Proteus:
An Adaptable Presentation System for Multi-media Documents

—

A Thesis Proposal

Ethan V. Munson

October 25, 1991

1 Introduction

The goal of this project is to design and implement a document presentation system appropriate for and adaptable to the needs of a variety of media including, but not limited to, text documents, two-dimensional object graphics, and three-dimensional animation. This presentation system, called Proteus, is one part of the infrastructure of the Ensemble editing system.

Proteus is the first document presentation system specifically designed to adapt to the demands of a variety of media.[1] The adaptability of Proteus results from its division into a single medium-independent kernel and a collection of medium-specific shells. A particularly unique aspect of Proteus is that these medium-specific parts are generated automatically from brief specifications. An important result of this design is that editing modules for previously unsupported media can be added to Ensemble without modifying the system's infrastructure.

In this proposal, I first give an overview of the Ensemble system. Then I describe the components of the Proteus system in sections on the presentation

[1] Proteus was a sea god of the Greeks who lived either on the island of Pharos (according to Homer) or on the island of Carpathus (according to Virgil). He was completely omniscient but also secretive. In order to consult him it was necessary to surprise and bind him during his noontime nap. Upon awakening he would attempt to escape by assuming a variety of shapes such as that of a lion, a serpent, a leopard, a boar, a tree, fire, or water. If unable to escape by shape-changing, he would return to humanoid form, answer the question and dive into the sea. [4]

The Proteus system was named for its adaptability but, unlike its namesake, it is also cooperative.

model, the design of the presentation languages, the presentation engine, and the automatic generation of the medium-specific parts of Proteus. Following the description of Proteus, I summarize some related work. Finally, I conclude with a brief summary of my research plan.

2 Ensemble

Ensemble is a multiple-representation, multi-media editing system currently being implemented by the research groups of Susan Graham and Michael Harrison. It is the direct descendant of the Pan program editor [1] and the VORTEX multiple-representation document editor/formatter [3, 2]. Ensemble stands at the confluence of several important trends in both software development environments and document processing systems. These include:

Structured Documents Much recent document processing research has focused on systems which support documents with complex hierarchical structure that can be specified by persons other than the system implementor. The basic ideas were introduced by Reid in Scribe [16] and by Goldfarb's earlier work on GML [6] and have been elaborated in work by Quint and Vatton [15], Furuta [5], and various researchers in the document interchange community. All Ensemble documents are structured.

Separation of Structure and Presentation The benefits of separating the structure of a document (or any other object) from the way it is presented to the user have been widely discussed. Such a separation was endorsed as early as 1983 for user interfaces [13] and is generally presented as a key justification for current approaches to document interchange. Both the Grif document editor [15] and the Centaur program editor [10] have adopted this idea, as has Ensemble. This separation is the key element which allows the Ensemble user simultaneously to view and edit multiple representations of a single document.

Integrated Formal Language Support The Ensemble system is unique in its intention to support authors of both natural language documents and computer programs. Previously, document editors and program editors have been seen as different types of programs. However, since both structured document editors and software development environments represent their documents as trees, providing both services is rather natural. Ensemble will include the same formal language services (lexical analysis, parsing, and static semantic analysis) previously found in the Pan system [1]. In addition to being used by programmers, these services will be used to analyze and maintain source language representations of "non-program" documents.

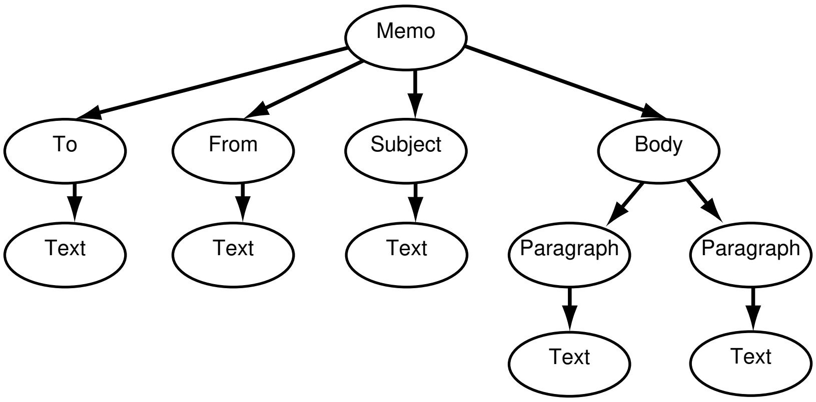

Figure 1: A structural view of a small Ensemble document, a memo.

Multi-media Ensemble is a multi-media system. Ensemble documents can contain heterogenous collections of material from every medium supported by the system. Among the media to be supported by Ensemble are text, computer programs, object graphics, three-dimensional animation, and eventually digitized audio and video. While Ensemble is by no means the first multi-media editing system, it is one of the first systems to apply the structured document paradigm to such a wide range of media.

While these are the main themes underlying the Ensemble system, some other aspects of the system must be understood in order to motivate the design of Proteus. Toward that end, the following subsections briefly explain the Ensemble document model and the architecture of Ensemble.

2.1 The Ensemble Document Model

The primary structure of every Ensemble document is hierarchical. Currently, all Ensemble documents are trees, though in the future a more complex hierarchical structure like that described in Furuta's thesis [5] may be allowed. Figure 1 shows a graphical representation of the structure of a simple document, a memo. Figure 2 shows a formatted view of this same memo.

The range of possible structures for a particular document type is specified in a formal definition called a *structure schema*. Structure schemas resemble the grammars used to specify programming language parsers, but can be ambiguous because they are used for user-guided generation of structure rather than deduction of structure by parsing. The structure schema for the memo document class is shown in Figure 3. It states that a memo is composed of four main parts, `To`, `From`, `Subject`, and `Body`. The first three parts contain text. The `Body` is further subdivided into a sequence of one or more `Paragraphs`, which

To:	Mike
From:	Ethan
Subject:	My quals proposal

I have completed the draft of my quals proposal and have placed it in your mailbox. I hope you'll be able to read it soon.

I think it's in pretty good shape, but there will certainly be some corrections and additions required. I look forward to getting your comments.

Figure 2: A formatted view of the same memo whose structure is shown in Figure 1.

are also composed of text. Note that in this memo document schema `text` is the only type of terminal node.

The appearance of an Ensemble document is determined primarily by a *presentation schema*, which is a formal description of the rules by which the document is to be formatted. To produce the memo shown in Figure 2, the presentation schema must state that a 10 point Computer Modern Roman font should be used for the entire document and that the user-supplied text of the `To` and `From` fields should be left-aligned with the text of the `Subject` field. The presentation schema would also determine which paragraphs are line-broken, how much vertical white-space appears between the paragraphs, and the size of the margins. The user may override these rules by issuing *presentation directives*, but this will probably be the exceptional case. The presentation schema languages will be discussed in detail in the following section.

2.2 The Architecture of Ensemble

Figure 4 presents a partial architecture diagram for the Ensemble system. The viewpoint of this diagram overemphasizes the role of the presentation system at the expense of editing and language-support services. However, it remains a useful illustration of the relationship between the presentation system and the rest of Ensemble. One important simplification is that the figure assumes that the document contains only one medium, though both Proteus and Ensemble can have many formatters active at one time even for a single document.

On the right of the diagram is the *window system* (X11/Motif) which accepts user input and transmits it to the editing module. The *editing module* is presented here as a black box but is actually subdivided into a structure editor, a number of media-specific editors, and a command dispatch system which

```
SCHEMA Memo;

Memo: To From Subject Body;

To: text;

From: text;

Subject: text;

Body: Paragraph+;

Paragraph: text;
```

Figure 3: The structure schema for the *Memo* document class.

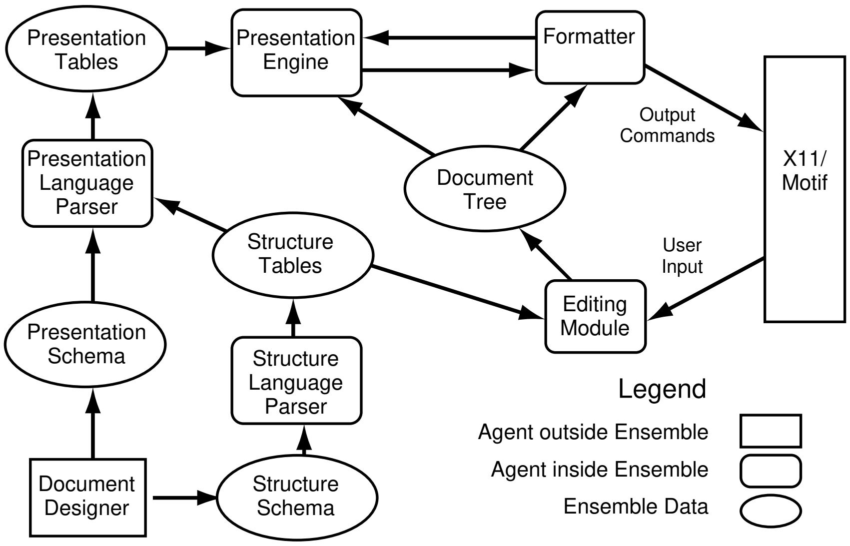

Figure 4: Partial architecture diagram for the Ensemble system. The direction of each arrow indicates the flow of information through the system. An arrow going from an agent to data indicates that the agent creates or modifies the data. An arrow going from data to an agent indicates that the agent consumes the data. The Document Designer agent is special in that it is a person. All other agents are programmatic.

multiplexes user commands to the various editors. Based on the user input, the editing module makes changes to the structure and content of the *document tree*. These changes are observed by a *formatter* which incrementally reformats the document and sends output commands to the window system. The formatter is specific to a single medium. If a document contained multiple media, then there would be formatters for each of them which would be responsible for displaying different parts of the document.

Jumping to the lower left corner of the diagram, we find the *document designer*. The document designer is a person who makes decisions about the structure and appearance of classes of documents, expressed in the form of *structure schemas* and *presentation schemas*. These schemas are translated by their respective *parsers* into *structure tables* and *presentation tables*.

The presentation schema parser uses the structure tables to find out the names of the nodes for the document class. More importantly, the structure tables are used by the editing module to guide the editing process. For instance, when the user asks to insert a new piece of structure, the editing module uses the structure tables to determine what kinds of nodes are allowed at the current location.

The presentation engine uses the presentation tables along with information about the actual structure of the document (taken from the document tree itself) to answer questions posed by the formatter. For example, a text formatter might ask what font should be used in formatting a node or where the left and right margins for a node are.

The presentation engine sometimes asks questions of the formatter. For instance, the presentation engine has no knowledge about breaking text into lines. If it needs to know the height of a paragraph in order to answer some other question about the document, it must ask the formatter. Thus, the formatter and the presentation engine have a two-way interface, though the initial question in one of these conversations always come from the formatter.

3 The Design of Proteus

The overriding concern in the design of Proteus has been to support a wide variety of media, not all of which can be identified in advance. Plans already exist for Ensemble to support the editing of both traditional text documents and real-time three-dimensional animation. These two media are extremely different. Text formatting is a two-dimensional activity whose complexity arises from the need to break the characters into lines and pages. Three-dimensional animation is actually rendered in four dimensions (three spatial dimensions plus time) and is free of all the arcana of typography. In addition, certain media, such as video, will not be supported by Ensemble initially but may be added later.

How can the presentation system of Ensemble support media with such differing requirements without requiring a complete re-implementation each time a new medium is added to the system? The obvious design solution is to find those elements that are common to the presentation needs of all media and to support those in a medium-independent kernel while all medium-specific aspects of presentation are provided separately. This is the approach taken by Proteus.

3.1 A Model of Presentation

The kernel of Proteus is based on a set of four medium-independent presentation services: attribute propagation, rectangular layout, tree elaboration, and interface functions. These services give Proteus the same range of function as existing presentation systems. One of the key questions for this research is whether these four services are sufficient for most media.

Attribute propagation is the most important of these services. Each medium has a set of attributes whose values determine the appearance of an object. For text documents, these attributes might include

- font family,
- type size,
- type style,
- a boolean determining whether to line break,
- a boolean determining whether lines are justified,
- and the amount of whitespace between lines.

Most of the time, the values of these attributes are not specified explicitly for every node. Rather, they are propagated through the document tree by constraints on the values of other nodes. A common example in text documents is the choice of font family, for which a value is typically specified at the root of the document tree and then inherited by all other nodes. Inheritance is seen here as a particular kind of constraint.

The *rectangular layout* service provides a minimal mechanism for laying out elements of the document. It makes the safe assumption that each medium has one or more dimensions (e.g. text has two). Every object in a particular medium has four attributes for each dimension: extent, minimum, maximum, and center. It is important to note that there are only two degrees of freedom among these four attributes. Typically, these two degrees of freedom are characterized as being extent and position (which subsumes minimum, maximum, and middle). This choice is intuitively reasonable, but arbitrary.[2]

[2]The layout service is called rectangular because it assumes that the objects being laid out are shaped like n-dimensional rectangles. This is much simpler than laying out general polygons. In particular, n-dimensional rectangles have unique values for the four attributes of each dimension, while polygons do not.

The dimensional attributes are similar to other attributes. Their values may be set explicitly or by constraint on attribute values of other nodes. The key difference between them and other attributes is the existence of the implicit constraints that result from having four attributes describing a system with two degrees of freedom.

The *tree elaboration* service adds material to the document tree that is not produced directly by the user. For instance, the labels for the `To`, `From`, and `Subject` fields of the memo document (see Figure 2) were produced by tree elaboration commands. The presentation engine automatically adds these nodes to the document tree in response to certain commands in the presentation schema. Whether these generated nodes are visible to the editing module has not yet been decided.

If the presentation engine were just a passive repository of information responding to queries by the formatter, certain complex formatting operations would be very difficult. Traditional text documents provide many examples. Scholarly books typically have page headings that show the title of the current chapter or section. However, the presentation engine has no way of knowing how content of the document tree is broken into pages; that is the job of the formatter. Meanwhile, the formatter cannot know what material should appear in the page header; that information is the domain of the presentation system.

The solution is to have the formatter designer create *interface functions* which allow the presentation engine to query the formatter when it needs to. In the case of the header, a function which returns the first leaf node falling on the current page will allow the presentation engine to ascend the tree to find the chapter or section title.

Together, the four services provided by the presentation system allow document designers to specify complex presentations of structured documents. Furthermore, these four services make no medium-dependent assumptions that limit the generality of the presentation system. This does not mean that the presentation system is guaranteed to meet the needs of every medium. It is possible that the course of this research will reveal other services which belong in the presentation system rather than in one or more formatters. However, the four services outlined in this section appear at this time to be a useful start.

The next three subsections describe the current state of the design and implementation of Proteus. They are followed by a section describing the work that remains to be done.

3.2 Presentation Schema Languages

As stated earlier, the appearance of an Ensemble document is primarily determined by a presentation schema, which specifies a set of formatting rules for the document's class. Each presentation schema is written in a *presentation schema language*.

Unlike earlier systems, Proteus does not have a single presentation language supporting all media. Rather, it supports a family of presentation languages, there being a separate one for use with each medium. These languages have in common their syntax and the core presentation concepts described in the preceding section. They differ in the attributes they support, the number of dimensions in which layout is performed, and the interface functions provided.

Having a separate language for each medium leaves each language uncluttered by concepts native to other media. The text language doesn't need to support layout in the Z or time dimensions, while the language for three-dimensional animation need not support the `Justify` attribute. In fact, the languages for different media can give different names to the same dimension if doing so better suits the conventions of the media. Furthermore, the addition of a new medium to Ensemble does not invalidate any existing schemas for other media.

The key disadvantage to having multiple presentation schema languages is that it is not possible to write a single presentation schema for a multi-media document. Instead, each subcomponent of the document which is to be presented in a foreign medium must have a separate schema. Whether this is an important limitation remains to be investigated.

3.2.1 A Prototype Language

Currently, only a simple prototype presentation schema language for textual documents has been designed. This language is essentially a subset of P, the presentation language of Grif [14]. This language supports attribute propagation and rectangular layout but doesn't allow tree elaboration. While a syntax for interface functions has been designed, it is not used in the prototype language. A sample presentation schema for documents of the `Memo` class is shown in Figure 5.

A presentation schema in this language has three parts, a preamble, a defaults section, and a rules section. The preamble gives the name of the presentation and the name of its corresponding structure schema. In the schema shown in Figure 5, the presentation is named `NormalMemo` and it is for use with the `Memo` structure schema.

The defaults section defines rules for computing attribute values that are to be used when no rule is given in the rules section. Most default rules define an attribute by constraint on some other node. In the `NormalMemo` example, the default rule for the `Height` attribute constrains it to be the height of the node's children. However, attributes can be given actual values by default, as the `Break` attribute is given the default value of `No`.

The rules section defines rules for each type of node in the document class. Only those rules that deviate from the default rules need to be defined. In the example, the rules for the `Memo` node set up values that are inherited by most of the other nodes. The remaining nodes in this simple document generally need

```
PRESENTATION NormalMemo
      FOR Memo;

DEFAULT
   BEGIN
   Width = Parent . Width;
   Height = AllChildren . Height;
   VertPos: Top = LeftSib . Bottom;
   HorizPos: Left = Parent . Left;
   Justify = Parent . Justify;
   Break= No;
   FontFamily = Parent . FontFamily;
   Style = Parent . Style;
   Size = Parent . Size;
   LineSpacing = Parent . LineSpacing;
   END;

RULES
   Memo :
      BEGIN
      Width = 432;
      Height = 324;
      HorizPos : Left = Parent . Left + 72;
      VertPos :Top = Parent . Top + 36;
      FontFamily = "Times";
      LineSpacing = 1.2;
      Style = "Roman";
      Size = 11;
      Justify = No;
      END;
   To :
      BEGIN
      HorizPos : Left = RightSib . Left;
      VertPos : Top = Parent . Top ;
      END;
   From :
      BEGIN
      HorizPos : Left = RightSib . Left;
      END;
   Subject :
      BEGIN
      HorizPos : Left = Parent . Left + 54 ;
      END;
   Body :
      BEGIN
      VertPos : Top = LeftSib . Bottom + 10 ;
      Justify = Yes;
      END;
   Paragraph :
      BEGIN
      Break = Yes;
      END;
END
```

Figure 5: Prototype presentation schema for the Memo document class.

only adjust their position.

While the gist of this prototype language should be clear, a number of things should be explained in more detail.

- The language is case-insensitive. Since it is hoped that document designers will not always be programmers, it seems best to prevent semantic errors caused by mistakes in character case.

- Attributes may be boolean, string or numerically valued. The boolean literals are `Yes` and `No`. Strings literals are enclosed in double quotes. Numeric literals are integers and they are considered to be in units of pixels. An exception to this last statement is the value given to the `LineSpacing` attribute, which is a unit-less scalar. This device-dependent use of pixel units will be eliminated soon.

- There are four terms which are used to specify relatives of the node in the tree: `Parent`, `LeftSib`, `RightSib`, and `AllChildren`. They can be composed to access values at non-adjacent nodes, as in

  ```
  LeftSib(Parent)
  ```

  which specifies an "aunt".

  The `AllChildren` term is special and can only be used with layout attributes. It instructs the presentation engine to scan all children of the node to find the smallest minimum position and the largest maximum position among them. The requested attribute is then computed from those two values. For instance, the value of `AllChildren.Height` would be computed by find the highest `Top` and the lowest `Bottom` among the children and taking their difference.

- The position attributes (`VertPos` and `HorizPos`) subsume the minimum, maximum, and middle of their respective dimensions. This is the approach taken by Grif. It gives the language simple semantics and facilitates a simple implementation of the presentation engine. I am interested in finding a more general approach.

- Certain kinds of rules can fail. For instance, constraints based on the left sibling fail for nodes which are the leftmost child. If a normal rule fails, the corresponding default rule is used instead. If a default rule fails, there is an implicit default rule that the value of the node's parent is used. To prevent the implicit default rule from failing, the root of the document tree must have a valid rule for every attribute.

3.3 Runtime Presentation System

The runtime part of the presentation system, like the linguistic part, has a well-defined division between the medium-independent and medium-specific aspects of presentation.

The medium-independent aspects are handled by the presentation engine, which provides the attribute propagation, dimensional layout, and tree elaboration services. The attribute propagation and dimensional layout services require some kind of constraint maintenance system. The current implementation of the presentation engine is hand-coded and uses direct depth-first evaluation of the attribute rules. The additional constraints imposed by the dimensional attributes are handled by always storing the position in terms of the minimum and computing the middle and maximum from the values for extent and minimum. The current implementation does not provide tree elaboration services.

The medium-specific aspects of the runtime system are handled through an interface that sits between the presentation engine and the medium's formatter. This interface is type-safe and completely hides the implementation details of the presentation system from the formatter.

3.4 Automatic Generation of Medium-Specific Parts

Proteus is designed to have its medium-specific parts automatically generated from formal specifications. These specifications must provide the following information:

- For each dimension used by the medium, the name of the dimension and the names of its four attributes.
- For some dimensions it will be necessary to identify the units which are available for use. For example, the time dimension uses hours, minutes, and seconds instead of distance units like inch or centimeter. In such cases, one type of unit must be identified as canonical and conversion ratios between it and the other units must be supplied.
- The name and type of each attribute.
- For each interface function, the name of the function and the types that it maps between.

Figure 6 gives a rough example of what one of these specifications might look like. The example would generate a parser for the prototype text presentation schema language. The specification begins by defining the two dimensions and then identifies the regular attributes. The interface function section is empty because the current language has no interface functions. However, if this medium had an interface function which returned the first leaf node on the current page, the specification might contain a line like

```
FirstNodeOnPage: null -> node;
```

indicating that the function `FirstNodeOnPage` maps from a null argument list to a node in the document tree. All interface functions have the current node as an implicit argument.

The prototype version of Proteus does not contain an automatic generation component.

3.5 Future Work

The current implementation of Ensemble's presentation system accepts a slightly different language than that just described, but the differences are small compared to the changes yet to come. In addition to bringing the implementation to the point where it accepts the language as designed, there is a considerable amount of work left to be done.

Tree Elaboration: Support for tree elaboration must be added to both the presentation languages and the presentation engine. For the languages, constructs similar to those used in Grif should be sufficient initially. For the presentation engine, the key issue to be resolved is how the nodes generated in response to instructions in the presentation schema will be connected to the main document tree.

Dimensional Attributes: In the prototype, the extent attribute is given a more important role than the three other dimensional attributes. The presentation languages would be more elegant if the four attributes of each dimension were treated identically. Early attempts at doing so have not met with success, but further exploration of the issue is warranted.

Tree Operations: The range of tree operations provided by the presentation languages will be expanded to include more complex functions. For instance, it would be useful to choose nodes based on their type.

Attribute Types: The current type system for attributes is very restricted. In particular, it might be desirable to support aggregate types like glue, which has values for size, stretch and shrink.

Numbering: Support for automatic numbering of document components (such as sections, tables, and figures) will also be added to the system. Proteus will act as a platform for empirically testing the analytic results in [9].

Error Checking: The prototype has no compile-time checks on the semantics of the presentation schemas. The prototype presentation engine is also largely lacking error checks. Proteus must have stronger error checking.

```
medium text;

dimensions {

  Horizontal {
    Extent: Width;
    Position: HorizPos;
    Min: Left;
    Max: Right;
    Middle: HMiddle;
  }
  Vertical {
    Extent: Height;
    Position: VertPos;
    Min: Top;
    Max: Bottom;
    Middle: VMiddle;
  }
}

attributes {
  FontFamily: string;
  Style: string;
  Size: numeric;
  Break: boolean;
  Justify: boolean;
  LineSpacing: numeric;
}

functions {
}
```

Figure 6: A possible automatic generation specification for the text medium.

4 Related Work

The development of Proteus has not occurred in an intellectual vacuum. Work related to Proteus falls into two major categories, presentation systems and language families.

4.1 Presentation Systems

Systems supporting the structured document paradigm have been around for a number of years. The batch formatter Scribe [16] and its spiritual descendant, LaTeX [12], both allow authors to describe documents in structural terms rather than physical terms. Both Scribe and LaTeX support a variety of document structures by having formal specifications of document styles but these specifications intertwine the definition of structure with the definition of presentation.

A more recent trend has been to separate the specification of structure from the specification of presentation. Such a separation is one of the guiding principles underlying the SGML [7] and ODA [11] document interchange standards. SGML is intended to describe only document structure. The specification of presentation is not considered by the standard. An ODA document description includes both a structure part and a physical part. However, the physical part of an ODA description describes only that particular document, not a way of presenting a class of documents.

The two systems closest in spirit to Ensemble are Furuta's **pedtnt**[3] and Grif [15]. Unfortunately, the presentation system for the **pedtnt** was never completed.

This brings us to Grif, the only interactive structured document editor of which I am aware. Grif does have separate formal specifications of structure and appearance and, like Ensemble, is an interactive editor/formatter. The design of Proteus owes a great debt to Grif, both in concept and in the details. The four presentation services of Proteus were derived by careful examination of the semantics of P, the presentation schema language of Grif. Furthermore, the basic syntax of Proteus's prototype language for the text medium is quite similar to that of P.

The most obvious difference between Proteus and the presentation system of Grif is that Grif tries to support all media with a single presentation schema language. This appears to make it difficult to add support for new media, since the language is not modular. This lack of modularity appears to extend to the runtime aspects of Grif's presentation system. However, there are no published descriptions of the internals of Grif, so comparisons are necessarily speculative. Beyond the lack of modularity, plans for Proteus's presentation languages include more powerful tree expressions and a different paradigm for

[3] **pedtnt** stands for "prototype editor of the tnt". "**tnt**" was the name of the core data structure used by Furuta, and stands for "tree/not-tree".

specifying component numbering. The key overall difference between Grif and Proteus is that Proteus is designed for use with arbitrary media.

4.2 Language Families

Superficially, it would seem that there exist many families of programming languages. One might speak of the family of JOVIALs or the family of parallel FORTRANs. There is some justice in using the term "family" for these cases. The parallel FORTRANs share the same uniprocessor core and differ only in how they manage parallelism. Clearly these languages are "related" to each other.

Proteus is based on a more ambitious notion of language families. The presentation schema languages share a syntax and a set of operators on the primitive types (document tree node, boolean, string, and number). Much of the semantics of the runtime system (the presentation engine) is also common to the entire family. However, the languages are so different in their details that a specification written for one medium would be nearly impossible to modify for use with another medium. This is quite different from the case of the parallel FORTRANs, which have in common all the constructs of uniprocessor FORTRAN.

Backus's several variations on the FP functional programming language are sometimes thought of as a family. In this case, the differences between the languages are almost entirely in their run-time semantics (e.g. lazy vs. strict evaluation).

The system most similar to Proteus is the Larch family of specification languages [8]. The Larch Shared Language is shared by all members of the family and is used to specify the semantics of data structures and the operations on them. The variant parts of the family members are specified using one of the Larch interface languages. The interface languages are used to bind the abstract concepts specified with the shared language to the data types and concrete semantics of normal programming languages like CLU and Pascal. A specification is not complete without elements from both the Larch Shared Language and one of the interface languages (e.g. Larch/CLU or Larch/Pascal).

In both Larch and Proteus, members of the language family are similar but do not even approach being identical. It is also informative to note that both Larch and Proteus are intended to be used for declarative specifications. Perhaps the language family paradigm is best suited for use with specification languages rather than programming languages.

Proteus differs from Larch in two important ways. First, unlike Larch, a Proteus presentation schema does not have a well-defined shared portion. Rather, the core concepts and the syntax by which they are invoked are shared. Second, the members of Proteus's language family are automatically generated while the Larch interface languages are custom-built. This is not a flaw of Larch. The Larch interface languages must adapt to the needs of programming languages whose runtime systems do not conform to a single well-defined machine model. In contrast, Proteus has the luxury of imposing a model of presentation on its languages.

5 Research Plan

The most immediate task facing me is to bring the prototype version of Proteus to a point where it conforms to the design described in this proposal. When brought to this point, Proteus will have sufficient functionality to allow Ensemble to create and edit simple documents. I expect to reach this point by the time of my qualifying exam.

Once this zero'th version of Proteus is somewhat stable, I will undertake two tasks in parallel. The most critical is to add the tree elaboration service currently missing from the prototype. This requires changes to both the schema parser and the presentation engine. Simultaneously, Ensemble's text formatter needs to be improved so that it can format more complex documents and, in the process, exercise the presentation services more energetically. At this point, concurrent work on the programming language components of Ensemble will have advanced to a point where experiments with automatic pretty-printing can be conducted.

The next step will be to construct the generator for the medium-specific parts of Proteus. This will require some additional language design and a parser implementation. The correctness of the generator will be examined by comparing the behavior of the generated text presentation system to that of the hand-coded version and by generating presentation systems for other media (object graphics and three-dimensional animation). I plan to reach this point by May of 1992.

Once the generator is working adequately, I will be able to return my attention to expanding the services provided by Proteus. Among the features to be added are:

- automatic component numbering,
- more powerful tree operations,
- support for eliding portions of a document,
- and better error checking.

I expect to be engaged in these and similar tasks until my departure from the hallowed halls of Cal.

To date, my key conceptual result has been the articulation of a model of document presentation that is medium-independent and the realization that the medium-specific parts of the implementation can be automatically generated. However, this model of presentation has yet to be tested against a variety of media. An important area for further research is to examine the adequacy of this model.

In addition, there is considerable research to be done on the details of presentation. For instance, a careful search for the best ways to handle the dimensional attributes is needed. Other areas for research include identifying a sufficient type system for presentation systems and examining the efficiency trade-offs in document presentation. I will explore these questions as the course of my work allows.

I hope to have completed enough of the research and engineering work to start writing my thesis in September, 1992. I plan to graduate in May 1993.

References

[1] Robert Allan Ballance. *Syntactic and Semantic Checking in Language-Based Editing Systems.* PhD thesis, University of California, Berkeley, Berkeley, CA 94720, December 1989.

[2] Pehong Chen. *A Multiple Representation Paradigm for Document Development.* PhD thesis, Computer Science Division, University of California, Berkeley, California, 1988.

[3] Pehong Chen and Michael A. Harrison. Multiple representation document development. Technical Report 87/367, Computer Science Division, University of California, Berkeley, California, July 1987.

[4] Encyclopedia Brittanica, 1964. Proteus.

[5] Richard K. Furuta. *An Integrated, but not Exact-Representation, Editor/Formatter.* PhD thesis, Computer Science Department, University of Washington, Seattle, Washington, September 1986. Published as Technical Report No. 86-09-08.

[6] Charles F. Goldfarb. A generalized approach to document markup. In *Proc. of ACM SIGPLAN/SIGOA Symposium on Text Manipulation*, pages 68–73, Portland, Oregon, June 8–10 1981. Available as *SIGPLAN Notices* 16(6) or *SIGOA Newsletter* 2(1–2).

[7] Charles F. Goldfarb, editor. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML).* International Organization for Standardization, Geneva, Switzerland, 1986. International Standard ISO 8879.

[8] John V. Guttag, James J. Horning, and Jeannette M. Wing. The Larch family of specification languages. *IEEE Software*, pages 24–36, September 1985.

[9] Michael A. Harrison and Ethan V. Munson. Numbering document components. Technical Report 90/568, Computer Science Division, University of California, Berkeley, California, June 1990. To appear in *Electronic Publishing*.

[10] INRIA: Centaur Project, Sophia-Antipolis, France. *The Ppml Manual*. For Version 1.1 of Centaur.

[11] International Standards Organization. *Office Document Architecture*, 1986. Draft International Standard 8813.

[12] Leslie Lamport. LaTeX*: A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1986.

[13] Gunther E. Pfaff, editor. *User Interface Management Systems*. Eurographic Seminars. Springer-Verlag, November 1985. Proceedings of the Workshop on User Interface Management Systems, held in Seeheim, FRG, November 1-3, 1983.

[14] Vincent Quint. Les langages de grif. Distributed with the program, July 1988.

[15] Vincent Quint and Irène Vatton. Grif: An interactive system for structured document manipulation. In J. C. van Vliet, editor, *Text processing and document manipulation*, pages 200–213. Cambridge University Press, April 1986.

[16] Brian K. Reid. *Scribe: A document specification language and its compiler*. PhD thesis, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, October 1980. Available as technical report CMU-CS-81-100.