Intelligent Computation of Presentation Documents

Joseph D. Oldham and Victor W. Marek and Mirosław Truszczyński

Department of Computer Science, University of Kentucky, Lexington, KY 40506,
{oldham,marek,mirek}@cs.engr.uky.edu

Abstract

Intelligent presentation of data requires flexibility of expression based on user needs and data content, both of which evolve. This flexibility is not offered by the current generation of database management systems. To address this problem systematically we are developing editing tools to quickly build intelligent, user-tailored presentation systems for databases.

Our presentation systems are called *computational registers* (registers). We describe registers as an architecture for generating documents that summarize data with a particular class of user in mind. A system to manage creation and maintenance of registers is a *register system.* We describe DEXTER, our own register system currently under development.

1 Introduction

There are groups of users who do not have sufficient access to information from databases for the following reasons:
1. Insufficient ability to use the standard database front end.
2. Inadequate presentation of the data for the user's specialized needs.
3. Lack of familiarity with the scheme of the database at hand.
In hospitals, for example, physicians, nurses and payment agents all need access to pieces of medical records, all have distinct perspectives on the data (beyond database views), and often lack skills and time to overcome the limitations of the presentation capabilities of database systems.

The problem is an impedance mismatch between users and data management systems. Intelligent presentation of data requires flexibility of expression based on user needs and data content, both of which evolve. This flexibility is not offered by the current generation of database management systems. To address this problem systematically we are developing editing tools to quickly build intelligent, user-tailored presentation systems for databases [5, 1].

Specialized presentation software is often written to present data to a particular kind of user. Our goal is to move toward automating this process. Our method is to describe an architecture for presentation, which we call *computational registers* (registers.) We are developing software tools for specifying, building, coordinating and maintaining registers. We refer to these tools as a *register system.* In this paper we will discuss registers, and DEXTER (Data EXpression Through Edited Registers), a register system we are developing.

Here is an example of data and a desired expression of this data in a form customized for a particular class of users. We will consider this example throughout the paper.

Example 1. Suppose our database contains the following data:

```
NAME            COURSE  Asn1  Poss1  Wt1  Asn2  ...  Asn12
Joe Marek       CS121   91    100    0.1   0    ...   -1
Victor Oldham   CS121   80    100    0.1   80   ...   -1
```

The summary we want is a "midterm grade report" to be sent to a student. We assume that our register operates on the data on a tuple by tuple basis. Outputs for these tuples might be as follows:

```
To: Joe Marek
Re: CS 121 Mid term grade report
Your mid term grade, based on a weighted average of 74.5 on due
assignments, is C.  PLEASE NOTE that homework 2 is missing.
Your score is based on the following homework scores: 91, 0, 84,
and a midterm score of 94.

To: Victor Oldham
Re: CS 121 Mid term grade report
Your weighted average of 85.4 on due assignments earns a mid term
grade of B.  Your score is based on the following homework scores:
90, 90, 90 and a midterm score of 80.
```

We assume that the database record contains all the essential information on which we would base our report. A database stores a lowest common denominator form of data. The fundamental task of a register is to restore to the data a semantics that reflects the user's perspective on the data. To achieve this functionality in an intuitive way, registers take advantage of conventions and language familiar to the user. For any register the range of expression is constrained. A report for a different purpose would be generated differently, hence would require a distinct register. Registers related by applicability to the same data, which also exist under the same register system, may share resources.

2 Registers and Register Systems

Computational registers are designed to generate, from a record of known structure (as in a database), a summary document (including multimedia) represented in a language such as HTML or SGML ([10, 15, 16].) *Register* describes an approach to presentation without regard for specifics of how that approach is implemented.

Specifically, a *computational register* consists of:
1. *Register domain description*: ontology, vocabulary and vernacular
2. *Register processing specification*: field, mode and tenor mappings.

Registers are processed by a *register execution engine.* This is a a register-independent program that interprets field, mode and tenor specifications and creates output documents on the basis of external database records.

To facilitate building registers, we need a set of tools that will allow rapid development of register components. This set of tools together with the register execution engine will be called a *register system.* Thus, a *register system* consists of:
1. An *authoring system* designed to allow creation and modification of registers
2. A *register execution engine* to process registers developed in the authoring system.
In the discussion that follows we always keep in mind that our end goal is not a register, but a register system. A register system should allow the register author to focus on data presentation rather than programming details, and allow resources for a set of registers to be shared and kept up to date.

2.1 Register domain description

Register domain description components specify information that must be available to the register for coherent management of presentations.

Ontology refers to a set of classes, including both system and author-defined classes within the register system, sufficient to represent database information within the register. Objects of these classes are the *terms* of the register.

Example 2. We should be able to represent data about a student for the report in some intuitive way in the ontology:

```
    Class Student
          StudentName     Name
          Homework[n]     Homeworks
          Exam            MidTermExam
          Character       CurrentGrade
          .  .  .
```

The register *vocabulary* is a dictionary of strings . In a register system, a register vocabulary is a subset of the system wide vocabulary. Vocabulary entries for register systems carry semantic information. In an attempt to represent data with user-appropriate semantics registers rely on conventions of language and presentation common in the targeted user group. While this language is powerful, it can be subject to misunderstanding. Online definition of terms supports both user trust and authoring coherence.

Example 3. The vocabulary of the register for the midterm-grade example must contain an entry for the string “midterm grade”. It might look as follows:

```
MIDTERM GRADE (noun: count, singular) Pertinent classes: STUDENT.
Synonyms: GRADE, CURRENT GRADE.
A letter GRADE assigned based on performance on work ....
```

The last component of a register's domain description is called *vernacular*. The vernacular is a collection of *register phrases*, which: express relationships on facts established by the register field; are used to form the document outline by the register mode; and are given expression in the register tenor.

Example 4. Here is an example of a phrase definition for a text phrase:

```
Phrase   SCOREEARNS( WA, G )
         1. your midterm grade, based on a weighted average of
              $WA$ on due assignments, is $G
         2. your weighted average of $WA on due assignments earns
              a midterm grade of $G
```

This phrase will yield one of the texts listed in its definition as a possible way of expressing the relation between the weighted average and the mid-term grade. We treat phrase definitions as text templates. There are other possibilities.

2.2 Register Processing Description

Register processing consists of field, mode and tenor processing. *Field processing* uses the field specification of a register, e.g. a set of rules, to transform a record in the original database scheme into a record in another scheme, called the *internal scheme*. The internal scheme is defined in terms of the classes of register's ontology. The transformation determined by the field is a database mapping, which may disregard some information, may synthesize new attributes from several original ones, and may break old attributes into several new ones.

All access to the original data occurs in field processing. If field processing is completed before the other phases are allowed to begin, then mode and tenor processing will be based on a fixed instance of the internal scheme.

(Re)modeling the original data via a field mapping has two chief purposes:
1. The data is cast in a form appropriate for direct use while building presentations for a particular class of users.
2. Mediating ([14]), i.e. finding a common representation, if a register needs to access data from several sources, or for register reuse, especially register ontology reuse.

Example 5. As the result of field processing in our running example of mid-term grading reports, we might obtain the following term of class `Student`:

```
Name = "Joe Marek"                CurrentScore = 74.5
CurrentGrade = 'C'                MissingAssignments = 'Hw 2'
Hw[] = 91, 0, 84                  MidTermGrade = 94
```

Mode processing operates on the internal representation of the original data. The two main tasks for mode processing are to: define content by selecting phrases from the vernacular that describe relationships between facts; establish the structure of the final document. Mode processing relies only on data in the instance of the internal scheme, as determined by the register field. The result of mode processing is a *document outline*.

Example 6. Here is a document following our running example:

```
Grade Report : GREETING(Name, Course)
               SCOREEARNS(CurrentScore, CurrentGrade)
Exceptional Circumstance : MISSINGWARNING( MissingAssignments )
Grade Support : SCOREBASE( Hw[], MidTermGrade )
```

The goal of register processing is a document that is consistent in meaning and always user appropriate, but variable in expression, word choice and, when necessary, phrasing and structure. Both document structure and phrase selection are variable in the mode. Variability in phrase expression, synonym substitution for terms, and so on, are handled at the tenor level.

Tenor processing executes a procedure for each phrase and builds the final presentation, for instance an HTML document.

2.3 Register Systems

Our main goal is to automate the process of register development by creating a register system. A register system should provide:
1. Authoring tools supporting definition and maintenance of a register domain and processing descriptions.
2. A register processor to convert the definitions defined in the above system into executable code.

A register system will allow register developers to create reliable document generating software. Consequently, they can focus more on domain issues and less on programming details. It will also help with software maintenance and, in many cases, will allow meta tags carrying semantic information to be added to documents automatically. Finally, such a system supports sharing and reuse of resources.

3 The DEXTER Register System

We will now briefly describe the DEXTER Register System which we are developing. Following a brief description of DEXTER's authoring subsystem, our focus will be on DEXTER's approach to field and mode processing. We will discuss neither DEXTER's register execution engine nor its tenor processing in this paper.

The authoring subsystem consists of a suite of editors specialized to support the register author's various tasks. There is also a library of classes and methods to support the details of presentation. The form of ontology classes definable by the author is restricted. The aggregation hierarchy for these classes must be acyclic and general methods are not allowed. Connectivity to only one database, *Mini SQL* ([17]) is supported, and any database meta data needed is expected to be supplied by the author. Since processing in DEXTER is specified by rules, editors for each phase of processing must support creation of the appropriate rule forms. As all access to the external database occurs during field processing the query editing component is a subcomponent of the field editor. Text phrases in DEXTER are specified by templates, defined by the register author.

3.1 Field in DEXTER

DEXTER's field processing is described as a set of rules. DEXTER assumes null values are legal in either the external or internal scheme. We will use the following notation to describe these rules. A tuple in the external database scheme will be denoted t. An attribute in the external scheme is denoted a_i. Thus $t.a_i$ is the i^{th} coordinate of tuple t. The subset of t attributes known to be non-null is denoted by $\hat{t}$. The internal scheme is noted analogously, with s b_j, and $\hat{s}$ replacing t, a_i and $\hat{t}$ respectively. The s_j are register terms, to which field rules assign value. Terms must be assigned in some order, and $s_{1 \leq i < k}$ denotes the set of terms assigned prior to assignment to $s.b_k$. Whether coordinates of s and t have values or not is necessary but not sufficient information to write rules. We must also be able to define relations on those coordinate values. Thus, $\mathcal{R}$ denotes a relation over $\hat{t}$ and $\hat{s}$, the non-null coordinates of s and t. Finally, E_j are expressions over $\hat{t}$, $\hat{s}$ and constants, evaluating to a type compatible with the $s.b_j$. A general form of a field rule in DEXTER is:

IF $(t.a_1 = \mathit{Null} \wedge \ldots \wedge t.a_k = \mathit{Null} \wedge t.a_{k+1} \neq \mathit{Null} \wedge \ldots \wedge t.a_p \neq \mathit{Null})$
$\wedge\, s.b_1 = \mathit{Null} \wedge \ldots \wedge s.b_l = \mathit{Null} \wedge s.b_{l+1} \neq \mathit{Null} \wedge \ldots \wedge s.b_q \neq \mathit{Null})$
$\wedge\, (\mathcal{R}(\hat{t}, \hat{s}))$
THEN $s.b_{q+1} = E_{q+1}(\hat{t} \cup \hat{s} \cup s_{1 \leq i < q+1})$ $, \ldots,$ $s.b_{q+r} = E_{q+r}(\hat{t} \cup \hat{s} \cup s_{1 \leq i < q+r})$

Note that these are epistemic rules; rule applicability depends on what we do and do not know. Here is a simple example of such a rule:

IF $(t.\mathit{AltGradeOption} = \mathit{Null} \wedge t.\mathit{ACut} \neq \mathit{Null} \wedge t.\mathit{BCut} \neq \mathit{Null})$
$\wedge (s.\mathit{CurrentGrade} = \mathit{Null} \wedge s.\mathit{score} \neq \mathit{Null})$
$\wedge (t.\mathit{BCut} \leq s.\mathit{Score} < t.\mathit{ACut})$
THEN $s.\mathit{CurrentGrade} = \mathit{Bgrade}$

The fact that rules may depend on values assigned to a tuple s earlier in the process makes the order of rule evaluation important.

3.2 Mode in DEXTER

DEXTER breaks mode processing into two distinct steps: *content determination* and *structure instantiation*. Both are managed with a rule based approach. Here is an example of a *DEXTER Content Rule:*

IF $(s.\mathit{MissingAssignments} \neq \mathit{Null})$
THEN *Include MISSINGWARNING*$(s.\mathit{MissingAssignments})$

The above rule can be generalized to epistemic rules of the form:

IF $(s.b_1 = \mathit{Null} \wedge \ldots \wedge s.b_k = \mathit{Null} \wedge s.b_{k+1} \neq \mathit{Null} \wedge \ldots \wedge s.b_n \neq \mathit{Null})$
THEN *Include* $P_1(\sigma_1) \wedge \ldots \wedge P_r(\sigma_r)$

where the $s.b_i$ and $\hat{s}$ are as above, P_j are phrases, and σ_j is an ordered subset of $\hat{s}$ forming an argument list for P_j. Order of evaluation does not matter for these rules.

DEXTER structure rules determine which sections of a document are opened or closed. A section must be opened for writing before a phrase can be expressed in that section. Here is a simple structural rule.

IF $(Open(Body) \land \neg Open(GradeSupport) \land$
$Included(SCOREBASE(X, Y)) \land$
$\neg Included(MISSINGWARNING(X)))$
THEN $Open(GradeSupport) \land Assert(SCOREBASE(X, Y))$

Included(phrase) holds if *phrase* appears in the output of the content rules. Notice that the applicability of this rule depends on absence of information about a grade warning and that these rules are again order dependent.

The general form of a structural rule follows. The Σ_i are document sections. If $Open(\Sigma_i)$ holds then it is legal to place phrases into Σ_i, and to open or close subsections of Σ_i. Further, $P_j(x_j)$ is a phrase P_j with arguments x_j.

IF $(Open(\Sigma_{m+1}) \land \ldots \land Open(\Sigma_n)) \land$
$(\neg Open(\Sigma_{n+1}) \land \ldots \land \neg Open(\Sigma_q)) \land$
$(Included(P_1(x_1)) \land \ldots \land Included(P_l(x_l))) \land$
$(\neg Included(P_{l+1}(x_{l+1})) \land \ldots \land \neg Included(P_r(x_r))))$
THEN $Open(\Sigma_\alpha) \land Assert(\pi_I) \land Close(\sigma_{A,O})$

Note that x_j and x_k may overlap. This means phrases may share variables. For instance, $Included(P_j(x_j))$ is true when the phrase (with associated variables) can be unified with some phrase included by the mode content rules. Thus to satisfy this condition some unification on shared variables (but with no function symbols) must occur. When the rule condition is satisfied then three actions may be taken:
1. A single section, Σ_α, is opened and marked as the current section.
2. Members of a possibly empty subset of included phrases, (π_I), are placed in the current section for later expression.
3. Members of a possibly empty set of sections which were previously open, $\sigma_{A,O}$, are marked as no longer open.

4 Additional Benefits of Register Generated Documents

The principal use for registers is to present data to the user in a way that takes into account the semantics that the user applies to the data. However, there are other applications of registers and register systems. Many have to do with the capacity of these systems to support meta data with very low overhead. To indicate the potential in this area we will briefly discuss the approach to meta data we take in DEXTER, which we call *semantic tags*. (This is distinct from the same term in [2].)

By a *semantic tag* we mean a tag (as in HTML) which, for some part of a document, indicates that part's *content* in the sense of its meaning. Families of registers, controlled by a register system with a common vocabulary, offer advantages in this area. A dictionary of tags used in a register system can be published for appropriate communities. Since the purpose of a register is to systematically transform data into a document representation, registers can add semantic tags to documents automatically. Beyond consistency, the principal benefit of register tagging is that it is a low overhead operation, taking advantage of already necessary work. As semantic tagging via registers is clearly possible, we will now discuss some applications of registers and register systems which make it clear that this form of systematic tagging of content is useful. This implies several directions for further research.

1. *Searching* for patterns with multiple meanings is inefficient. Register generated documents can be automatically and consistently tagged for content with a level of specificity that is prohibitive for hand-written documents. The tagging scheme used in the register system may be published and thus may support search for patterns (tags) with agreed upon meanings.

2. We can say that a database must: store data, allow searching on that data, and allow viewing of search results. Consider a collection of documents that are semantically tagged. Content is represented consistently and hence searchable as above. Tags can guide data expression. Hence, a collection of semantically tagged documents may be viewed as a *virtual database*. Such a virtual database might stand alone, or we might have several such databases on top of a standard database, acting as front ends for several user communities.

3. There are implications for *information synthesis and control*. With untagged documents we can ask the question "Is there a document that describes process X?" With a set of tagged documents we can ask if it is possible, from this document set, to *construct* a document that describes process X. We can ask if our collection makes a complete set of facts publicly available.

4. There is a point at which a document is not yet generated, but its content and structure are determined. This can serve as a form of document *compression*. Assuming that a register can be executed at the client side, documents can be encoded either by the underlying external data, its internal representation or the document outline, whichever is more compact.

5 Related Work

A fundamental problem that the register author must solve is referred to in [7, 6] as the *the writer's problem*. Specifically, how does an author proceed, given some facts, in using background knowledge and communicating those facts with some end in mind? Our current focus is on text presentations. Thus, generating coherent, reliable text is critical. The works of McKeown [13], Paris [5] and Bateman and Paris [1] are related. In DEXTER, however, we take a simpler approach, borrowing from descriptive linguists ([9] among others.) IVORY [3] is a tool developed at Stanford University to assist the physician in writing progress notes. In this case the approach to text generation in a single DEXTER register is similar, but DEXTER's phrases are not hardwired.

The advent of the World Wide Web has resulted in intense scrutiny of automation and authoring support for documents, e.g. [11]; creating documents that are more content accessible via automated means, e.g. [16]. Mapping SGML document type definitions into object oriented database schema is considered in [4]. The problem of analyzing and representing data has been considered, e.g. [12] and the TSIMMIS project [8, 14]. As in TSIMMIS we assume self describing internal representations of data in our systems. Our approaches differ as our goals differ.

6 Conclusions

There is a legitimate need to increase the functional power of systems to support information access. Computational registers as described here support information access by attempting to restore to data some of the semantics expected by particular groups of users in a natural way. Continuing to write *ad hoc* code to accomplish this task is not sufficient. That the task of writing data presentation software is common, and usually achievable, implies that we can understand and should try to automate the process. The end goal is to clarify and support or automate the process of writing registers such that the register author can focus more on meeting user expectation, and less on manipulating background systems. Our system, DEXTER, is a step in that direction.

DEXTER's register architecture allows for substantial register reusability. An ontology developed for the needs of a register to build presentations of medical data can be used for a number of other registers working with possibly different databases. Given such a register, to make it usable with yet another database requires only the development of an appropriate field specification. Similarly, if a modification of a presentation of the same data is required, only the tenor must be changed. Consequently, DEXTER's architecture maximizes reusability in the context of changing requirements.

Acknowledgements

This work was supported by USARO contract DAAH-04-96-1-0398. Helpful discussions with Professor Raphael Finkel and Anthony Borchers are gratefully acknowledged.

References

1. J. A. Bateman and C Paris. Phrasing text in terms a user can understand. In *Proceedings of IJCAI*, 1989.

2. P. Buitelaar. A Lexicon for Underspecified Semantic Tagging. Available via Computation and Language EPrint Archive, http://xxx.lanl.gov/cmp-lg/ as paper 9705011, 1997
3. K.E. Campbell, K. Wickert, L.M. Fagan, and M.A. Musen. A computer-based tool for generation of progress notes. In *Proceedings of the Sixteenth Annual Symposium on Computer Applications in Medical Care*, 1993.
4. V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proceedings of SIGMOD 94*. ACM, 1994.
5. Paris C.L. *The Use of Explicit User Models in Text Generation: Tailoring to a User's Level of Experience*. PhD thesis, Columbia University, 1987. also technical report CUCS-309-87.
6. D. Estival and F. Gayral. A nlp approach to specific types of texts: Car accident reports. In *Proceedings of Workshop on Context in Natural Language Processing IJCAI 95*, 1995.
7. D. Estival and F. Gayral. A study of context(s) in a specific type of texts: Car accident reports. Available via Computation and Language EPrint Archive, http://xxx.lanl.gov/cmp-lg/ as paper 9502032, 1995.
8. H. Garcia-Molina, J. Hammer, Y. Ireland, K. and Papakonstantinou, and J. Ullman. Integrating and accessing heterogeneous information sources in tsimmis. In *Proceedings of the AAAI Symposium on Information Gathering*, 1995.
9. M. Gregory and S. Carroll. *Language and Situation: Language Varieties and Their Social Contexts*. Routledge And Keegan Paul, Ltd., 1978.
10. E. van Herwijen. *Practical SGML*. Kluwer Academic Publishers, 1990.
11. K. Jones. Tops on-line – automating the construction and maintenance of html pages. In *Electronic Proceedings of Second World Wide Web Conference 94: Mosaic and the Web*, 1994. Electronic Publication: http://www.ncsa.uicu.edu/SDG/IT94/Proceedings/Autools/jones/jone.html.
12. R.E. Kent and C. Neuss. Creating a web analysis and visualization environment. In *Electronic Proceedings of Second World Wide Web Conference 94: Mosaic and the Web*, 1994. http://www.ncsa.uicu.edu/SDG/IT94/Proceedings/Autools/kent/kent.html.
13. Kathleen R. McKeown. *Text Generation*. Cambridge University Press, 1985.
14. Y. Papakonstaninou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings ICDE 96*, 1996. available via ftp://www-db.stanford.edu/pub/papkonstantinou/1995/medmaker.ps.
15. D. Raggett. HTML 3.2 reference specification. Electronic Publication: http://www.w3.org/pub/WWW/TR/REC-html32.html, 1997.
16. C.M. Sperberg-McQueen and R.F. Goldstein. Html to the max a manifesto for adding sgml intelligence to the world-wide web. In *Electronic Proceedings of Second World Wide Web Conference 94: Mosaic and the Web*, 1994. Electronic Publication: http://www.ncsa.uicu.edu/SDG/IT94/Proceedings/Autools/sperberg-mcqueen/sperberg.html.
17. Hughes Technologies. Hughes technologies library. Published Electronically at http://Hughes.com.au/library/. Contains documentation on miniSQL.

This article was processed using the LaTeX macro package with LLNCS style