Putting Integrated Information in Context: Superimposing Conceptual Models with SPARCE

[1]Sudarshan Murthy, [1]David Maier, [1]Lois Delcambre, [2]Shawn Bowers

[1]Department of Computer Science and Engineering
OGI School of Science & Engineering at OHSU
20000 NW Walker Road, Beaverton, OR 97006 USA

[2]San Diego Supercomputer Center
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093 USA

{smurthy, maier, lmd}@cse.ogi.edu bowers@sdsc.edu

Abstract

A person working with diverse information sources—with possibly different formats and information models—may recognize and wish to express conceptual structures that are not explicitly present in those sources. Rather than replicate the portions of interest and recast them into a single, combined data source, we leave *base information* where it is and *superimpose* a conceptual model that is appropriate to the task at hand. This superimposed model can be distinct from the model(s) employed by the sources in the base layer.

An application that superimposes a new conceptual model over diverse sources, with varying capabilities, needs to accommodate the various types of information and differing access protocols for the base information sources. The *Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE)* defines a collection of architectural abstractions, placed between superimposed and base applications, to demarcate and revisit information elements inside base sources and provide access to content and context for elements inside these sources. SPARCE accommodates new base information types without altering existing superimposed applications. In this paper, we briefly introduce several superimposed applications that we have built, and describe the conceptual model each superimposes. We then focus on the use of context in superimposed applications. We describe how SPARCE supports context and excerpts. We demonstrate how SPARCE facilitates building superimposed applications by describing its use in building our two, quite diverse applications.

Keywords: Conceptual modelling, superimposed information, software architecture, excerpts, context, SPARCE.

1 Introduction

When a physician prepares for rounds in a hospital intensive care unit, she often creates a quick synopsis of important problems, with relevant lab tests or observations, for each patient, as shown in Figure 1. The information is largely copied from elsewhere, e.g., from the patient medical record, or the laboratory system. Although the underlying data sources use various information structures, including dictated free text, tabular results and formatted reports, the physician may organize the selected information items into the simple cells or groups as shown in Figure 1 (without concern for the format or information model of the base sources). Each row contains information about a single patient, with the four columns containing patient identifying information, (a subset of) the patient's current problems, (a subset of) recent lab results or other reports, and notes (including a "To Do" list for the patient). While the information elements selected for this synopsis will generally suffice for the task at hand (patient rounds), the physician may need to view an element (such as a problem or a lab result) in the original source [Gorman 2000, Ash 2001]. However, this paper artefact obviously provides no means of automatically returning to the original context of an information element.

In an ICU, we have observed a clinician actively working with a potentially diverse set of underlying information sources as she prepares to visit a patient, selecting bits of information from the various information sources, organizing them to suit the current purpose, possibly elaborating them with highlighting or annotation, or mixing them with new additional information, including new relationships among bits of information [Gorman 2000].

In our work [Delcambre 2001], we have put forth the notion of *superimposed information* for use in such scenarios. The *superimposed layer* contains *marks*, which are encapsulated addresses, to the information elements of interest in the *base layer*. More than that, the superimposed layer may contain additional information (beyond marks) and may be structured according to an appropriate conceptual model. We are particularly interested in viewing and manipulating base information using tools appropriate for the information source (e.g., Microsoft Word for .doc files, Adobe Acrobat for .PDF files, and an electronic medical record system for patient data). We have built several superimposed applications that use conceptual models that are quite different from those of any of the underlying base information sources.

In past work we have implemented superimposed applications and models that rely solely on the ability of a base application to create a mark and to return to the marked region. In this paper, we explore the use of *excerpts* and *context* for marks in superimposed applica-

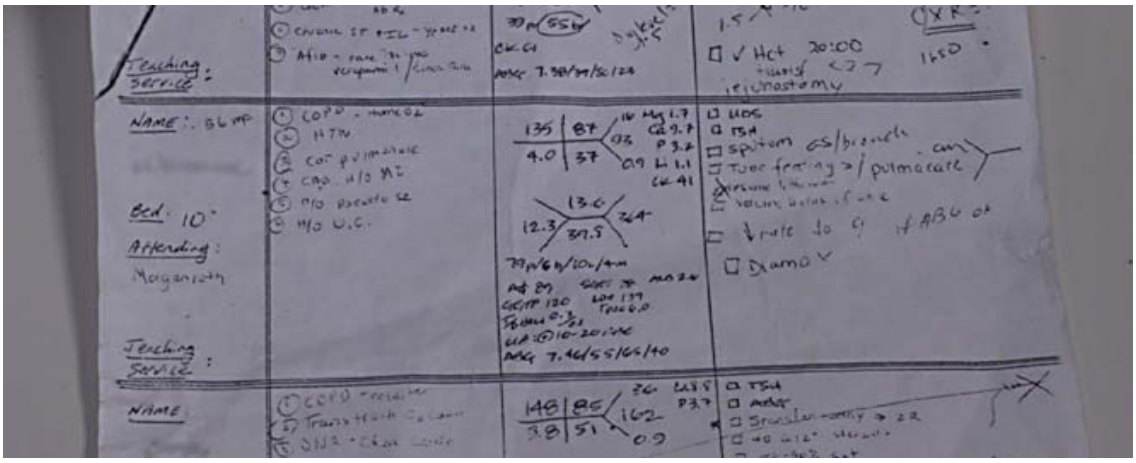

Figure 1: (Hand-drawn) Information summary as prepared by a resident prior to conducting rounds in a hospital intensive care unit (used with permission)

tions. An excerpt consists of the extracted content for a mark and the context contains additional descriptive information (such as section heading and font characteristics) about the marked information.

In Section 2 we present two superimposed applications that superimpose a new conceptual model over the base information (which is largely text documents), and makes use of excerpt and mark capabilities. In Section 3 we describe the notion of excerpts and contexts in more detail and provide the rationale for using middleware to access them. The main contribution of this paper is our architecture for building superimposed applications called the *Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE)*, presented in Section 4. This architecture makes it easy for a developer to build superimposed applications, including those that superimpose a conceptual model that is different from any of the base conceptual models. The paper concludes with a discussion of how to structure and access context, a summary of related work, and conclusions and plans for future work, in Sections 5, 6, and 7, respectively.

2 Sample Applications

We present two superimposed applications built using SPARCE to demonstrate the ability to superimpose different conceptual models, over the same corpus of base information. These applications are designed for use in the Appeals Decision Process in the Forest Services of the US Department of Agriculture (USFS).

USFS routinely makes decisions to solve (or prevent) problems concerning forests. The public may appeal any USFS decision after it is announced. The appeal process begins with a set period of time during which an appellant can send in an appeal letter that raises one or more issue with a USFS decision or the decision-making process. A USFS editor processes all appeal letters pertaining to a decision and prepares an appeal packet for a reviewing officer. An appeal packet contains all documents a reviewing officer might need to consult while formulating a recommended decision about the complete set of issues raised in the appeals. This set of documents is called the Records, Information, and Documentation (RID) section of the appeal packet. This section contains a *RID letter* that lists the issues raised and a summary response for each issue. An Editor synthesizes a RID letter using documents in the RID such as the Decision Notice, the Environmental Assessment, the Finding of No Significant Impact (FONSI), and specialists' reports. In the RID letter, the editor presents information from other documents in a variety of forms such as excerpts, summaries, and commentaries. In addition, the editor documents the location and identity of the information sources referenced in the RID letter.

2.1 RIDPad

Composing a RID letter requires an editor to maintain a large working set of information. Since it is not unusual for an editor to be charged with preparing appeal packets for several decisions simultaneously, the editor may need to maintain several threads of organization. Though using documents in electronic form can be helpful, such use does not necessarily alleviate all problems. For example, the editor still needs to document the identity and location of information. In using electronic documents, the editor may have to cope with more than a dozen documents simultaneously.

RIDPad is a superimposed application for the USFS appeal process. A USFS editor can use this application to collect and organize information needed to prepare a RID letter. A RIDPad instance is a collection of *items* and *groups*. An item is a superimposed information element associated with a mark. It has a name and a description. The name is user-defined and the description is the text excerpt from the associated mark. A group is a convenient collection of items and other groups.

Figure 2 shows a RIDPad instance with information concerning the "Road 18 Caves" decision (made in the Pacific Northwest Region of USFS). The instance shown has eight items (labeled Summary, Details, Comparison of Issues, Alternative A, Alternative B, Statement, Details, and FONSI) in four groups (labeled Environmental Assessment, Proposed Action, Other Alternatives, and Decision). The group labeled "Environmental Assessment" contains two other groups.

The information in the instance shown comes from three distinct base documents in two different base applications. (The item labeled "Comparison of Issues" contains an MS Excel mark; all other items contain MS Word marks.) All items were created using base-layer support included in the current implementation of SPARCE.

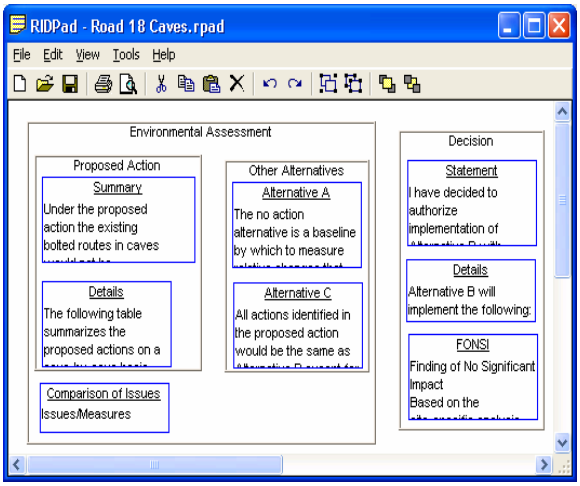

Figure 2: A RIDPad Instance

RIDPad affords many operations on items and groups. A user can create new items and groups, and move items between groups. The user can also rename, resize, and change visual characteristics such as colour and font for items and groups. With the mark associated with an item, the user can navigate to the base layer if necessary, or browse the mark's context from within RIDPad via the *Context Browser* (as shown in Figure 3). Briefly, the Context Browser is a superimposed application window with information related to a mark. Figure 3 shows the Context Browser for the item labelled "FONSI". From the context elements listed on the left we see that this item has both content and presentation kinds of context elements. The browser displays the value of the selected context element to the right. The formatted text content is currently selected and displayed in the browser.

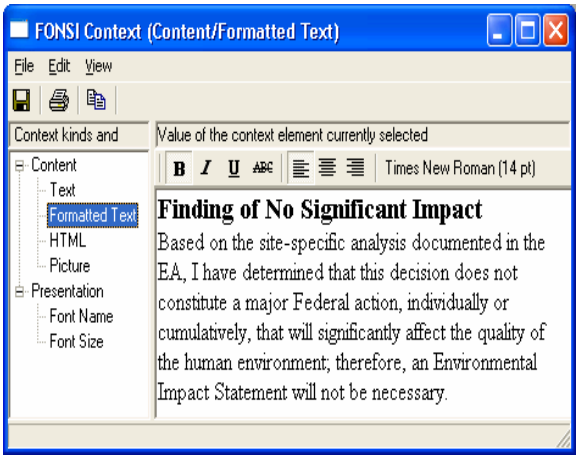

Figure 3: Context of a RIDPad Item

RIDPad superimposes a simple conceptual model over the selected base information with Group and Item as the only model constructors. A group contains a name, size, location, and an ID. An item contains a name, description, size, location, and an ID. Items can occur within a Group and Groups can be nested within a Group. Figure 4 shows the model as a UML Class Diagram. The class RIDPadDoc represents the RIDPad instance which includes information that will likely be used to prepare the RIDPad document.

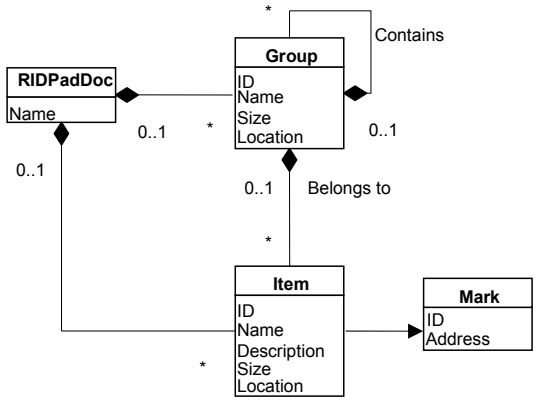

Figure 4: RIDPad Information Model (Simplified)

2.2 Schematics Browser

Appeal letters from different appellants in the USFS appeal process tend to share features. They all contain appellant names and addresses, refer to a Decision Notice, and raise issues. Such similarities suggest a schema for appeal letters. A *superimposed schematic* is an E-R schema superimposed over base information [Bowers 2002]. The Schematics Browser (see Figure 5) is a superimposed application that demonstrates the use of superimposed schematics. It is meant to allow USFS personnel to consider a set of appeal decisions to look for important issues or trends. The Schematics Browser might be used to support strategic planning activities.

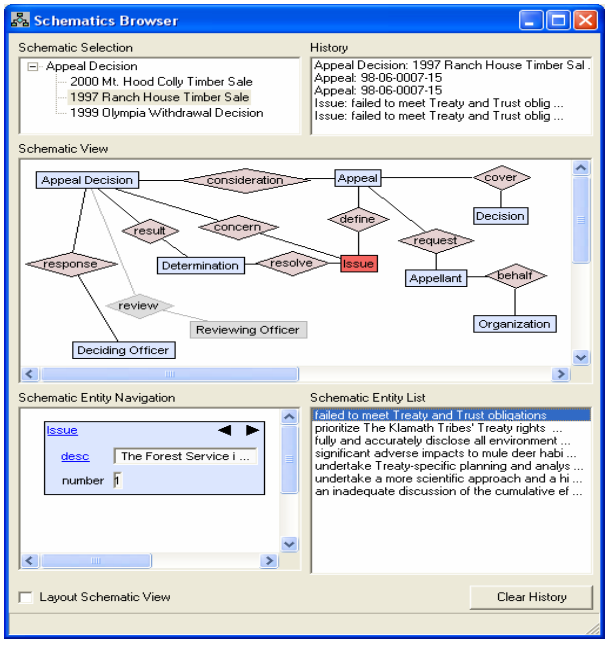

Figure 5: Schematics Browser

Figure 5 shows an instance of a USFS *appeal decision* schematic opened in the Schematics Browser. The upper left frame lists instances of the appeal decision schematic. The user can select one of these instances, and then use the large middle frame to browse through information associated with the decision. The "1997 Ranch House Timber Sale" appeal decision is selected in Figure 5. This schematic allows the user to easily browse from a particular issue to the appeal letter(s) where the issue was raised to the appellant who raised the issue, for example.

Marks into any number of base sources can be associated with entities, relationships, and attributes (but only one mark per entity and attribute). When an entity, relationship, or an attribute has an associated mark, a user can either visit the base layer or choose to view the excerpt from within the browser.

Figure 6 shows a simplified version of the information model the Schematics Browser uses in superimposing the E-R model over base information. The browser stores all superimposed information in a relational database. This structure is a simple generic model that accommodates arbitrary Entity-Relationship style schematics.

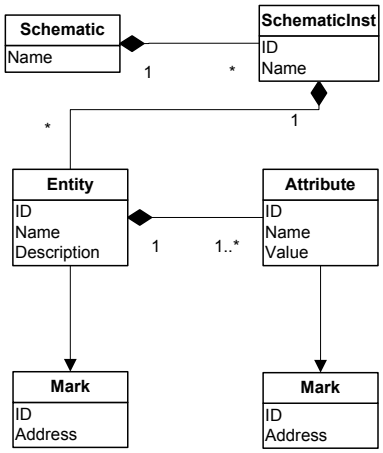

Figure 6: Schematics Browser's Information Model

Figure 7 uses the Schematic Browser's meta model to show a partial superimposed schematic instance. It shows an instance of the "1997 Ranch House Timber Sale" appeal decision schematic (also shown in Figure 5) and an Issue entity. It also shows the two attribute instances, `desc` and `number`, of the Issue entity. The `desc` attribute is associated with a mark instance (ID 41). In this simple implementation, the schematic instance data has its corresponding type information stored in the `Name` field.

2.3 Impact of Superimposed Information on Conceptual Model(s)

Superimposed information introduces one significant modeling construct – the mark. The mark spans between information at the superimposed layer and information in the various base layer sources. The mark thus serves as a bridge between the conceptual model used in the superimposed layer and the conceptual model used in a base information source.

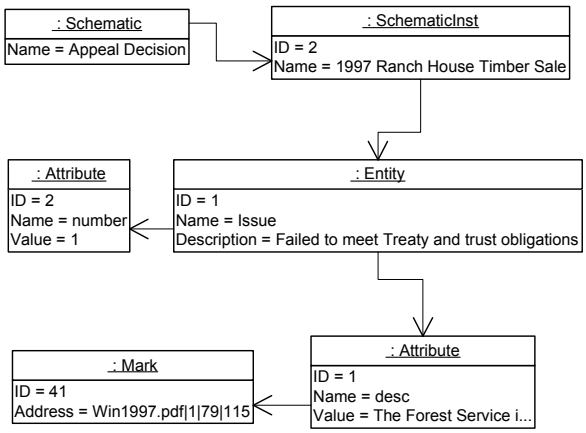

Figure 7: Partial Superimposed Schematic Instance

In the RIDPad application, the superimposed model consists of groups and items, where groups can be nested. This model is somewhat like a simplified XML model where groups are analogous to elements. But one important difference is that items contain marks, as opposed to PCDATA or other content. In a similar manner, the Schematics Browser uses a superimposed model that is similar to an entity-relationship model, but marks may appear as attribute values. In addition, each entity and relationship instance may be anchored, i.e., may be in one-to-one correspondence with a mark.

Any superimposed application, by definition, includes marks in the superimposed layer. Thus, the conceptual model used in the superimposed layer must, necessarily, be extended to include marks in some manner.

The use of marks has no impact on the conceptual model of the base layer. In fact, the use of marks, in general, requires no change to the base information or the base application. Marks encapsulate an address to an information element in the base source. Thus, the use of marks requires an addressing scheme for each base source that participates in a superimposed application. The addressing scheme may exploit the data model of the base information source. As an example, we could use XPath expressions to address information elements in an XML document. It is also possible to use addressing schemes that are independent of the data model used in the base information source. For example, a MS Word document could be converted to a PDF document and a user could create a mark using a bounding box where the interior of the box contains parts of individual characters. Regardless of the addressing scheme used in a mark, the superimposed layer is shielded from the details of the addressing scheme as well as the details of the conceptual model used in the base information source.

3 Excerpts and Contexts

Superimposed applications may want to incorporate contents of base-layer elements in the superimposed layer. For example, an application might use the extracted base-layer content as the label of a superimposed element. We call the contents of a base-layer element an *excerpt*. An excerpt can be of various types. For example it may be plain text, formatted text, or an image. An excerpt of one

type could also be transformed into other types. For example, formatted text in a word processor could also be seen as plain text, or as a graphical image.

In addition to excerpts, applications may use other information related to base-layer elements. For example, an application may group superimposed information by the section in which the base-layer elements reside. To do so, the application needs to retrieve the section heading (assuming one exists) of each base-layer element. We call information concerning a base-layer element, retrieved from the base layer, its *context*. Presentation information such as font name and location information such as line number might be included in the context of a mark. The context of a base-layer element may contain more than one piece of information related to the base-layer element. Each such piece of information is a *context element* (and context is a collection of context elements).

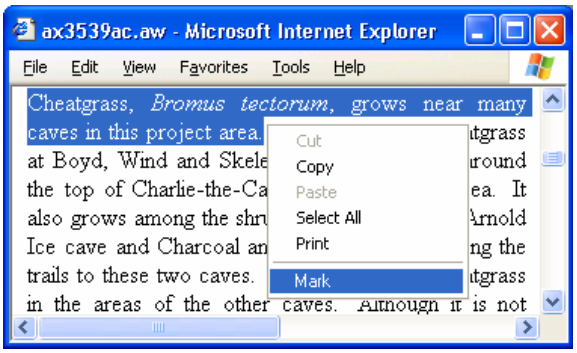

Figure 8: A Base-Layer Selection

Figure 8 shows a fragment of an HTML page as displayed by a web browser. The highlighted region of the fragment is the marked region. Table 1 shows an excerpt and a few context elements of this marked region. The column on the left lists names of context elements whereas the column on the right shows values of those context elements.

Name	Value
Excerpt	Cheatgrass, Bromus tectorum, grows near many caves in this project area.
HTML	Cheatgrass, Bromus tectorum , grows near many caves in this project area.
Font name *(Inherited)*	Times New Roman
Font size *(Inherited)*	12

Table 1: Sample Context Elements of an HTML Mark

Note that superimposed applications may access context information that a user might not explicitly access (or even be aware of). For example, consider the marked region shown in Figure 8. The HTML markup for this region (shown in Table 1) does not contain font information. If a superimposed application needs to display the mark's excerpt exactly as it is in the base layer, the application needs to examine the markup of the enclosing element, possibly traversing to the beginning of the document (because font characteristics can be inherited in HTML). The superimposed application may also need to examine the configuration of the Web browser to retrieve some or all of the format specification.

Several kinds of context are possible for a mark. The following is a representative list of context kinds along with example context elements for each kind.

- Content: Text, graphics.
- Presentation: Font name, color.
- Placement: Line number, section.
- Sub-structure: Rows, sentences.
- Topology: Next sentence, next paragraph.
- Container: Containing paragraph, document.
- Application: Options, preferences.

Contexts can vary across base-layer types. For example, the context of a mark to a region in a graphics-format base layer might include background colour and foreground colour, but not font name. However, the context of a mark to a selection in a web page might include all three elements. Contexts can also vary between marks of the same base-layer type. For example, an MS Word mark to text situated inside a table may have a "column heading" context element, but a mark to text not situated in a table does not include that context element. Lastly, the context of a mark itself may change with time. For example, the context of a mark to a figure inside a document includes a "caption" context element only as long as a caption is attached to that figure.

Supporting excerpts and contexts for marks are a natural extension of our original notion of mark as an encapsulated address. Because we use the same mechanism to support both contexts and excerpts, we will often use the term "context" broadly to refer to both kinds of information about a base-layer element.

Accessing information inside diverse base-layer types requires superimposed applications to work with a variety of base information models, addressing mechanisms, and access protocols. In addition, base applications may have different capabilities. For example, base applications may vary in their support for navigation or querying, but users of superimposed applications may want to navigate through selected base information elements seamlessly and uniformly, e.g., using the Schematics Browser. We use middleware to ease communication between the two layers and make up for deficiencies of base applications. And we want the middleware to allow independent evolution of components in these layers.

By providing a uniform interface to base information and its context, the middleware reduces the complexity of superimposed applications and allows superimposed application developers to focus on the needs of their applications such as the intricacies of the conceptual model they aim to superimpose.

4 SPARCE

The Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE) is a middleware for mark and context management [Murthy 2003]. It is designed to be extensible in terms of supporting new base-layer types

and contexts, without adversely affecting existing superimposed applications.

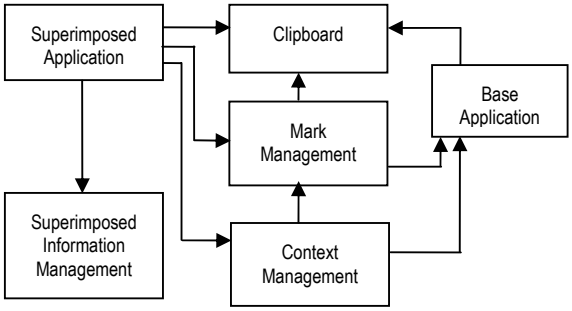

Figure 9: SPARCE Reference Model

Figure 9 shows a reference model for SPARCE. The Mark Management module implements operations such as mark creation. It also maintains a repository of marks. The Context Management module is responsible for retrieving context of base information. This module depends on the Mark Management module to locate information inside base layers. The Clipboard module is modelled after the Clipboard object in operating systems such as Macintosh and MS Windows. The Superimposed Information Management module provides storage service to superimposed applications. We have developed a generic representation for information, called the Uni-Level Description [Bowers 2003], that can represent information (including superimposed information) structured according to various data models or representation schemes, such as XML, RDF or database models, in a uniform way. In this architecture, superimposed applications can choose whether they use this module for storage, or another storage manager.

4.1 Key Abstractions

Table 2 provides a brief description of the classes and interfaces SPARCE uses for mark and context management. SPARCE supports context for three classes of objects: marks, containers, and applications (using the classes Mark, Container, and Application respectively). A *Container* is an abstraction for a base document (or a portion of that document). An *Application* is an abstraction for a base application. SPARCE also defines the interface *Context-Aware Object* to any base-layer element that supports context. The classes Mark, Container, and Application implement this interface. Superimposed applications use the class SPARCE Manager to create new marks and to retrieve existing marks. The SPARCE Manager maintains a repository of marks.

SPARCE treats context as a property set (a collection of name-value pairs). *Context* is the entire set of properties of a base-layer element and a *context element* is any one property. For example, the text excerpt and font name of a mark are context elements. Modelling context as a property set makes it possible to support a variety of contexts, both across and within base layers, without affecting existing superimposed applications. This model also provides a uniform interface to context of any base-layer element, for any base-layer type.

SPARCE uses the interface *Context Agent* to achieve its extensibility goal. A class that implements this interface takes a context-aware object and returns its context. That is, SPARCE does not access base-layer elements or their contexts directly. It uses external agents to do so on its behalf. However, SPARCE is responsible for associating a context-aware object with an appropriate context agent. The SPARCE Manager obtains the name of the class that will be the context agent for a mark from the description of the marks. The SPARCE Manager instantiates the context agent class by name whenever a superimposed application accesses the context of a context-aware object. Typically, there is one implementation of the context agent interface per base-layer type. For example, a PDF Agent is an implementation of this interface for use with PDF documents. A context agent implementation determines the constitution of context for its context-aware objects. SPARCE does not require an implementation to support particular context elements (nor does it prevent an implementation from defining any context element). However, we expect implementations to support kinds of context elements commonly expected (such as those listed in Section 3), and use meaningful names for context kinds and elements.

Class/Interface	Description
Mark	A mark to base-layer information.
Container	The base document (or a portion of it) in which a mark is made.
Application	The base application in which a mark is made.
Context-Aware Object (interface)	Interface to any base-layer element able to provide context. Classes Mark, Container, and Application implement this interface.
Context	Context of a context-aware object. It is a collection of context elements.
Context Element	A single piece of context information about a context-aware object.
Context Agent (interface)	Interface to any base-layer. An implementation will retrieve context from a context-aware object.
SPARCE Manager	Creates, stores, and retrieves marks; associates context-aware objects with appropriate context agents.

Table 2: SPARCE Classes and Interfaces

4.2 Creating Marks

A user initiates mark creation after selecting some information in a base application. The mark creation process consists of two steps: (1) generating the address of the selected base information, perhaps with other auxiliary information (collectively called *mark fodder*) and (2) creating a mark object in the mark repository. The address contained in mark fodder uses the addressing mechanism appropriate for the base information source. For example, the address to information inside a PDF document contains the page number and the starting and ending word indexes; the address to a selection in a spreadsheet contains the row and column numbers for the first and last cell in the selection. (Other addressing schemes are possible for these base types.)

Figure 10 depicts two possible mark-creation scenarios as a UML Use Case Diagram. (The boxes in this figure denote system boundaries; the broken arrows denote object flows.) In both scenarios, a user starts mark creation in a base application and completes it in a superimposed application. In the first scenario, labelled "Copy", the user is able to use the normal copy operation, e.g., of a word processor, to create the mark fodder. In the "Mark" use case, the user invokes a newly introduced function (such as the Mark menu item shown in Figure 8). The superimposed application retrieves the mark fodder from the Clipboard, and passes it to the SPARCE Manager. The SPARCE Manager creates a mark object (from the fodder), assigns it a unique ID, stores it in the mark repository, and returns the new object to the superimposed application.

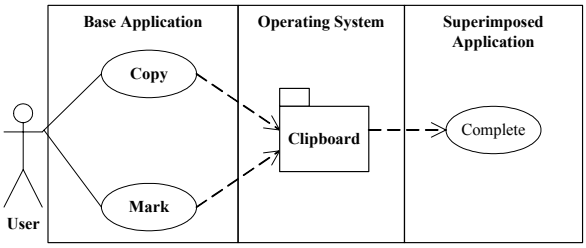

Figure 10: Two Mark-creation Scenarios

The first scenario allows a user to select base information in a preferred base application and copy it to the Clipboard without having to learn any new application, tool, or process to create marks. However, supporting this scenario requires cooperative base applications such as Microsoft Word and Excel. Some base applications do not directly support Clipboard operations, but they provide mechanisms (such as plug-ins or add-ins) to extend their environments. A special mark creation tool or menu option can be inserted in to the user interface of such applications. The Mark use case in Figure 10 demonstrates this scenario. Early versions of Adobe Acrobat and Netscape Navigator are examples of base applications in this category.

Figure 11 shows the internal representation of a mark. This mark corresponds to the selection in the HTML page shown in Figure 8. Superimposed applications do not have visibility of a mark's internal representation. They simply use the mark's interface to access its details.

4.3 Accessing Marks and Context

A superimposed application sends a mark ID to the SPARCE Manager to retrieve the corresponding mark object from the marks repository. The SPARCE Manager instantiates an implementation of the context agent interface that is appropriate for the mark. The superimposed application can work with the mark object directly (for example, to navigate to the base layer) or can interact with the mark's context agent object (for example, to retrieve mark context).

With a context object in hand, a superimposed application can find out what context elements are available. It can also retrieve values for context elements of interest. The superimposed application may use a context-element's value in various ways. For example, it may use the text content of the mark as a label, or it may apply the font characteristics of the marked region to some superimposed information.

```
<Mark ID="HTML2003Apr22065837YZXsmurthy">
   <Agent>HTMLAgents.IEAgent</Agent>
   <Class>HTMLMark</Class>
   <Address>4398|4423</Address>
   <Description/>Noxius Weeds in ea1.html
   </Description>
   <Excerpt>Cheatgrass, Bromus tectorum,
   grows near many caves in this project
   area.</Excerpt>
   <Who>smurthy</Who>
   <Where>YZX</Where>
   <When>2003-04-22 06:58:37</When>
   <ContainerID>cdocsea1html</ContainerID>
</Mark>
```

Figure 11: Internal Representation of a Mark

For ease of use, our design also allows the application to retrieve the value of a context element from the context-aware object or even from the context-agent object. An application developer may choose the access path that is most convenient to his or her particular situation.

4.4 Implementation

We have implemented SPARCE for Microsoft-Windows operating systems using ActiveX technology [COM]. The current implementation includes support for the following base applications: MS Word, MS Excel, Adobe Acrobat (PDF files), and MS Windows Media Player (a variety of audio/video file types). The agents for these base applications support the following kinds of context: content, presentation, containment, placement, sub-structure, topology, document, and application. (Some possible context elements of these kinds are listed in Section 3.)

We have implemented reusable view facilities such as the *Context Browser* to display the complete context of a context-aware object, and tabbed *property pages* to display properties of context-aware objects. We have also implemented a few testing aids. For example, we have implemented a generic context agent with limited functionality (that can be used with any base-layer type) to test integration with new base-layer types. The Context Browser is also a good testing tool when support for a new a base type is added or when definition of context is altered for a base type.

4.5 Extensibility

Supporting new context elements is straightforward in SPARCE: The new context element name is just added to the property set. Superimposed applications may ignore the new context elements if they are not capable of handling them.

Supporting new base-layer types is more involved. It requires a developer to understand the base layer and its addressing mechanisms. The developer must implement the context agent interface for the base-layer type. And the developer must implement a means to allow users to

select regions within this type of base information and copy mark fodder to the Clipboard. As we mention in Section 4.2, the developer might be able to exploit extensibility mechanisms of base applications for creating mark fodder.

We have used the extensibility mechanism to add support for MS Word, MS Excel, Adobe Acrobat, and MS Windows Media Player. It took us about 7-12 hours to support each of these base types. The SPARCE implementation and the superimposed applications were not changed or recompiled when new base types were added.

4.6 Evaluation

Our observations show that developing superimposed applications with SPARCE is relatively easy. Although the effort required to develop a superimposed application depends on the specifics of that application, using abstractions such as marks and contexts alleviate the need to model those entities in each application. For example, RIDPad is a complex application due to its graphical nature and the variety of operations it supports. However, we were able to develop that application in approximately 30 hours. As we added support for new base types using the extensibility mechanism of SPARCE, RIDPad was able to automatically work with the new base types.

The original Schematics Browser application worked only with PDF files. The application was responsible for managing marks and interacting with Adobe Acrobat. The application had no context-management capabilities. We altered this application to use SPARCE and it instantaneously had access to all base-layer types SPARCE supported (and those it will support in future). In addition, it also had access to context of base information. In less than 7 hours, we were able to alter the Schematics Browser to use SPARCE.

There are many ways to deploy the components of SPARCE and its applications (based on application and user needs). For example, RIDPad is expected to be a single-user application. Thus, all components of RIDPad and SPARCE may run on a single computer. In contrast, the Schematics Browser is likely to be used by many USFS personnel to browse schematic instances of past appeal decisions. That is, shared repositories of superimposed information and marks can be useful. Based on such analyses, we are currently in the process of evaluating different deployment configurations of SPARCE and its applications. In addition to studying performance of these configurations, we intend to explore the benefits of caching context information.

5 Issues in Context Representation

One of the areas of SPARCE design we are still exploring is the representation of context. We have considered defining contexts via data types (say, a context type for each base type), but feel that approach would be too restrictive. The set of context elements available for a mark might vary across a document. For example, a mark in a Word document might have a "column name" context element if it is in a table, but not otherwise. It is even possible that the context elements available for a single mark may change over time. For instance, the "image" context element might only be available while the invocation of the base application in which the mark was originally created is still running. A context type could define all possible context elements, where a particular mark produces null values on elements undefined for it, but that approach complicates the application programming interface (especially for context elements of scalar types such as numbers and strings). Another issue with types is making it possible to write a superimposed application without specifying in advance all the base sources it will be used with (and their context types). We have demonstrated with our current approach the ability of a superimposed application to work with new context agents without modifying the application. The superimposed application can make use of any context elements it knows about (from the elements the new agent supplies). While inheritance schemes can support some polymorphism in types, they do not seem adequate to support the arbitrary kinds of overlap we have seen among context elements across base types.

Another issue is the internal structure of a context. Currently a context is a property set of context elements, where each element is a name-value pair. Context elements also have kinds (such as presentation and substructure), which allows grouping context elements in user interfaces. We are considering giving contexts an explicit hierarchical structure. There are several alternatives for such an approach: Make a context a compound object capable of holding sub-contexts, use of qualified names (for example, format.font.fontsize), or employ a hierarchical namespace as in a directory structure. We do not see great differences in these three alternatives. The advantage of some kind of hierarchical structure, however, versus the current flat structure might come in the interface between superimposed applications and the context agent. Rather than the application asking for context elements individually (or for all context elements), it could ask for a particular subgroup of elements of interest.

A methodological issue related to context structure is how to coordinate the naming of context elements across multiple base types and multiple superimposed applications. There is no requirement currently that the "same" context element be named the same thing for different base types (or, in fact, in alternative context agents for the same base type). Even if the same name is used, the types of the associated values could be different. With an individual or small group writing context agents and superimposed applications, informal methods will work for consistency in naming. However, a more structured process will be needed at the point that context agents and superimposed applications are being produced by different organizations.

6 Related Work

Memex and Evolutionary List File were visionary proposals for organizing information from diverse sources [Bush 1945, Nelson 1965]. Hypertext and compound document models are two classes of systems that attempt to realize these visions. Hypertext systems are helpful in

preparing information for non-linear media. Although designed to help organize information, they tend to be limited in the types of source, granularity of information, and location of information that can be organized. For example, NoteCards and Dexter both require information consulted to be stored in a proprietary database [Halasz 1987, 1994]. Intermedia can address base information only at sub-document granularity [Yankelovich 1988]. Hypertext systems typically do not support retrieval of contextual information from sources.

Compound document systems are helpful in preparing information for linear media (such as paper). They can address base information at both document and sub-document granularity, but they tend to constrain display models of tools developed. For example, OLE 2 requires rectangular display areas [COM]. Like SPARCE (and unlike hypertext systems), compound document systems provide architectural support for building applications. Compound document systems support only retrieval of contents. Information sources decide the content, its format, and geometry.

Table 3 provides a brief comparison of SPARCE with hypertext and compound document systems. NoteCards, Intermedia, and Dexter are hypertext systems. OpenDoc [Apple 1994] and OLE 2 are compound document systems.

	NoteCards	Intermedia	Dexter	OpenDoc	OLE 2	SPARCE
Base types	2	3	Any	Any	Any	Any
Base location	Custom	Files	Custom	Any	Any	Any
Base granularity	Whole	Part	Both	Both	Both	Both
Context kinds	None	None	None	Content	Content	Many

Table 3: SPARCE Compared with Related Systems

Multivalent documents [Phelps 2000b] allow multiple behaviours to be superimposed on to a *single* base document using an abstraction similar to the context-agent interface in SPARCE. The system uses contents of a region of interest (and its surrounding), but only to address that region [Phelps 2000a].

In the area related to dynamism and representation of context, OLE Automation [Microsoft 1996] provides an interesting comparison to our approach. An OLE automation object exposes an interface to the type information object (ITypeInfo) that corresponds to itself. The type information object is resident in a type library (that contains type information for possibly many automation object types). Changing type information (deleting members or adding new members) requires creation of a new type-information object and a new type library. Although the framework allows each instance of an object type to return a different type-information object, the requirement to create new type information and a type library makes it impractical to do so. Consequently, type information of an OLE automation object tends to include all possible elements, without regard to whether those members are relevant in a given situation. For example, the type information for a Range object of a MS Word document contains over 30 members [Microsoft]. The value of a member of scalar type that is not applicable for a given Range object will be equivalent of NULL (and the inapplicable collection-type members will be empty). In SPARCE, the context of a mark contains only those elements that apply to the mark.

It might seem that links in OLE 2 compound documents provide similar functionality to marks. An OLE 2 compound document supports only retrieval of contents from links. It does not provide a mechanism from within a compound document to obtain the OLE automation object that corresponds to a link (even when the link source defines an automation object corresponding to the region the link represents). As a consequence, context-like information about the linked region cannot be accessed via the link directly. For example, linking a selection S from the main body of MS Word document D1 into Word document D2 makes D2 a compound document. However, the Range object for S (which is available in D1) is not accessible through the link in D2. A user needing more information about S must navigate to the source document D1. SPARCE not only provides the ability to link information via marks, it also provides access to context of the mark through the mark itself.

7 Discussion and Future Work

One way to view our work is that we have extended the standard modelling building blocks (integers, floats, dates, strings, etc.) with a new primitive—*mark*—that encapsulates an information element from an external source. A conceptual model (extended to be a superimposed model) can permit the use of marks in any of its structuring constructs (tuples, relationships, attributes, entities, etc.), without regard to the complexities of the underlying element. Support for *context* allows superimposed applications to extract information from that element and its surrounding elements or the information source in a controlled manner, to augment what is explicitly stored in the superimposed model.

As a means to provide "new models for old data," our approach is quite different from data integration approaches such as mediators and data warehouses. Such approaches seek to provide an integrated view through a global schema describing base information that faithfully reflects the structure of the base source. In our work, we are exploring the use of selected base information elements (using marks). Note that the selection of marks is often performed manually, by a domain expert (e.g., a clinician or a USFS scientist), for a specific purpose (e.g., to treat a patient or prepare a RID). We have no requirement to represent the structure or relationships present within the base layer. Rather, we rely on the original application to provide interpretation for a mark and, if appropriate, to describe any relationships among marks. Standard integration approaches describe information from various sources and expect the mediator to be responsible for its interpretation.

The superimposed layer, by definition, allows the user to mix marks with additional information that may not exist in any of the base information sources. Such information may augment the base layer, e.g., by making implicit information explicit (e.g., "this issue relates only to Alternative A") or by providing commentary. Another use of superimposed information is to link related information from multiple sources, e.g., by placing marks in the same group or by explicitly linking between information elements in two sources. Finally, the superimposed approach permits reinterpretations that are much less structured than the original. For example, base information elements can be grouped or linked without having to observe any type constraints imposed in the original source.

Exploring different representations of context and ways to reconcile context definition from different context agents is one area of our future work. Understanding the needs of new superimposed conceptual models (other than those we have described), and exploiting contexts to superimpose richer conceptual models is another area of our interest. A natural application of superimposed conceptual models would be to create means of querying jointly over superimposed and base information. We are also interested in superimposed applications that facilitate "schema later" organization of diverse information. That is, a user can start accumulating and arranging information items of interest, and—as he or she starts forming a mental conceptual model—incrementally define a superimposed model that reflects it.

8 Acknowledgements

This work has been supported by US NSF grants IIS 9817492 and IIS 0086002. We thank John Davis for helping us understand the USFS appeal process. We also thank the anonymous reviewers for their comments.

9 References

Acrobat SDK: Acrobat Software Development Kit, Adobe Systems Incorporated.

Apple (1994): The OpenDoc Technical Summary. *Apple World Wide Developers Conference Technologies CD*, San Jose; CA.

Ash, J., Gorman P., Lavelle, M., Lyman J., Delcambre, L., Maier, D., Bowers, S. and Weaver, M. (2001): Bundles: Meeting Clinical Information Needs. *Bulletin of the Medical Library Association* 89(3):294-296.

Bowers, S., Delcambre, L. and Maier, D. (2002): Superimposed Schematics: Introducing E-R Structure for In-Situ Information Selections. *Proc. ER 2002*, pp 90–104, Springer LNCS 2503.

Bowers, S. and Delcambre, L. (2003): The Uni-Level Description: A Uniform Framework for Representing Information in Multiple Data Models. Proc. of the *22nd International Conference on Conceptual Modeling (ER 2003)*, Chicago, IL, October 2003.

Bush, V. (1945): As We May Think. *The Atlantic Monthly*; 1945; July.

Delcambre, L., Maier, D., Bowers, S., Weaver, M., Deng, L., Gorman, P., Ash, J., Lavelle, M. and Lyman, J. (2001): Bundles in Captivity: An Application of Superimposed Information. *Proc. ICDE 2001*, Heidelberg, Germany, pp 111-120.

Gorman, P., Ash, J., Lavelle, M., Lyman, J., Delcambre, L. and Maier, D. (2000): Bundles in the wild: Managing information to solve problems and maintain situation awareness. *Lib. Trends 2000* 49(2):266-289.

Halasz, F.G., Moran, T.P. and Trigg, R.H. (1987): NoteCards in a Nutshell. *Proc. ACM CHI+GI Conference*, New York, NY, pp 45-52, ACM Press.

Halasz, F.G. and Schwartz, F. (1994): The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30-39, ACM Press.

Maier, D. and Delcambre, L. (1999): Superimposed Information for the Internet. *Proc. WebDB 1999 (informal)*, Philadelphia, PA, pp 1-9.

COM: The Component Object Model Specification, Microsoft Corporation.

Microsoft Corporation. (1996): *OLE Automation Programmer's Reference.* Microsoft Press.

Microsoft Office: Microsoft Office Development Resources, Microsoft Corporation.

Murthy, S. and Maier, D. (2003): SPARCE: Superimposed Pluggable Architecture for Contexts and Excerpts. OGI CSE Technical Report #CSE-03-010. 2003, May.

Nelson, T.H. (1965): A File Structure for The Complex, The Changing and the Indeterminate. Proc. *ACM 20th National Conference*, Cleveland, OH, pp 84-100.

Phelps, T.A. and Wilensky, R. (2000a): Robust intra-document locations. *Proc. 9th World Wide Web Conference*, Amsterdam, Netherlands.

Phelps, T.A. and Wilensky, R. (2000b): Multivalent Documents. *Communications of the ACM*, 43(6):83-90.

Yankelovich, N., Haan, B.J., Meyrowitz, N.K. and Drucker, S.M. (1988): Intermedia: The Concept and the Construction of a Seamless Information Environment. *IEEE Computer* 21(1): 81-83, 90-96, IEEE.