an applicable equivalence for the value, it will be replaced the its equivalent;
- Build a list of instances satisfying a (context, value) pair as described previously; the value will be replaced by an applicable equivalent (if there is such) before performing the operations.

Of course KBC performs many additional functions, some of which will be described and others are left to the imagination of the interested reader: KBC is a complete software package, so it performs all the tasks that can be expected from an indexing method.

The result of a query (formula) submitted to KBC is a result list that contains the item-numbers that satisfy the selection criteria. KBC can prepare and deliver this list as one of several data structures, though two are basic and the others are combinations of these: an array of item numbers: a bitmap which will indicate which items are in the subset by a value of 1 in the corresponding bit. Suppose the U-book contains items numbered from 1 to 3000. The result list would be an array of 3000 bits (725 bytes). If the list contains items number 15, 73 and 98, the corresponding bits would be turned on, whilst all others would have a value of 0. The use of bitmaps was obviously not included for subsets such as this one, where an array would have been considerably more efficient. But logical operations on bitmaps are much faster then merging arrays. Bitmaps are explained in a separate section of the Appendix.

The Database Tables and Other Structures Used by KBC

Instead of using its own B-trees, KBC uses 2 relational database tables (MUNI and MTIP), though a separate instance of the table is used for each context. MUNI is used for values of unique contexts, whilst MTIP serves others. MUNI was not described here, since the way records are retrieved and used is similar to the other one, though its fields are different.

We now describe the MTIP table used to store [Value, instance] pairs for a context, but not before issuing the following warning: this table is awkward, and may require a bit of effort to understand, or alternatively, a skipping strategy. The reason to use such a model, instead of a straightforward one, is that the main enemy – though not the only one - of a database table is the number of its records. The repetition of values of related fields and duplicate information due to the indexes are also drawbacks. In a table with the fields context-number, value and instance number, the two first fields would contain the same values for many records. Dividing the table into several smaller ones (by using a table per context) was an obvious strategy. Allowing several in-

stance values per record was the other, and finally, to avoid huge collections of records for the same value, the pointers to lists described below were used.

The actual name of a table is MTIP*nnn*, where *nnn* is the context number. The table will contain one of more records for every value of that context. These records are used in one of 2 different ways (one might say the fields are interpreted differently). For values with less than 100 instances, the instance numbers are stored as fields of the record, whereas for those with more than 100, the other design applies, where pointers to arrays of instance-numbers - stored separately, that is, not in the database – are indicated in the records.

Figure 8 depicts records for a value of the context ("JONES") and shows the changes caused by two consecutive insertions. Though there are actually 16 instance numbers in the table, the model depicted assumes there are only 8: of course the concept is the same. The fields of this table are:

- VALUE; the value of the context for which the list of instances will be stored in that record. If value = "JONES", only instances of that value will be stored on the record.
- Min_instance_number: no instance number less than this can be stored here.
- Upper-bound_of_instance_number: no instance number equal or larger than this can be stored here.
- Numbers_are_instances: false indicates that the fields contain pointers to arrays of instances.
- How_many_positions_are_occupied (of the 16 available).
- N1, N2 … N16. 16 fields that represent an array of 16 integers, of course implemented as 16 separate integer fields. It has 2 different interpretations, and will be explained separately. But the way records are created and retrieved is the same for both situations.

VALUE	MIN-NUM	UPPER_BND	OCCUPIED	N1	N2	N3	N4	N5	N6	N7	N8
JONES	1	654	5	23	40	67	123	130	158	215	
JONES	654	LN	1	654							
Insert new value 140 (it fits)											
VALUE	**MIN-NUM**	**UPPER_BND**	**OCCUPIED**	**N1**	**N2**	**N3**	**N4**	**N5**	**N6**	**N7**	**N8**
JONES	1	654	6	23	40	67	123	130	140	158	215
JONES	654	LN	1	654							
Insert new value 151 (the array was full)											
VALUE	**MIN-NUM**	**UPPER_BND**	**OCCUPIED**	**N1**	**N2**	**N3**	**N4**	**N5**	**N6**	**N7**	**N8**
JONES	1	151	4	23	40	67	123	130	140		
JONES	151	654	3	151	158	215					
JONES	654	LN	2	654	871						

Figure 8. Result of adding 2 successive new instance numbers of a value

If there are many instances, the fields are used in a different manner. Essentially, 8 of the 16 fields are used to indicate the first element of a list of fixed length (currently set at 128 elements), and are stored in a plain file with random access. The other 8 fields contain pointers to the location of the list on that file. In Figure 9, used to illustrate this design, once again only fields for 4 values and 4 pointers are illustrated, though as we said, there are really 8 of each of these. Only a single record is shown, since the way to insert a new (first of list, pointer) pair is exactly the same as for the other use of the table explained before.

VALUE	MIN-NUM	UPPER_BND	OCCUPIED	N1	P1	N2	P2	N3	P3	N4	P4
SMITH	1	LN	3	23	54	6035	17	9276	5		

REC #	OCCUPIED	ELEMENTS OF THE LIST (array of integers)
5	72	9276, 10432, 10572, ... (72 values)
17	3	6035, 7777, 8003
54	128	23, 40, 900, (128 values, the last was not larger than 6034)

Figure 9. Use of the fields as pointers to lists of instances

Observation: obviously, there will be many empty slots in the lists. KBC has a utility program to compress the tables and files: a new version of the table is created and records for each value are added, so that all lists but the last one (for that value) will be full.

Result Lists in KBC

A result list is an ordered set of item numbers that satisfy certain criteria, the simplest of which is: find the set of all items that include the pair [context C, value V]. Suppose the list of items marked with [Context = 43, Value = "PENDULUM"] is requested by the application. KBC will retrieve all the records with a query such as *"Select * from MTIP043 where value = 'PENDULUM' order by min_instance_number"*. Then it will act according to the Numbers_are_instances field to find all the instance numbers for value V, and add them to an (empty) array of numbers. Observe that if the fields of the table contain pointers, the corresponding lists may be added as such, not one element at a time. Though this will leave empty slots in the array (minimized if the files were compressed), the process is much faster. This is quite significant if the value has millions of instances. In the simulation mentioned above, there were values of contexts that had several hundred thousand instances.

Operations involving other keywords by context and previously built result-lists are also performed by KBC. As was mentioned, RISP builds formulae and KBC delivers the corresponding result lists. For each operand - either a previously built result list or a [context, value] pair - it will build the list for the pair and then perform the operation on the lists involved.

Actually, KBC uses different types of result lists:

- An array of item numbers
- A bitmap representing the item numbers in the list
- A list of bitmap intervals representing the numbers.

The choice of the appropriate type of list to be built is either made by the request or by a special component of KBC which determines the appropriate type. This happens when the list is constructed as part of a formula, since operations on bitmaps are much more efficient than merging lists. Perhaps a comment on these bitmaps is appropriate: they may save memory if the lists are lengthy and if the ratio (number of elements of the list / largest instance number in the application) is relatively high, say at least 20%. This happens often, since there may be contexts with few values, and a large number of items – or even all of them - are marked with the context. For example, most applications will "mark" the class of the item (in RISP, the item-type). Though some readers would have liked to see the actual routines that perform these operations, the already excessive length of this paper prompted the decision not to include them. They consist of several routines to combine a number of arrays for an operation, or using the AND and OR operations between numbers (for bitmaps).

Features of RISP to Alleviate the Task of Key-Wording

We consider the process of key-wording large collections of items as part of a set of methods included in our programs called "massive updates". Essentially, a criterion is indicated which will create a subset of items, and then a marking scheme is indicated. Instead of attempting to explain the underlying theory, we use an example to indicate the nature of the process. Many other ways to perform similar tasks are implemented in RISP.

Suppose the collection includes many item referring to MATHEMATICS, some of which will describe the topic CALCULUS, and that both of these keywords have already been included in the corresponding items. However, we wish to indicate if an item refers to (or includes some reference to) the fact that it was created by a teacher called JONES and/or by another one called SMITH. There is a context number 32 defined precisely to allow providing this information about items. The process to include these keywords involves 3 steps.

- Build a (result) list of all items satisfying: (19, "MATHEMATICS) and (32 = CALCULUS) and, as stated before, these marks have been introduced previously and context 19 is the knowledge field and 32 is the topic;
- Indicate the marks you wish to include, and the way the user will select the items to which they apply via certain keystrokes (or clicks of a mouse or otherwise);
- Execute the program.

The system will display the items of the result list one after another and the user takes the appropriate action. Notice that it does not show a record, but the item itself, for example, an image, a text or even a presentation.

Figures 10 and 11 show prototypes of the forms used to invoke these operations. The actual interfaces are dynamic, meaning they adapt to options as they are selected. For example, certain users may prefer clicks of the mouse or may have touch-screens. The programs offer many ways to perform the tasks, and will adapt the "show the item" form according to the user's preferences. Additionally, more complicated criteria or rules may be included. Other facilities are of the type: use all items marked at one stage as a starting list for another massive marking sessions; this allows step-wise marking of items with very little effort. The system also allows specifying several contexts at a time.

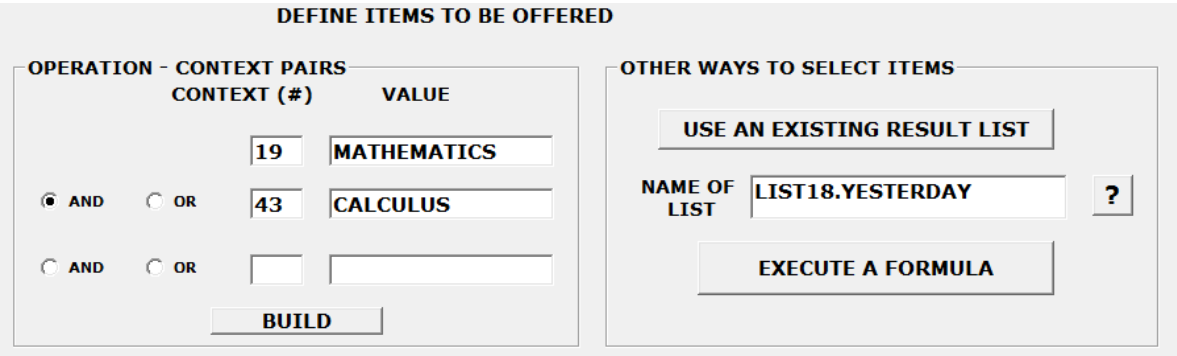

Figure 10. Interface used to specify the items to be offered for key-wording

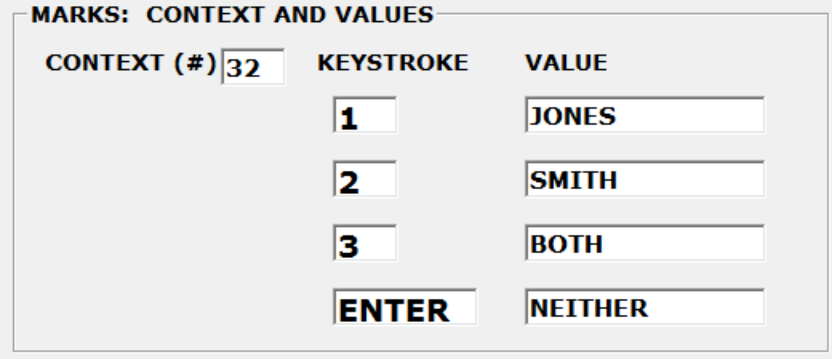

Figure 11. Interface used to specify the way to include keywords

Different Applications of U-books

U-books were conceived for a particular book. A young marine in the Second World War was part of a little known invasion of the Southern Coast of France in August 1944. He spent the next 60 years gathering material of all sorts about this event. Of course he wished to publish his research, and chose to do this in the form of a novel. Thus, most of the material could not have been included. Our suggestion to use a U-book, created precisely for him, was not followed, and the book was published several years later (Sussna, 2008). However, even the 740 pages of the published book were insufficient to include the wealth of documents, affidavits, interviews, photographs, maps, layouts and other items he had assembled. The use of a U-book would have had a disadvantage (it would not have appeared in print) but this would have been – partially – offset by a number of advantages, led by the possibility to include all his materials. He would have built several sequences for different types of readers, some of which would be quite similar to the actual book, but perhaps with the possibility to show interested readers some of the details he could not furnish in the book, as well as the opposite.

As it turned out, besides composing different versions of the same story or books where the reader had a selection of ways to enjoy it, some other obvious applications surfaced. Playlists or similar applications were the first of these. Actually, the example of the use of chapter constraints in the first description of U-books was the catalogue of a museum that would enable its readers to eliminate certain parts that did not interest them.

A text-book where a student might browse an initial version, have the complete one during his semester, but not having to prepare for the pre-exam cramming version, was amongst these. Very recently (February 2012) Apple announced a new software product, called an Ibook which is described as "Author Lets Anyone Create Stunning iBook Textbooks" (Apple, 2012). This prompted several of my former students to inform me of this *misfortune,* but I managed to calm their worries since the - enviable - product only performs very few of the functions offered by Ubooks, though with the huge advantage of making it available to readers on their Ipads. To point out just a few differences, Ubooks three main attributes are not present: present several readings of the same book, offer materials in different versions to readers with different interests or approaches, and finally the restricted privileges component which allows certain items to remain hidden to readers who should not see them or would like to skip them. Actually these new books are just variations of what has been offered in the past (for example Safari-U mentioned above) with the possibility to view them on mobile devices.

Several *non-book* applications surfaced as the product gained form. We mention some of them briefly. The documentation of an information system or a software product such as SPRP provided an example of a situation where U-books could be really helpful. RISP would be wonderful to describe items of the set of documents, but also to easily prepare the inevitable presentations.

The possibility to reuse certain items for different purposes is not exclusive of this product, but is made easier with the features offered by the software.

In the U-book for delegates of a conference a totally different use was explained. We have also conceived their use for a restaurant: the menu could be offered in many different ways, including languages, levels of understanding of its customers and the amount of detail provided for the different dishes, but also making use of multimedia files. And the use of computers and mobile devices is becoming common in certain types of restaurants.

When applications for teachers were considered, we thought U-books could become a part of techniques applicable to e-learning, besides situations as the one described by our example, which refer to in-class teaching. They seem to offer a different and perhaps efficient way to build and deliver learning objects. However, the greater challenge was to answer: what could the concepts of U-books do to increase a teacher's toolkit?

When reading this paper, which is inevitably about U-books though it should be about RISP, one might still think of several uses of the latter, even without using other features of SPRP. One would use the items as if they were part of a document filing system, and every once in a while could produce a list of certain of its documents via a sequence. The authors of this paper actually want to organize "everything" they have in this manner, especially all research materials. An attempt to compose this paper using this strategy failed only because not enough time was dedicated to the task. We started creating items: paragraphs, quotes, references including the actual papers whenever they were available, figures, tables, screenshots and other materials. This would in turn facilitate the rewriting - especially reordering – of the paper and allow images to be interchanged. The final version would be a copy and paste operation. Alas, we will attempt to apply this technique in other papers.

The previous section was about one of the main barriers to the use of U-books for several of their possible applications: the need to provide descriptions of the items. Almost everyone presented with explanations of U-books asks if there is some way to do this without having to actually indicate the keywords. Our answer is always the same: yes, it is possible; no, the software does not do it. It contains no elements of knowledge discovery. We thing that eventually it should have such elements, but not as part of the software itself, but through ways to include other products' results, that is, providing a way to accept keywords stored in other databases. The present version does this partially: the import facilities can be used to obtain descriptors of files, but their conversion to keywords by context for the time being still requires specific coding. It is considerably easier to design a program that does this for a given product than for any knowledge discovery program.

Conclusions

The use of keywords by context to further describe items of a U-book was explained, after providing background about the books. The way in which sequences of a U-book are constructed, and the convenience of having further descriptions of items especially to build lists of these to be included in sequences led to the use of keywords by context. Equivalences between keywords with the same interpretation were described, especially the feature that allows these relations to be limited by context or language. The process to add keywords to items was illustrated, and KBC was described as the software product that provides the indices needed to use the added descriptors. Some technical comments were provided, especially to indicate the depth of the product but also to allow interested readers to have a more concrete grasp of the software.

The newest version of SPRP is in its final stage of development; some of its components will only be added after a usable version including the main features is available. This is precisely part of our current research. Several features were added to U-books, especially due to the new em-

phasis on teaching applications. Items may now be blackboards, indicating that a screen is provided for a teacher to write or draw explanations. One can also include live queries to any database as items, meaning that they will be executed when the item is invoked as an element of a sequence.

Perhaps the most significant additions were what we called associated elements: another item (or a sequence) may now be added to an element of a sequence. These associated items may in turn be displayed simultaneously with the original element, or alternatively, meaning that the associated item will replace the original one whenever the user indicates he wishes this to happen. The session can now use more than one device: it can use other projectors, monitors or television sets connected either to the user's computer or using an additional computer connected to the former. Thus the simultaneous associated items can be displayed on another monitor. Since an associated item may be a sequence, some of its elements may in turn have an associated simultaneous item. The maximum implemented in SPRP is 3 of these concurrent items.

The same is true for speakers: the audio component can be switched from one set of speakers to another. Once again, a maximum of three sets of speakers is imposed, though both computers could have this number of speakers. These features were added to provide a sort of extended reality, so that one may simulate a conversation between 2 or more persons as part of a lecture or show. Timers were included to enable synchronization of items displayed simultaneously.

Differences between versions of SPRP also address the complexity issue: we might say that now almost anybody can use the product, a circumstance that was far from true in the first versions, notwithstanding the great amount of effort invested in its design.

Further research is primarily concerned with applications of U-books, especially in teaching. How can they serve different learning styles? And what other features could be offered to teachers? For example to switch their presentation due to some circumstance such as discovering that nobody was paying attention or that several members of his or her class did not understand what was being presented. The use of touch screens, pads and pens is also included. So far there is no software to provide use of sequences on mobile devices, another topic that not only worries us, but also motivates future efforts.

There are quite a few research topics that might be conducted by specialists in knowledge discovery and management, as described in the paper. Our research team does not include such researchers, so we hope others will take up the challenge.

Efforts to improve efficiency have pretty much concluded, since the response times for all functions now almost satisfy our standards: usually we do not tolerate any duration of more than a second, and only operations involving huge data collections and complex queries may take a little longer, but never more than a few seconds. Finally, since current programs were written using Visual Basic, the main program (the one used to build and show sequences) is being converted to Visual Basic.Net, and the next step is to produce an online version.

We think that the concept and software will eventually have many users, when people discover that it is not only useful, but relatively easy to use. As stated, there is an inherent difficulty in explaining U-books: everyone already knows what they are. But U-books are useful and the RISP component described in the paper is partially responsible for this fact.

References

Adabas Comprehensive Data Management System. (2010). Retrieved November 11, 2011 from http://www.softwareag.com/corporate/products/adabas _ 2010/default.asp

Apple Computer Inc, Iphoto and Itunes. (2006). Software product description. Retrieved February 21, 2011 from http://www.apple.com/ilife/iphoto/

Apple Computer Inc. (2012). *Apple reinvents textbooks with iBooks 2 for iPad – New iBooks*. Apple.com. January 19, 2012. Retrieved February 27, 2012 from http://www.apple.com/pr/library/2012/01/19Apple-Reinvents-Textbooks-with-iBooks-2-for-iPad.html

Arvidsson, F., & Flycht-Eriksson, A (2009). Ontologies I (PDF). Retrieved December 4 from http://www.ida.liu.se/~janma/SemWeb/Slides/ontologies1.pdf

Bartunov, O., & Sigaev, T. (2006). *PostgreSQL* Summit. Retrieved March 26, 2011 from http://www.sigaev.ru/gin/Gin.pdf

Bauer Mengelberg, J. R. (2001). A set of components to include security, on-line translation and other features in information systems. *AMCIS 2001 Proceedings.* Paper 247.

Bauer Mengelberg, J. R. (2007). The concept of an unstructured book and the software to publish and read it. Information and beyond: Part II. *Journal of Issues in Informing Science and Information Technology, 4,* 801-810. Santa Rosa. CA: Informing Science Press.

Bauer Mengelberg, J. R. (2011). The use of u-books to inform delegates to a conference. *Proceedings of Informing Science & IT Education Conference (InSITE) 2011*

Bayer, R., & McCreight, E. (1970). *Organization and maintenance of large ordered indices*. Original Historic Documents, Technical Report Di-82-0989, Boeing Scientific Research Laboratories.

Bayer, R., & McCreight, E. (1972). Organization and maintenance of large ordered indices. *Acta Informatica, 1*(3), 173-189.

Birdsall, W. F. (2009). The role of the client in informing science: To be informed and to inform. *Informing Science: the International Journal of an Emerging Transdiscipline*, *12,* 147-157.

Cohen, E. (1999). Reconceptualizing information systems as a field of the transdiscipline informing science: From ugly duckling to swan. *Journal of Computing and Information Technology*, *7*(3), 213-219.

Comer, D. (1979). The ubiquitious B-tree. *Computing Surveys*, *ll*(2), 121-137.

Comparison of e-book readers. (2011). Wikipedia.Retrieved March 8, 2011 from http://en.wikipedia.org/wiki/Comparison_of_e-book_readers

Cortázar, J. (1963). *Hopscotch*. Julio Cortazar's Hopscotch. Retrieved August 18, 2010 from http://www3.iath.virginia.edu/elab/hfl0117.html

E-book. (2007). Wikipedia. Retrieved March 2, 2009 from http://en.wikipedia.org/wiki/E-book

ebook88: ebook Resources. (2002). *Software product description*. Retrieved November 3, 2006 from http://www.ebook88.com/articles.html#3.%20What%20software%20do%20I%20need

Gackowski, Z. (2005). Informing systems in business environments: A purpose focused view. *Informing Science Journal, 8*, 101-122. Available at http://inform.nu/Articles/Vol8/v8p101-122Gack.pdf

Gardiner, E., & Musto, R.G. (2010). The electronic book. In M. F. Suarez & H. R. Woudhuysen, *The Oxford companion to the book*. Oxford: Oxford University Press.

Gill, T. G. & Bhattacherjee, A. (2007). The informing sciences at a crossroads: The role of the client. *Informing Science Journal*, *10,* 17-39. Retrieved February 20, 2010 from http://inform.nu/Articles/Vol10/ISJv10p017-039Gill317.pdf

Gill, T. G. & Hicks, R. C. (2006). Task complexity and informing science: A synthesis. *Informing Science Journal*, *8*, 1-30. Retrieved February 20, 2010 from http://www.inform.nu/Articles/Vol9/v9p001-030Gill46.pdf

Greenwald, R., Stackowiak, R., & Stern, J. (2004). *Oracle Essentials: Oracle Database 11g*. Boston, MA: O'Reilly.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition 5* (2), 199–220. Retrieved December 6 2011 from http://tomgruber.org/writing/ontolingua-kaj-1993.pdf

Gruber, T. R. (2001). *What is an Ontology?*. Stanford University. Retrieved November 2011 from http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

Hart, M. (1992). *History and philosophy of project Gutenberg*. Retrieved March 3, 2007 from http://promo.net/pg/history.html

Kindle 3 Wi-Fi 3G. Amazon. (2011). Retrieved March 7, 2011 from http://www.amazon.com/s/ref=nb_sb_noss?url=search-alias%3Ddigital-text&field-keywords=kindle+wifi+3g&x=0&y=0

Nookcolor. (2011). Barnes & Noble Retrieved from http://www.barnesandnoble.com/nookcolor/index.asp

NoSQL (CONCEPT). (2010). Wikipedia. Available: http://en.wikipedia.org/wiki/nosql_(concept)

Null, L., & Lobur, J. (2006). *The essentials of computer organization and architecture*. n.p.: Jones and Bartlett, pp. 741-742.

O'Reilly's Safari Online. (2006). Product description. Retrieved September 7, 2006 from http://safari.oreilly.com

Rob, P., & Coronel, C. (2009). *Database systems: Design, implementation, and management*. Boston, MA: Course Technology, pp. 90-91.

Shapiro, J. (2006). *Microsoft SQL Server 2005: The complete reference.* NY: Mc Graw Hill, pp. 54-55.

Sussna, S. (2008). *Defeat and triumph: The story of a controversial Allied invasion and French rebirth*. Jerusalem, Israel: Xlibris Corporation.

Tag metadata. (2010). Wikipedia. Retrieved January 2011 from http://en.wikipedia.org/wiki/Tag_%28metadata%29

The PostgreSQL Global Development Group. (2009). *PostgreSQL 8.4 Official Documentatio, Volume V*. n.p.: Linbrary, pp. 161-162.

Weiss, M. A. (1993). Data *structures and Algorithm Analysis*. Redwood City, CA: The Benjamin Cummings, pp. 133-138.

Appendix

B-trees

Weiss (1993) defines a directed tree – recursively - as a collection of n nodes, which may be empty. Otherwise, a tree consists of a distinguished node r, called the root, and zero or more (sub) trees, each of them connected by a directed edge to r. A B-tree was first defined by Bayer and McCreight in 1972, though there is a previous reference by the same authors (Bayer and McCreight, 1970) thus: Let h >= 0 be an integer, k a natural number. A directed tree T is in the class t(k,h) of B-trees if T is either empty (h=0) or has the following properties: i) Each path from the root to any leaf has the same length h, also called the height of T, i.e., h = number of nodes in the path; ii) each node except the root and the leaves has at least k + 1 sons. The root is a leaf or has at least two sons; iii) each node has at most 2k+1 sons.

Comer (Comer, 1979) was not the first to define a B+ Tree, but he includes it in a description of certain variations of B-trees specially used to store an index. In a B+ tree, all keys reside in the leaves. The upper levels, which are organized as a B-tree, consist only of an index set (its nodes), a roadmap to enable rapid location of the index and key parts. Figure 12 shows the logical separation of the index and key parts in a B+ tree. Naturally, index nodes and leaf nodes may have different formats or even different sizes. In particular, leaf nodes are usually linked together left-to-right, as shown, to al-low sequential processing of the leaves. The linked list of leaves is referred to as the sequence set.

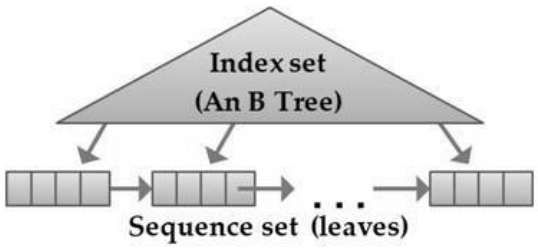

Figure 12. The index and sequence sets of a B-tree

Bitmaps

A bitmap is an array of 0's and 1's where each element represents an element into an index. A bitmap index is used in data warehouse applications – and many others - for tables with large number of rows in which a small number of column values repeat many times. In (Greenwald and Stackowiak, 2004) a bitmap index is described stating that each bit in the index represents a RowId. If a particular row contains a particular value, the bit for that row is "turned on" in the bitmap for that value.

In KBC the bitmaps are represented by arrays of 4-byte integers. Bits are numbered in each integer: bit 0 is the least significant bit of the integer, that is, the one representing 2^0, and bit 31 is the sign bit. The reason to adopt this strategy, instead of using an array of 1-byte integers and their 8 bits, as is often done, is that the amount of operations to use a list may be reduced considerably. For example, when building a result-list or using it, a value of 0 of an integer will allow skipping the 32 "positions" represented by the integer, instead of only 8 positions that would result from the use of 1-byte integers. Analogously, similar savings are obtained when performing logical operations between bitmaps or during conversion of a bitmap into an array of instances. We use the array Only_bit_set (0 to 31) to turn bits on and off or to determine their value. Its values are Only_bit_set (k) = 2^k for k = 0 to 30, and Only_bit_set (31) = -2^{31}

A similar array of constants contains the complements: values of numbers where all bits except the one corresponding to the index are on. Its values are: Only_bit_off (I) = $-(2^{31} - 1 - 2^{i})$ Only_bit_off (31) = $2^{31} - 1$

Thus, the operation 2235 AND Only_bit_off (17) will turn off bit 17 of the number. Two frequent operations are: NUM = -2^{31} indicates all bits are ON, and if NUM = 0 they are all off.

Bitmaps are used by KBC whenever they can save processing time but space considerations are also taken into account. Operations between bitmaps are fast and easy to program. For example, to obtain the union of 2 bitmaps of length Len1 and Len2, which will be represented by 2 arrays of 4 byte integers ARR1 and ARR2, we build the FINAL as a copy of the longest one (supposing the initial positions to which the bitmap refer are the same). Finally, for each element of the other array (say it is array 2), we use FINAL (k) = FINAL (k) OR ARR2 (k). When several long result lists are combined the number of operations can cause considerable processing time, even on very fast computers and efficient algorithms. This never happens when bitmaps are used. Actually, many applications nowadays use this structure since they manage huge collections of data.

Types of Indexes

As quoted in the paper, Rob and Coronel define an index as an orderly arrangement used to logically access rows in a table. Every element of the will be a pair (key, position) where the key is the value of the index. The value is usually that of one particular field, or a concatenation of

fields. The position is an identification of the row involved. This definition is applicable to other data collections, not only to databases. In RISP, the position will be the item-number.

To maintain the order of the elements during updates, convenient data structures are used to store its values. The considerations that should be taken into account when choosing the adequate data structure for a particular index are: the space necessary to store all its values and the number of operations required to update or use the index. The size of the index is important since usually it will be loaded into memory, besides the obvious disk space considerations. However, there is another critical aspect, especially when the index refers to a very large collection of data: the number of input-output (I-O) operations necessary to use the index, either to load it totally or partially in memory or for other uses, is crucial, since I-O operations often contribute a significant part of the total processing time required for a particular task.

Many strategies are used to reduce the number of I-O operations, but probably the prevalent one is to use paging of some type, where nodes are not retrieved individually, but stored and loaded into pages of memory. Shapiro (Shapiro, 2006) defines paging as the process of moving data in and out of physical memory, whereas Null & Lobur (2006) interpret it as a method usually used for implementing virtual memory in which main memory is divided into fixed size blocks (frames) and programs are divided into the same size blocks (pages).

Rob and Coronel, in the same reference, state that most database management systems implement indexes using one of the following data structures; Hash indexes, where a hash algorithm is used to create a hash value from a key column; B-tree indexes, the default and most common type of index used in databases; and Bitmap indexes as described here. Generalized Inverted Indexes (GIN) described below, have joined this list since they are used extensively, especially for large collections of data and very specially to index key /value stores, that is, where unstructured or semi-structured data is present, a situation that does not happen in relational databases, where all values are included in specific fields of a table. We had used these indices in the past, though the name had not been coined, especially to index values of several tables (not just one as in databases) and especially to allow several values of the “same field” to occur in the same record. This is the case in RISP, of course, so the choice of an appropriate indexing method was quite straightforward.

Generalized Inverted Index (GIN)

The Adabas database (Adabas, 2010) was the first commercial product to use inverted lists in the early 70’s, but the term GIN had not been coined, so we reproduce the definition of a GIN provided by PostgreSQL (2009): It is an index structure storing a set of (key, posting list) pairs, where a “posting list” is a set or rows in which the key occurs. Each indexed value can contain many keys, so the same row ID can appear in multiple posting lists. Internally, a GIN index contains a B-Tree index constructed over keys, where each key is an element of the indexed value (a member of an array, for example) and where each tuple in a leaf page is either a pointer to a B-tree over heap pointers (PT, posting tree), or a list of heap pointers (PL, posting list) if the list is small enough. Sometimes, this list of values is represented by a bitmap. In RISP terminology, it stores the list of item numbers (row number) that contain a given value (the key). Observe that a separate GIN is used for each context.

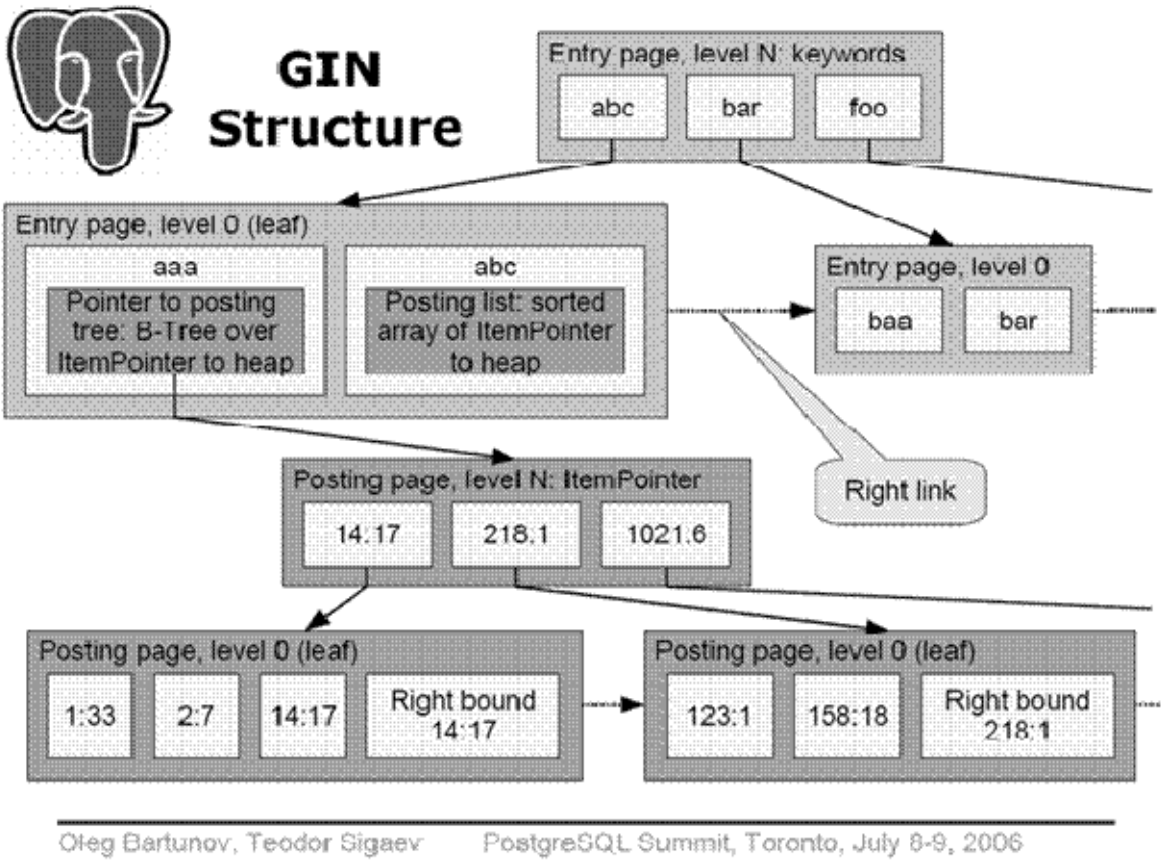

Figure 13. A structure used for a Generalized Inverted Index

Figure 13 shows a GIN structure diagramed by Bartunov O. y Sigaev T. (2006). Each indexed value can contain many keys, so the same row ID can appear in multiple posting lists. Internally, a GIN index contains a B-Tree index constructed over keys, where each key is an element of the indexed value (a member of an array, for example) and where each tuple in a leaf page is either a pointer to a B-tree over heap pointers (PT, posting tree), or a list of heap pointers (PL, posting list) if the list is small enough. KBC actually adds two additional ways to store the lists: a bitmap representing all possible values of the positions, or a list of bitmap intervals, as we have call them if they only represent part of the possible values. Often in applications certain keys will only occur in certain ranges of the numbered records. In a collection of – consecutively numbered – 100,000,000 records, it could happen that certain keys could only occur in two ranges, say 50,000,001 to 60,000,001 or 120,000,001 to 140,000,001. To store several bitmaps of length 12 Megabytes (approximately) for the entire domain might be unfeasible, but the smaller 1 or 2 megabyte arrays may be handled.

As we pointed out before, it is not only a matter of performing operations with result lists: the actual construction of the individual lists, though very fast if a posting list is used, will be considerably slower if a B-tree were used. Remember that nowadays applications not only handle billions of records, but they are used simultaneously by many users, so that computing resources may be strained.

Equivalences in RISP and KBC

The concept and its use

A DBB-equivalence (further simply referred to as an equivalence) is a quadruplet [Equivalent, context, language, Basic] where Basic indicates the value that is actually indexed, and equivalent is a term - or value - that will be replaced by the Basic when a keyword [context, equivalent] is included in a particular language. Note that we refer to the terms actually used by indices as BASIC, and thus other terms which have the same meaning are called EQUIVALENTS. That is, the equivalent term will be replaced by the basic one.

The name has two backgrounds: DBB because it was designed for the DBB database; and x-equivalent as an instance of an equivalence relation in Mathematics. However it is important to note that it is a misnomer: it is neither transitive, nor reflexive nor symmetric. Though all of these properties are present *in abstracto*, they do not apply as far to the use of the equivalences: these are only one-way relationships. In other words, if A is DBB-equivalent to B in context C1, they actually indicate the same concept but are not interchangeable except in the direction A will be replaced by B. An obvious comment: if an item is marked with "A" in context C1, the list of keywords of the item will contain the duplet [C1, "A"], that is, the original term is preserved as the descriptor of the item.

Suppose the mark [Context 23, Value "RETRATO"] is added to an item, where Context 23 is "type of material" and the user determines that when the language is 2 (Spanish) then "RETRATO" is equivalent to "PORTRAIT". When a triplet [23, "RETRATO", item-number] is sent to KBC, the language is included as a parameter and thus, since there is an equivalence applicable to ["RETRATO, 23, 2]) the mark is included in the index with the value "PORTRAIT". This substitution would also be made in any request to retrieve items of this [Context, Value] pair, but only if the user (or the system) indicates that the value was provided in language 2.

An equivalence may specify a context of 0 (zero): this means it applies to all contexts. Similarly, language = 0 indicates that it applies regardless of the language. For example, the equivalence ("EEUU", 0, 0,"USA") could be introduced so that any other name would be replaced by the adopted standard one, regardless of the context in which it is used or in which language it occurred (actually for this type of terms, no language would be specified by the user when invoking one of KBC's functions).

Equivalences are also useful to introduce codes. For example, colors may be coded in some way, so that in every language the color green is assigned the value "4". Items will be marked with "GREEN", "VERDE", "VERT", "GRUEN" (not "GRÜN since no special characters are included in marks) but the index would always use "4". Observe that users can still ask for the value in its original form, and when they request a list of values of the context, they will be offered "4" as well as "GREEN" if they specify that the language is English. (Actually there is an option to exclude the basic value from the list if it has an equivalent.)

Constraints on equivalences (what should not happen)

Chained references (one term is equivalent to another one, which in turn is equivalent to a third) are prohibited in KBC. Thus, when adding an equivalence to the corresponding catalogue, it will not be performed if the "basic" term is present as an *applicable* equivalent of yet another word. Also, the equivalence must be unique: there cannot be another applicable equivalence for the term being added. Here *applicable* indicates that it applies for the combination of context and language specified by its use.

These are the functions performed by the programs regarding equivalences:

1. Add an equivalence: This may be done individually or as a batch of such equivalences, typically when many items are created by a process.
2. Delete an equivalence (changes are made via delete-add).
3. List all equivalents for a particular (basic) value.
4. List all equivalences for a particular context (by language or all).
5. Find the equivalent of a value to include it in an update or as a search criterion. Of course this is the main (most used and essential) function.
6. Produce a list of all values that may have been used as keywords for a particular context. This list should not only include all basic values (the values as used by the index), but also their equivalents applicable to the particular context. This list may or may not be limited to one language. The function will provide the list of values for which keywords were added to items in a certain context, such as needed by someone formulating a query or search criterion to build a result-list. Observe that the list of basic values may not suffice for this purpose. The user may specify he does not with the list to include a basic value for which there is at least one equivalent.

The data model used for equivalences

Table 2: The EQUIVALENCES table of the relational model

Field name	Data Type	Description
Equivalent	Char (16)	The term as used in a keyword
Context-num	Integer-2	The context in which the equivalence applies
Language-num	Integer-2	The language in which the equivalence applies
Basic	Char (16)	The term that is inserted in the index

We start the description with a comment: the type of model used is of course transparent to users of RISP. Additionally, in a typical U-book there probably won't be millions or even hundreds of thousands of equivalence quadruplets, so performance issues are not important as far as RISP goes. Therefore, the current version of KBC uses a very simple model: a single relational database table called EQUIVALENCES, described in Table 2. Integer-2 indicates a 2 byte integer. Char (16) is a string of not more than 16 characters. Observe that we use upper case for the name of the table, and capitalize field names to avoid confusion, and have replaced underscores with hyphens. The principal index is formed by the first 3 fields. A secondary index (the Basic field) is added for performance.

Since KBC also serves applications which might have a very large number of equivalences, the newest version of KBC uses a different model. To describe it succinctly, it uses a B-tree of all "terms" (values of any context) that are part of an equivalence, either as equivalent or as basic. Every term points to an array of equivalences containing the term, though the array is separated into 2 parts: as an equivalent or as a basic. In each element of the array, the context and language as well as the "other" term are specified. Other are added for further efficiency. In several simulated situations, the model outperformed the relation model by a considerable factor. One advantage is the size of the "main index": the B-tree usually can be loaded into memory even for huge collections of data. Disk space is not a significant factor (anymore), but if no clustered indices are used, the relational model will be considerably larger. However, the main reason to use the hybrid model is that the most frequent function (finding an equivalent) is much faster: though even in

the relational model it only takes milliseconds, in the hybrid model it consists of 1 search in a B-tree (in memory) and 1 access to a plain disk file.

Examples of queries to achieve the functions

Some examples are provided for an interested reader. The most frequent function is of course: find the applicable equivalent of E1 for context C1 and language L1. The query is: "Select Basic from EQUIVALENCES where (Equivalent = E1) and (Context-num = c1 or Context-num = 0) and (Language-num = L1 or Language-num = 0)". This will return no record if there is no applicable equivalence, or furnish the correct Basic value. Observe that there cannot be 2 values due to the constraints placed on equivalences.

Now consider the process of adding a quadruplet [E1, C1, L1, B1] where C1 or L1 – or both - could be 0. First determine if there is a previous equivalent for E1 in those contexts of languages. If there is none (and thus the operation may be feasible) you still have to check that B1 is not in an applicable equivalence of its own. You could perform the following query: "Select Equivalent, Basic from EQUIVALENCES where (equivalent = E1 or equivalent = B1) and (Context-num = c1 or Context-num = 0) and (Language-num = L1 or Language-num = 0)". If this query produces a record, the add operation should fail for any of the 2 reasons, and you could inform the user which equivalence is the one that causes the conflict. Two successive queries could also be used to determine the absence of an equivalence that would block the new one.

Some of the other functions also may be performed with straightforward queries, but a relatively frequent one requires specific coding: Produce a list of all values that may have been used as keywords for a particular context and perhaps, in a given language. Here we have to find all the equivalent terms for every basic term that occurs in the index. So we start building the list of all values in the index (this is performed on the index's structures, not the equivalences) and sort the list in ascending order. Next we have to find all the applicable equivalents of every basic value in this list.

One way to do this is as follows. Build the list of all applicable equivalences (constrained by context and or language as usual): "Select Equivalent, Basic from Equivalences where (Context-num = c1 or Context-num = 0) and (Language-num = L1 or Language-num = 0), order by Basic". Finally, merge the two lists (both are sorted by the Basic field). KBC performs this with specific code, since it cannot be done using a query. Observe that the query will either browse the entire table, or use or build a "Context" index. This may be a good reason to include the Context index as part of the design, especially if the application uses many contexts that allow equivalences.

An improvement of the data model could be obtained by using a clustered key if they are offered by the RDBMS. This will cause the records to be stored in the order of the index, instead of building a structure to contain the index besides the table itself. If the table is not very large, it will be loaded into memory, just as the index would be if its size allows that operation. One would still add the other index – the Basic field.

Updates of equivalences

The interface used to update equivalences is difficult to show via a screenshot, since it has many options that cause objects to appear and disappear as needed. Perhaps the prototype shown in Figure 14 will serve the purpose partially. The system will not perform any updates until the operation is confirmed; this allows the use of the same interface for certain queries about equivalences, but also to "try" an equivalence to find out if you should introduce it. For example, there might be a different "basic" value that you should use. Notice that a very important validation is necessary (unless the user is sure it is not the case): the term you furnish to be replaced should not be in a triplet already in the index (that is, the value was included as a keyword of an item in this

context *before* you declare the equivalence). Note that such triplets would never be accessed after you update the equivalence.

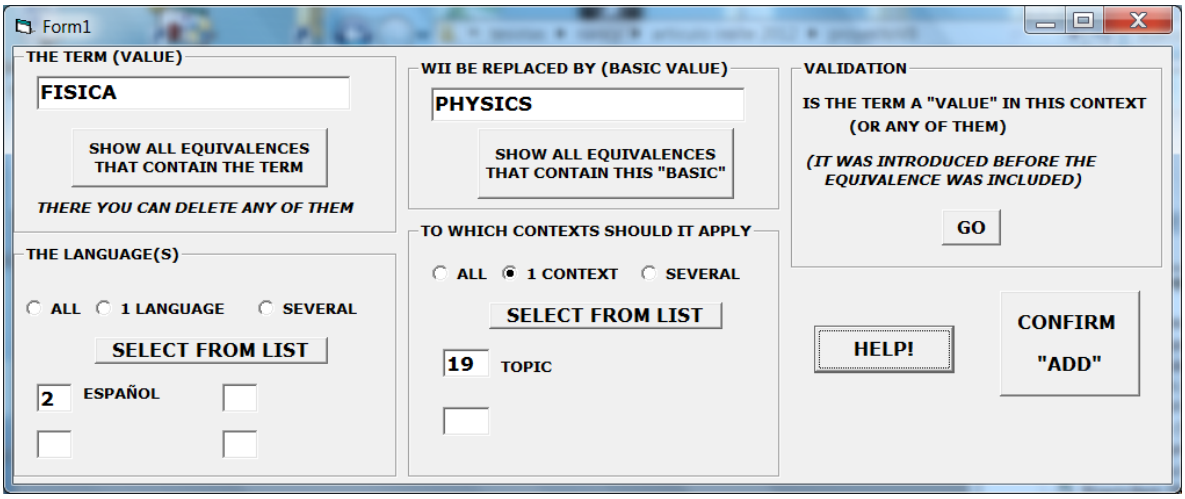

Figure 14. The interface to update equivalences

The “Select from list” buttons will display the requested list (languages or contexts). The descriptions of contexts will be furnished in the language of the session, unless a single language was indicated for the equivalence: then the name of the context will appear in that language. The “SHOW” commands will cause the equivalences to appear on the screen.

Contexts in RISP

When RISP is invoked by an application, it loads this catalogue into memory, so the model was designed for this purpose, and not to provide constant access to the tables during execution. This should explain why it is somewhat complicated or even confusing. The description of the model begins with the fields of the context table (already described in the paper) and is followed by the complementary tables. The explanations provided for certain fields were used to introduce further concepts regarding the contexts. Note that the context is described in every language used by the book (though there is a maximum of 8 languages). Please note that in this paper we have replaced underscores used for the actual names of fields by hyphens. Also, we use upper case for table names and capitalize field names.

Fields of the CONTEXT table

Context-num: Every context is assigned a number between 1 and 999 (this is the maximum supported by RISP).

Type-of-context: 1=unconstrained; 2=unique; 3= “catalogue”, where “unique” means there can only be one instance for every value of the context (this corresponds to a unique index). “Catalogue” indicates that marks will be rejected if there is no previous instance for the value. Of course there is a way to insert the first instance that occurs.

Names: up to 8 names (descriptions) are stored in every language used by the book.

Actually, the specification of a context includes 2 additional components: the contexts that are applicable to a particular class (in RISP it is the type of item, but in other application it might be determined by other attributes), and a grouping of contexts. These features were added so that in a U-book with a large number of contexts (such as is the case in the example used in this paper), the list of contexts offered when keywords are updated can be limited either by the class of the item or by a selection of a group of contexts.

Table GROUPS: group-number; description in 8 languages.

Table CLASS-CONTEXT: class-number; applicable-contexts. The latter is a 128 byte string (1024 bits, which are interpreted as 128 4-byte integers by the program). Its first 999 bits are turned ON if the context is used by the class. Note that we could have used a 125 byte string instead, but decided to waste 3 bytes per class. The table is used to create an array for every class at the start of a session, so the model was designed for efficiency, not for ease of use.

Table GROUP-CONTEXT: group-number, context-included-in-group. The same design as in the previous table was used.

Thus, RISP has a component to create and update the Catalogue of Contexts it will use. Its design was chosen to enable even an occasional user to operate its functions without having to remember how it is done (this is what *we* call user-friendly). The convenience of the users was also taken into account. For example, two ways to indicate the classes that will use a context are furnished: select contexts used by 1 class, or indicate the classes that use a context.

Biographies

Juan R. Bauer Mengelberg (aka. John). After obtaining a degree in Mathematics at the Universidad de Buenos Aires, Argentina (1963), he got a PhD in Statistics and Operations Research at the University of Wisconsin, at Madison (1970) where he also taught courses in the area of Stochastic Programming. He has since worked in Mexico, where besides teaching at the Colegio de Postgraduados, a school primarily involved in the field of Agronomy but also offers graduate degrees in Statistics and Applied Computing, he has held several positions, always connected with the field of Information Systems, in which he has also been a consultant all his professional life. He is concerned with the subject of "systems that work", a concept he formulated to indicate that they work even in abnormal circumstances. He has created and implemented many software packages, and his current research includes informing processes - especially teaching and electronic publishing – as well as hybrid databases and data-warehouses.

Nancy Hernández Negrete. A Master's Degree in Applied Computing (2010) completed her current education after obtaining a Bachelor's degree in Administrative Informatics, both in Mexico. Her thesis dealt precisely with KBC, basically in the selection of efficient data structures and their implementation. She presently works as a systems analyst and programmer at the CRIL department (Crop Research Informatics Laboratory) of CIMMYT (International Maize and Wheat Improvement Center), an international research center with its headquarters in Mexico. Her plans include obtaining a Doctorate degree in the field of data-warehousing and analytics.