January 2000

Bundles in captivity : an application of superimposed information

Delcambre. Lois

David Maier

Shawn Bowers

Longxing Deng

Matthew Weaver

See next page for additional authors

Author
Delcambre. Lois, David Maier, Shawn Bowers, Longxing Deng, Matthew Weaver, Paul Gorman, Joan Ash, Mary Lavvelle, and Jason A. Lyman

Bundles in Captivity: An Application of Superimposed Information[1]

by
Lois Delcambre, David Maier, Shawn Bowers, Longxing Deng, Mathew Weaver
Computer Science and Engineering Department
Oregon Graduate Institute
{lmd, maier, shawn, longxing, mweaver}@cse.ogi.edu
and
Paul Gorman, Joan Ash, Mary Lavelle, Jason A. Lyman
Medical Informatics and Outcomes Research
Oregon Health Sciences University
{gormanp, ash, lavellem, lymanja}@ohsu.edu

Abstract

It is a common human trait to gather bits of information in one place, and group, classify, and otherwise annotate them, such as on the back of an envelope or in the margins of a paper. Such "bundles" of information appear to serve several different purposes, such as creating and maintaining situation awareness, problem solving, and aiding communication. These bundles are often structured via mechanisms such as juxtaposition of items, headings, and nesting. Bundles often repeat or reference information that exists elsewhere, but can contain new information as well.

Our observational work in the medical domain confirms that bundles are widely used and reused. We focus in this paper on the challenges of building generic technology that supports the creation of structured, digital bundles, with an emphasis on keeping references that appear in the bundle linked to the referenced information in its original source. Bundles are one form of *superimposed information*, information placed over a base layer of data. To explore issues associated with bundles, and more generally superimposed information, we have implemented SLIMPad, the Superimposed Layer Information Manager scratchPad, an application for creating structured, digital bundles.

1 Introduction: What we've observed

In field observations of expert clinicians caring for patients in critical care units, bundles appear to be a widely used means of managing information to support diverse, complex, often simultaneous tasks. A bundle, for this work, is any sort of information collected and structured by the clinician during problem solving. A *bundle* can be created on the back of an envelope, on a blank sheet of paper, or on a form of sorts with headings and groupings. Regardless of the media, bundles are free form; both the content and the structure are created by and for the clinician. Figure 1 shows several examples of *bundles in the wild*, i.e., bundles that we have observed in our work [8]. We see two bundles that have been written on the most available scrap of paper (an unopened bandage package on the left and a paper towel on the right). On the left side of Figure 1 we see (underneath the bandage package) a more structured bundle called a *flowsheet*, where the status of an intensive care patient is tracked over time.

[1] This work is supported, in part, by the National Science Foundation, Grant Number II-98-17492.

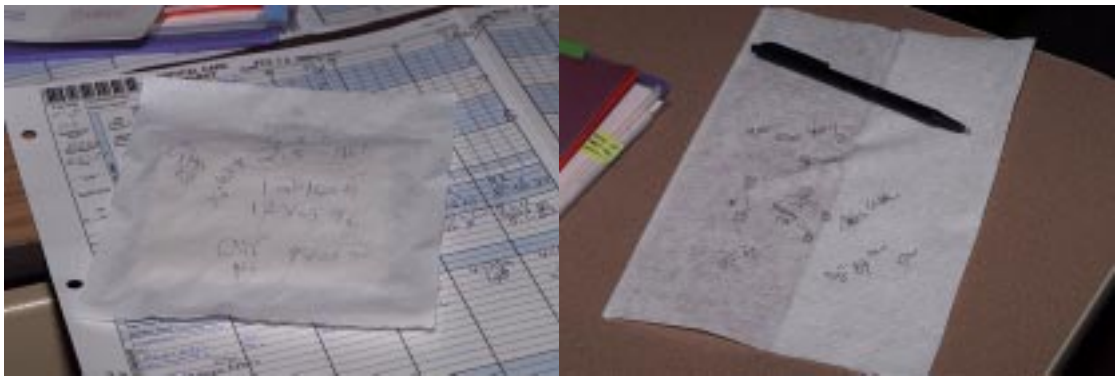

Figure 1: Selected bundles from the Intensive Care Unit.

Bundles may be especially useful in these settings characterized by high uncertainty, low predictability and potentially grave outcomes, where time and especially attention are highly constrained, and where interdisciplinary teamwork is essential. Reports of observations from other, analogous domains such as air traffic control suggest that bundle use may be common, outside the medical area [9, 10, 12, 18].

Bundles facilitate the selection, collection, organization, and sharing of information. They enable individuals or groups to create and maintain a coherent representation of the current state and expected future states. There is benefit in creating them (through the active processing of information to improve understanding), in reusing them (by triggering memory) and in sharing them to establish collectively maintained, situated awareness. Bundles give meaning to information by organizing it in a context-specific, problem-focused way. Selection alone adds value by excluding information that's not considered important or relevant to the current context.

Computer-based tools for creating and managing bundles may be useful as the information in these settings is increasingly represented in electronic information systems. We report here on our first prototype application for creating bundles, specifically digital bundles that are manipulated on a computer screen. The remainder of the paper discusses the more general context of our work on digital bundles as an example of superimposed information. We note that our prototype was not designed specifically for the medical domain, though it has influenced our continuing development.

2 SLIMPad: What we've built

The Superimposed Layer Information Manager scratchPad (SLIMPad) allows users to create structured, digital bundles. SLIMPad is motivated by our observational work indicating that structured bundles are widely used and reused. Although SLIMPad is not necessarily intended to support any specific, observed medical task, it has allowed us to explore technology for superimposed information in parallel with the observational work.

Many bundles in the wild have a scratchpad-like feel and appearance. SLIMPad provides this same scratchpad look and feel, in a computerized tool. The data model we used to implement SLIMPad is shown in Figure 2, represented using the Unified Modeling Language (UML) [17].

The data model consists of four main entities with several relationships. The top-level object is a *SLIMPad*, which serves as a container for all of the other objects. The SLIMPad is an *AbstractBundle* and thus can contain any number of Scraps and any number of Bundles. In a similar way, each *Bundle* is an AbstractBundle and can contain any number of Scraps or Bundles. A *Scrap* object (i.e., a single piece of information) may or may not contain a Mark object – and appears either directly in the SLIMPad or contained in a Bundle. The Mark object contains a

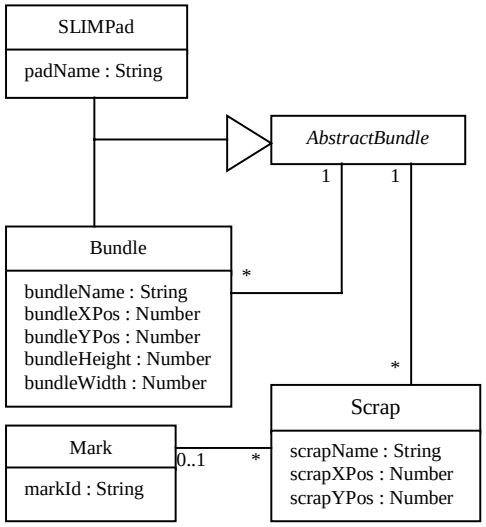

Figure 2: Structured Bundle Model for SLIMPad.

mark identifier, which is used to reference an information element in the base layer (i.e., the original information source).

The SLIMPad application provides an interface between the user and data in this data model. Each visual entity the user sees on the screen corresponds to an object in an instance of the data model. Figure 3 shows the SLIMPad application being used in a medical setting. The larger window, titled 'Rounds', is the visual representation of a SLIMPad object. In this scenario, the user has created a bundle, titled 'John Smith'. The bundle contains three scraps and another bundle. The top two scraps represent medications the patient has received. The mark contained in each scrap references the corresponding medication in a complete medication list (shown in a Microsoft Excel document). By double clicking on the scrap, the mark is de-referenced and the original information source, the medication list, is displayed with the appropriate medication highlighted, as shown in the upper right of Figure 3.

The 'Electrolyte' bundle contains a set of scraps that come from a lab report – in this case, formatted as an Extensible Markup Language (XML) document. Each of these scraps can be double-clicked, which opens the report and highlights the appropriate section of the XML document referenced by the scrap.

To create a new scrap on the SLIMPad, the user first selects an information element from a base-layer application (such as Excel or an XML viewer). The base-layer application must support the creation of marks, which act as a reference to base-layer information. Once the user has created a mark, it can be placed onto the SLIMPad, which creates a scrap that can be named and moved within the SLIMPad (e.g., strategically placed next to related scraps or placed in an appropriately titled bundle). By creating the mark and attaching it to a scrap, the user creates a digital "sticky-note," which comes with a digital "wire" that always leads back to the information in the original data source, as referenced by the mark. It is important to note that the user defines the physical layout of the scraps and the bundles. The scraps and bundles seen on the screen are a representation of the user's conceptual model. For example, each number in the 'Electrolyte' bundle has a specific meaning to a medical professional, which can quickly be deduced by their arrangement relative to each other. The SLIMPad data model does not impose structure – but allows the user to create structure.

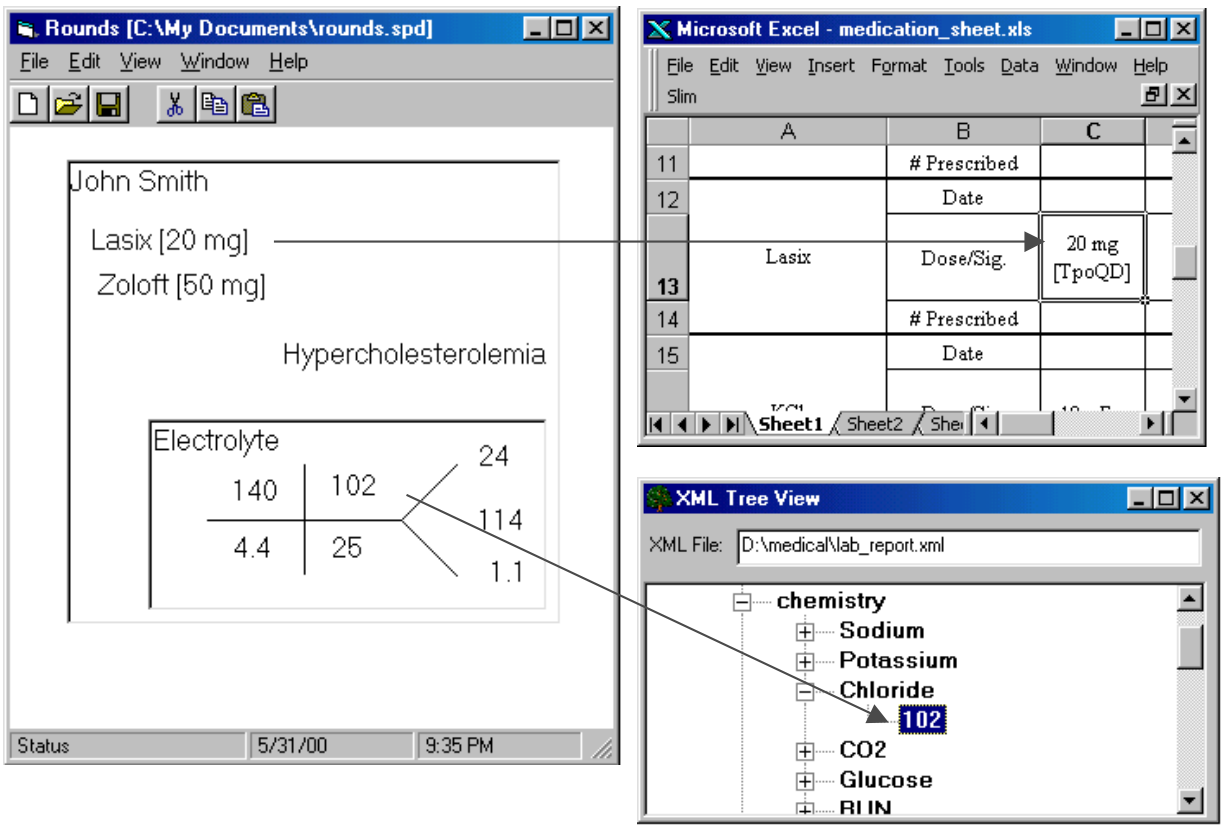

Figure 3: SLIMPad Screenshot showing two marks: one to Excel and one to XML.

SLIMPad provides building blocks so the user can digitally construct his or her own conceptual model, while at the same time keeping the user only a click away from the underlying data. The SLIMPad application is our first application that supports digital bundles; SLIMPad uses a relatively simple model of nested, digital bundles containing scraps. We are also investigating the notion of digital bundles in a more general context where the model may be more complex, e.g., with more structure, with multiple marks per scrap, and with explicit linkages between scraps.

3 Digital Bundles: What issues we considered

In our first SLIMPad implementation, we included the following key features for digital bundles:

- Users can select precisely which scraps they want to include on a pad or in a bundle.
- Users can label scraps however they like.
- Users can collect related scraps into a bundle.
- Users can label bundles however they like.
- Users can nest bundles inside other bundles.
- Users can put additional information, e.g., annotation or a “to do” list, in a bundle.

Digital bundles can be compared to other technologies including views, collections, forms, and workflows, but we believe there are key distinctions in each case.

A digital bundle is not a view because it might *contain less information* or it might *contain more information* than a view defined on an underlying data source. One could define a view over a

patient medical record and list all the known problems for the patient. However, a clinician may select three items from the problem list for a patient in an Intensive Care Unit (ICU) when, in fact, there are seven known problems. On the other hand, a bundle may also contain arbitrary information, such as annotation, highlighting, or structuring, which is not deducible from the underlying data.

In a similar way, **a digital bundle is not an index** because the set of scraps need not be complete and because the bundle may contain additional information. The scraps in a bundle serve as an index to the corresponding information elements in the original information source but it's a specialized form of index.

A digital bundle provides a way to group marks and supplemental information but **a digital bundle is not simply a collection of documents**. Each scrap in a bundle contains one mark; each mark references an information element in an underlying information source. But a mark may reference information at various levels of granularity,[2] e.g., a whole element or a single attribute value in an XML document, or one cell or a contiguous block of cells in an Excel spreadsheet. Moreover, the mark references the information *in situ*, without extracting it or copying it. A digital bundle, by its very nature, contains a collection of scraps but it is not a conventional collection of documents.

In our observational work, we observe "bundles in the wild." Whenever someone gathers together a bit of information, perhaps with groupings or headings, we call that a bundle. Some of the bundles we have observed are quite elaborate. They can be highly structured and are sometimes written on preprinted sheets, and their creation and maintenance has been institutionalized. But **a digital bundle is not a form**. The primary difference is in the prescriptive vs. proscriptive dimension. A form is rigid; it sets forth the fields that can be entered into the system. A digital bundle, on the other hand, is essentially free form. Structure can be introduced, fields can be named and possibly even typed, but the user is never limited to the currently available structure. Some of the bundles that we observed in use by clinicians look like preprinted forms, with fields, headers, and divided areas. But some areas, such as in the ICU flow sheet, are intentionally multi-purpose. This flexibility in structure, including the possibility of no structure, is a dominant theme of our research. We are exploring schema-optional and schema-later approaches for managing information in bundles. We seek to exploit structure when it's present but not require it, unlike traditional database management systems.

Finally, **a digital bundle is not a workflow**. The information present in a digital bundle, especially observed over time, may be the result of the execution of a workflow. But it need not be. For example, certain information might be placed into a bundle just before a physician begins rounds. As each patient is examined, certain information may be annotated or highlighted and new information may appear, e.g., as a reminder to order new medications or lab tests. However, the bundle doesn't dictate the order in which information is gathered, where it is gathered, nor even if it has to be gathered. We are currently not working on the specification of a workflow or on the problem of linking workflows to the contents of a bundle.

4 Superimposed Information: What we're researching

Bundles and scraps are one example of *superimposed information* [6, 13]. In general, superimposed information is used to organize, highlight, supplement, connect, and reuse select information from pre-existing information sources. With SLIMPad, users can organize and

[2] The granularity of a mark is limited to the granularity supported by the various addressing schemes used in the marks supported on different sources.

annotate marks to information sources through bundles and scraps. We define superimposed information as a layer of data (the *superimposed layer*) placed over existing information sources (the *base layer*) with the following characteristics (see Figure 4):

- It can contain additional data, not included in the information sources.
- It can have varying degrees of structure.
- It does not modify the existing information sources.
- It contains marks, which connect superimposed information to information sources.

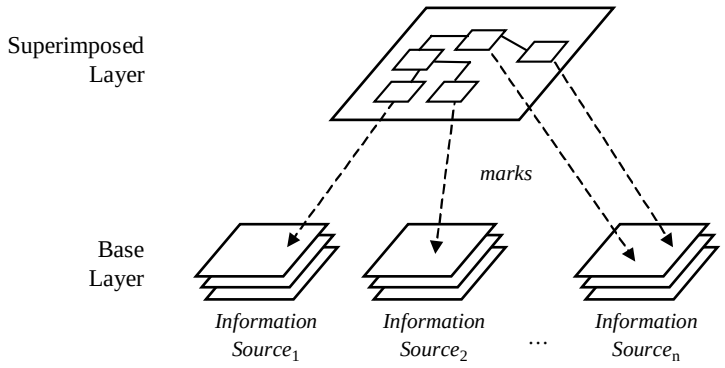

Figure 4: The superimposed and base layers with marks.

SLIMPad uses a specific model of superimposed information, the SLIMPad-bundle-scrap model. But, there are a number of other, general-purpose superimposed models gaining in popularity including Topic Maps [1], the Resource Description Framework (RDF) [11], and XML [3] when used with an addressing scheme such as XLink [7]. An important property of these models is the use of schema to optionally type superimposed information. Figure 5 shows an example of the superimposed layer for model-based superimposed information, where the superimposed layer consists of the application's superimposed model, schema information (which is possibly optional), and the superimposed data, termed *instances*.

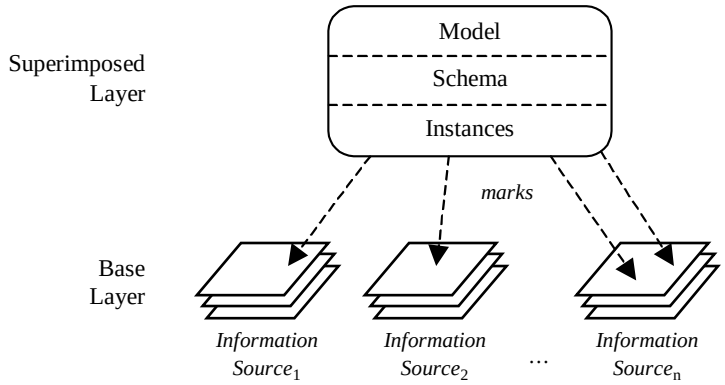

Figure 5: Model, schema, and instances as the superimposed layer.

In general, we are interested in model-based superimposed information because tools that use specific models of information are broadly applicable. For example, CASE tools, SQL, and even spreadsheet tools each exploit a fixed model of information and are applicable across diverse data that conform to their model. We see SLIMPad as such a tool, which leverages a superimposed model and can be used universally for information that conforms to the SLIMPad-bundle-scrap model.

Our goal in researching superimposed information is twofold. We want to provide superimposed applications such as SLIMPad with generic tools: (1) to perform mark management including the ability to mark heterogeneous information sources; and (2) to represent and manage model-based superimposed information.

There are a number of examples of superimposed applications, including Multivalent Documents (MVD) [15, 16], CoNote [5], and Third Voice [20], which are designed to let users add superimposed information to Web pages and digital documents using annotations. Other, more general examples of the use of superimposed information include pages of related links, bookmark lists, and Web indexes such as the Open Directory [14]. Although the Open Directory uses metadata to provide additional structure to information sources, metadata is generally not considered superimposed information since it is typically embedded within the documents of the base layer. Also, it is important to note that superimposed information and superimposed applications are not limitted to the Web, SLIMPad being one such example.

5 Generic Technology for Superimposed Information: How We Built It

Figure 6 shows the architecture we use to provide generic technology to support the management of superimposed information. A superimposed application (e.g., SLIMPad) interacts with both base-layer applications and the generic management components, which manage marks to base-layer data as well as the application's superimposed information (e.g., bundles and scraps). Superimposed applications can use one or more base applications through the *mark management* component, which allows the superimposed application to access and resolve marks. Base applications use mark management to create marks. A mark contains an address that is understood by the base application. For example, with Microsoft Excel, a mark can represent a spreadsheet cell through its row and column position, which can be used to access the information element within the cell. Further, the base application knows how to render or extract the information elements referenced by marks.

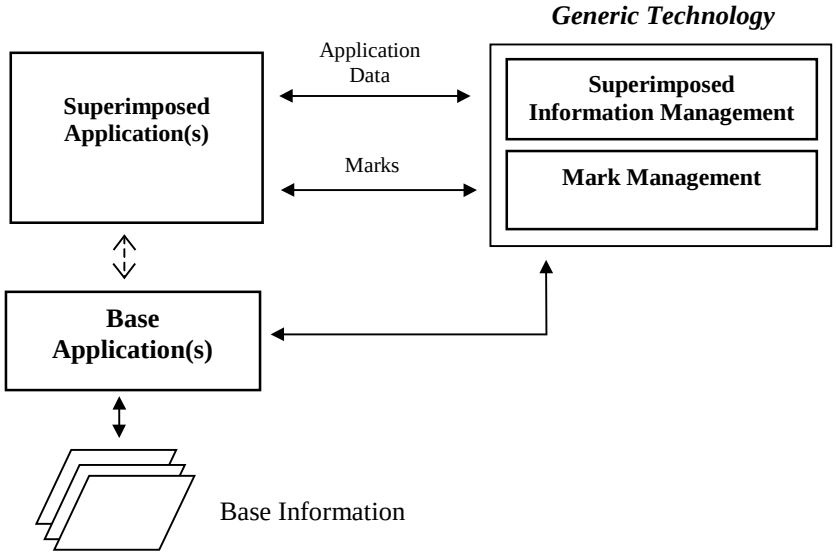

Figure 6: Overview of our Architecture for Superimposed Applications.

There are three basic viewing styles for superimposed applications: simultaneous viewing, enhanced base-layer viewing, and independent viewing as shown in Figure 7.

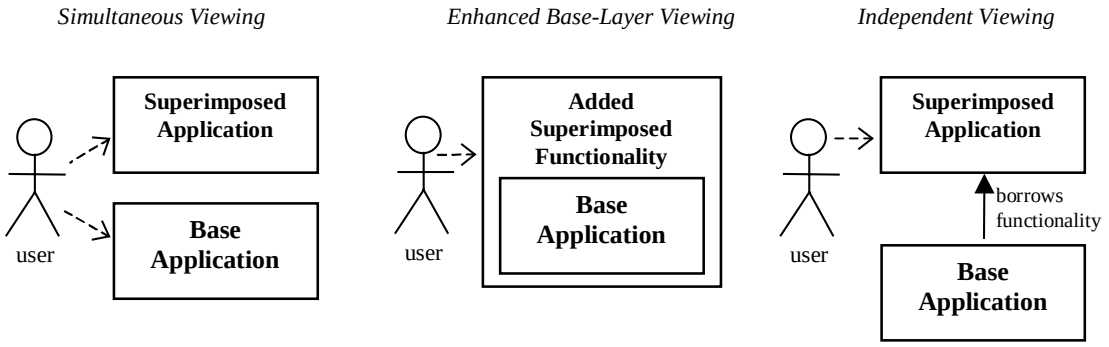

Figure 7: Three viewing styles for superimposed applications.

With *simultaneous viewing*, a user accesses a superimposed- and base-layer application at the same time. Usually, there are two windows active on the computer screen: one for the superimposed application and one for the base application. A user interacts with either window as desired.

With *enhanced base-layer viewing*, the functionality of a base application is enhanced to manage superimposed information. Third Voice is such an example, which enhances Microsoft's Internet Explorer and Netscape's Navigator by allowing the user to create and view annotations in the same browser window as the Web page.

With *independent viewing*, the base application is hidden from users. A user sees only the superimposed application, which exposes the functionality of the base application, usually in a limited way. The base application can work in the background to extract base information elements for the superimposed layer, or it can work as an in-place viewer for base information (i.e., part of the base application is embedded within the superimposed application).

SLIMPad is most often used with simultaneous viewing, but it can also use independent viewing. In most cases, the user juxtaposes the SLIMPad window and a base document viewer to interact with base and superimposed information simultaneously. Independent viewing is used when marks on the SLIMPad support in-place viewing, e.g., to read a range of spreadsheet cells in the SLIMPad without launching Excel. In-place viewing can be effective when the user does not need full spreadsheet functionality to manipulate the marked information because Excel is a resource-consuming application that is slow to launch.

Figure 8 shows how SLIMPad fits into the architecture of Figure 6. Currently, five kinds of base information types are supported: XML documents, Microsoft PowerPoint files, Excel spreadsheets, Web pages and Adobe PDF files. Each kind of base information uses a specific type of mark, and each type of mark can access information at different granularities. For example, a PowerPoint mark can reference not only individual pages (i.e., slides), but also individual graphics and text within a page, whereas the finest granularity supported for PDF marks are at the page level. Both the format of base information and the capabilities of the base application influence the granularity of marks.

5.1 The Mark Management Component

Mark management hides the details of the different kinds of marks from the superimposed application. From the superimposed application's viewpoint, a base information element is addressed by a mark, which can be of any type (e.g., an XML or PowerPoint mark). This transparency greatly simplifies the development of superimposed applications. The architecture

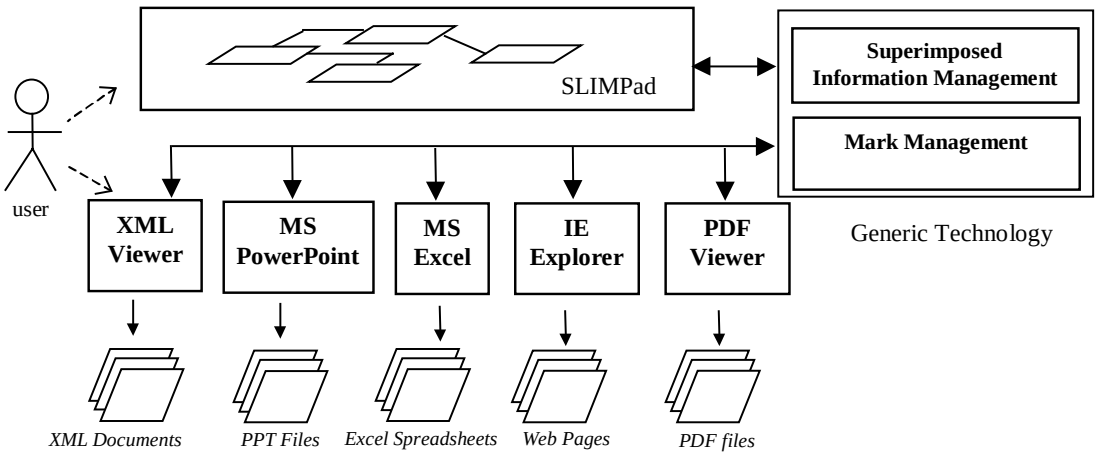

Figure 8: The SLIMPad Architecture.

is also easy to extend; new kinds of base information can be introduced into the system without disturbing existing superimposed applications.

In the current implementation, a mark is represented as a Java object, called a *mark object*. Each mark object has a unique identifier (Mark-ID). The main content of a mark object is an address. Mark objects know their own type, know how to find the correct base information elements, and know how to invoke corresponding base applications for rendering or extraction.

Figure 9 shows the general architecture of mark management (within the SLIMPad implementation). The *mark manager* is the kernel of mark management; it creates, stores, and retrieves mark objects with unique Mark-IDs. The *mark database* is the repository of mark objects. A *mark module* is used to handle the addressing scheme used for a particular type of mark. In order to support new kinds of base information, new mark modules need to be introduced. The base application must be enhanced to support the creation of marks and react appropriately when asked to resolve a mark by its corresponding mark module.

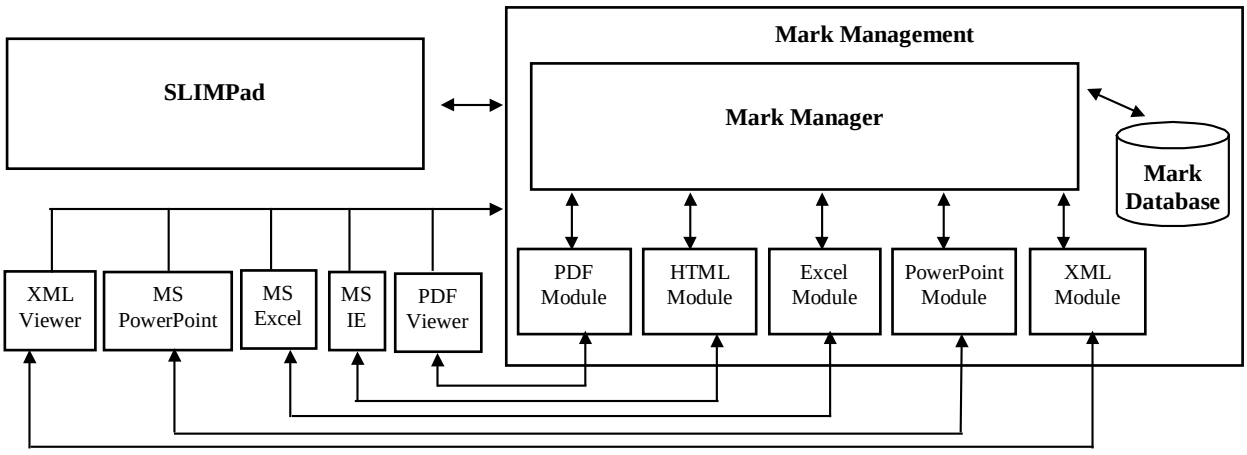

Figure 9: Architecture for Mark Management (as implemented in SLIMPad).

5.2 The Superimposed Information Management Component

Along with managing marks, we are also interested in providing generic technology to manage the superimposed information used by superimposed applications. Figure 10 shows our general strategy for providing superimposed information management. To build a superimposed application within our framework, the application designer first defines a superimposed model or model-schema combination. Based on the fixed model or model-schema, we automatically generate [19] a set of application-specific programming interfaces (APIs) for the application. These specialized APIs let the superimposed application manage application data, which represents superimposed information for the application.

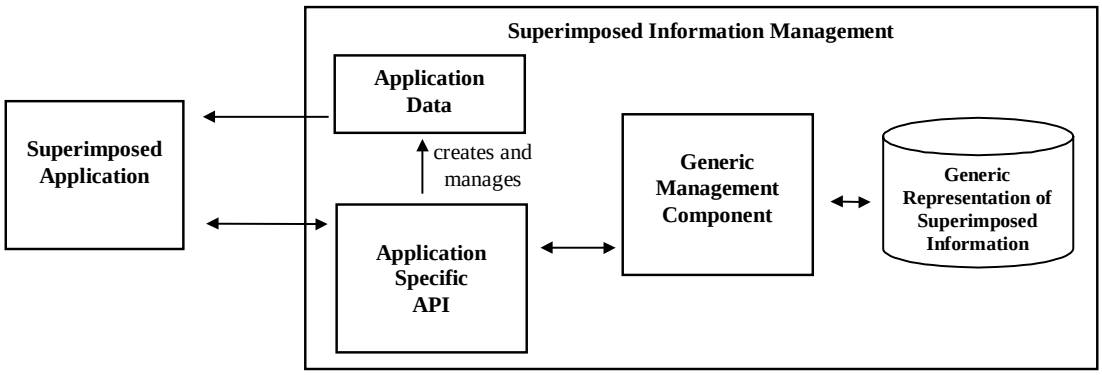

Figure 10: The general architecture for managing superimposed information.

An application-specific API has two roles. First, it is used by the superimposed application for general management functions such as creation, update, removal, query, and storage of application data (i.e., superimposed information). The second role is to maintain consistency between the application data and the underlying representation of the superimposed information. While the specialized API and application data are used for a particular application's model or model-schema combination, the underlying representation is generic for any superimposed model and schema.

The generic representation uses a metamodel for superimposed information [2] that allows multiple superimposed models to be defined in a standard way. The metamodel is defined using RDF Schema [4], which allows superimposed models, schemas, and instances to be represented as RDF data. All three levels (model, schema, and instance) are represented uniformly as RDF triples [2].

Figure 11 shows how SLIMPad uses our superimposed information management architecture. Application data is represented as ActiveX objects written in Java. SLIMPad uses the *SLIMPad API*, also a Java ActiveX object, to create and manage the application data. The SLIMPad API maintains consistency between the application data and the generic representation with the *TRIM Store* (Triple Manager). The TRIM Store provides basic management capabilities such as creation, update, removal, query, and retrieval over the RDF triples.

There are a number of advantages to our approach for managing superimposed information. First, by defining a generic representation, applications can store superimposed information using their desired model instead of a common, fixed data model such as those used in traditional database management systems (e.g., the relational or object-oriented model). With TRIM Store, however, we can still provide general management capabilities over the representation. Additionally, for a particular model-based application such as SLIMPad, we provide specialized management capabilities and access to superimposed information based on the application's

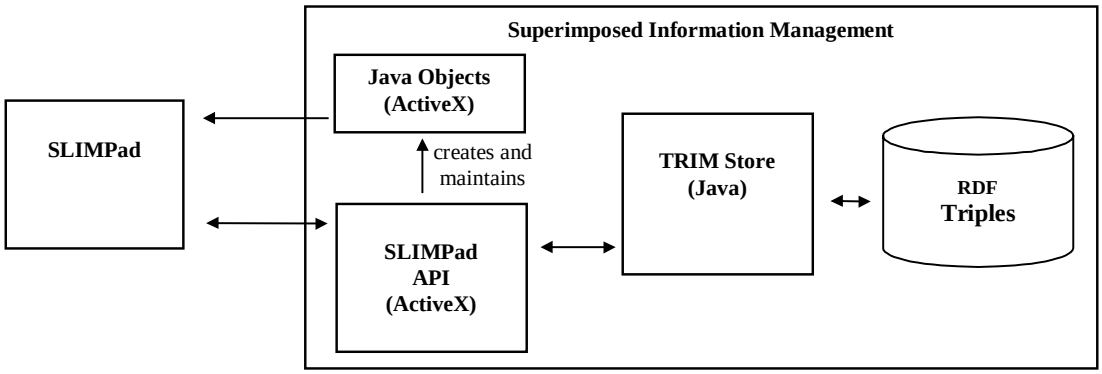

Figure 11: The architecture for managing SLIMPad's superimposed information.

desired model or model-schema. These specialized interfaces can be automatically generated for the application and allow the application to access superimposed information in a convenient way (e.g., as objects in an object-oriented application). Besides ActiveX objects, we have also represented application data as relational database views over the RDF triples. In this way, we are able to leverage existing database functionality to support the management of superimposed information, where the relational database acts as both the specialized API and TRIM Store. Notice that in general, applications will not want the overhead required by a relational database management system.

Another advantage of our architecture is that it supports application interoperability through a common representation of superimposed information. That is, using the generic representation, applications can share and reuse superimposed information. We have looked at using mapping rules between superimposed layers to transform and convert superimposed information from one application into valid superimposed information of another. With this approach, users can manipulate superimposed information by leveraging one application, and then convert the superimposed information into another tool to exploit the new tool's capabilities.

6 Open Issues and Work in Progress: What we're doing next

Our planned future work falls into three major areas: extension and refinement of the SLIMPad-bundle-scrap model, issues in modeling and architecture for superimposed information in general, and producing and testing a SLIMPad-like tool to help with specific clinician tasks.

Some issues we are interested in that concern modeling bundles include:

- **"Schema" for bundles**: As embodied in SLIMPad, the model is fixed, consisting of bundles and scraps, with bundles built up from scratch. It is clear from our observations, however, that people or groups often create multiple bundles with common patterns. We would like to support such patterns with *templates* that predefine certain bundles and labels, and possibly predefine scraps. However, templates should be less restrictive than a normal database schema, in that users should be allowed to extend or modify the predefined collection of bundles in a template. On the other hand, we would like bundles to be more than just stationary (as in a word processor). When a bundle originally comes from a template, we want to remember that fact, to identify common semantics, e.g., to assist with merging the information from two SLIMPads.
- **Sharing**: As realized in our prototype, the bundles on each SLIMPad are disjoint (though scraps on different pads can contain the same mark), and are used by one person at a time. There are several notions of "sharing" we would like to add to this model, such as a bundle

that belongs to multiple pads, a pad with multiple simultaneous users, and interchange of pads and bundles between users. One issue that these various kinds of sharing bring up is the "scoping" of base layers and marks over those layers. For example, one can currently mark into a local file. If a scrap containing such a mark is sent to a SLIMPad tool on another machine, how should that mark be handled? We probably don't want to interpret it as a mark into a same-named file on the new machine. Should it just be flagged as "out of scope" if an application tries to follow it? Or should such an access try to copy or remotely access the original file? Note that if a scrap has a mark that is currently not resolvable, we shouldn't necessarily discard or void the mark because the scrap might eventually find its way back to a context where it is resolvable. In any case, we need a means to explicitly record the scope for base layers and marks, so that users know whether marks they create are sharable and to what extent. The scope can also tell us when a mark is in the proper context for access.

There are several general issues for modeling and management of superimposed information that have emerged from this work and other efforts.

- **Distribution**: In our current SLIM architecture, a single mark manager can service multiple instances of a superimposed application, or even multiple superimposed applications. However, it's obvious that having a single mark manager won't scale, and there are many decisions to be worked out in implementing distributed mark management. Do multiple mark managers support a single space of marks, or must applications keep track of which manager handles a particular mark? Can a mark obtained from one manager be presented to another manager for resolution? Are there possibly different styles of marks: some that require the context of a particular manager to interpret, and others that are manager-independent?

- **Mark Maintenance**: The medical domain has the advantage that much of the base information takes the form of permanent records, reports or reference material, hence base documents don't change much. However, there are instances where they do, and we need to deal with marks into documents that have been updated. The worst case would be to blithely interpret the mark over an updated document, and potentially access an element unrelated to the originally marked element. Better would be to detect that a mark might be invalidated by an update. Better still would be a mark that holds additional information on how to identify a marked element, so that a mark could be "reattached" after update, such as in the MVD system [15, 16].

- **New Kinds of Sources and Marks**: We can currently mark base documents in XML, PDF, Excel, PowerPoint, and HTML. However, we are not totally satisfied with the granularity we can select in each of these sources. Sometimes it is only at the page level (PDF for example). However, to get finer granularity may require constructing our own viewing applications for such documents (which we have done for XML). There are other sources, such as databases, we would like to mark into. A particular challenge comes when the data is encapsulated in a monolithic application, as is the case with some Electronic Medical Record systems. We have found it quite difficult to obtain information form such applications on what the current selection is, and to drive such an application back to a particular selection. We are also interested in marks into a "base" layer that is actually superimposed information for another application.

- **Application Interface**: The SLIMPad uses one particular style of interaction with the superimposed information and mark management components, but many others are obviously possible. For example, the SLIMPad currently sees application data as Java objects, but a different application (or an enhanced SLIMPad) might want a more query-like API. An interesting problem is coming up with API generator capabilities where the superimposed information model, the particular schema and the interface style are all configurable. In terms

of marks, we recognize that the application from which a mark was obtained need not be the one used when that mark is accessed. For example, there could be additional applications for viewing mark contents, or widgets for displaying contents in place. We need a discipline for registering the different pieces of software that work with different mark types, along with their capabilities, so that the information doesn't have to be hard coded into an application. Such a framework would also make superimposed applications more robust in different contexts, not depending on particular programs to be present to be able to operate.

- **Superimposed Information Conversion**: The SLIMPad-bundle-scrap model is just one example of a superimposed information model (Topic Maps and RDF are others). Different applications could obviously want to see their superimposed information in different models. Our current metamodel approach to superimposed information management allows the possibility of simultaneously presenting the same superimposed information under different models, without explicit copying or conversion.

Finally, we reiterate that SLIMPad was not originally designed to be a clinicians tool (though our development direction has been influenced by "bundles in the wild" observations). We plan to produce a bundle-manager tool directed at a particular task or class of use, and test it with actual clinicians (though probably not initially in a clinical setting).

7 References

[1] Michel Biezunski, Martin Bryan, and Steve Newcomb, editors. ISO/IEC 13250, *Topic Maps*, URL:http://www.ornl.gov/sgml/sc34/document/0058.htm.

[2] Shawn Bowers. A generic approach for representing model-based superimposed information. Technical Report, Oregon Graduate Institute of Science and Technology, Number CSE-00-008, May 1, 2000.

[3] Tim Bray, Jean Paoli, and C.M. Sperger-McQueen, editors. Extensible Markup Language (XML) 1.0, W3C Recommendation 10-February-1998, URL:http://www.w3.org/TR/REC-xml.

[4] Dan Brickley and R.V. Guha, editors. Resource Description Framework Schema (RDFS), W3C Proposed Recommendation 03 March 1999, URL:http://www.w3.org/TR/PR-rdf-schema/.

[5] James Davis and Daniel Huttenlocher. Shared annotation for cooperative learning. *Proceeding of the Computer Support for Cooperative Learning Conference*, Bloomington, Indiana, October 17-20, 1995.

[6] Lois Delcambre and David Maier. Models for superimposed information. *Advances in Conceptual Modeling ER '99*, Lecture Notes in Computer Science Volume 1727, pages 264-280, Paris, France, November 15-18, 1999.

[7] Steve DeRose, Eve Maler, David Orchard, and Ben Trafford, editors. XML Linking Language (XLINK), W3C Working Draft 21-February-2000, URL: http://www.w3.org/TR/2000/WD-xlink-20000221.

[8] P.N. Gorman, J.S. Ash, M. Lavelle, J. Lyman, L. Delcambre, and D. Maier. Bundles in the wild: tools for managing information to solve problems and maintain situation awareness. Oregon Health Sciences University, Portland, Oregon, June, 2000.

[9] Edwin Hutchins. *Cognition in the wild*. MIT Press, 1995.

[10] E. Hutchins and T. Klausen. Distributed cognition in an airline cockpit. In Y. Engestrom and D. Middleton, editors, *Cognition and communication at work*, New York: Cambridge University Press, pages 15-34, 1996.

[11] Ora Lassila and Ralph R. Swick, editors. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, URL:http://www.w3.org/TR/REC-rdf-syntax.

[12] W.E. Mackay. Is paper safer? The role of flight strips in air traffic control. *ACM Transactions on Computer-Human Interaction* 6(4), pages 311-340, 1999.

[13] David Maier and Lois Declambre. Superimposed information for the Internet. *ACM SIGMOD Workshop on The Web and Databases WebDB'99*, pages 1-9, Philadelphia, Pennsylvania, June 3-4, 1999.

[14] The Mozilla Open Directory. URL:http://dmoz.org.

[15] Thomas Phelps and Robert Wilensky. Multivalent annotations. *Research and Advanced Technology for Digital Libraries* (ECDL '97), Lecture Notes in Computer Science Volume 1324, pages 287-303, Pisa, Italy, September 1-3, 1997.

[16] Thomas Phelps and Robert Wilensky. Robust intra-document locations. *Proceedings of the 9th World Wide Web Conference*, Amsterdam, May 15-19, 2000.

[17] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley, 1999.

[18] T. von Thaden. Social informatics and aviation technology. *Bulletin of the American Society for Information Science* 26(3), pages 13-14, 2000.

[19] Mat Weaver. SlimML. Term project report for CSE 511. Oregon Graduate Institute of Science and Technology, March 2000.

[20] A. Zeichick. Third voice: extending the interpersonal communications potential of the world wide web. White Paper, URL:http://www.thirdvoice.com/about/WhitePaperPage1.htm.