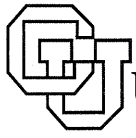
SmartMedia Tools: Bridging the Gap Between Generic Applications and Domain-Oriented Systems

Marcus Stolze
Tamara Sumner

CU-CS-792-95



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1995		**00-00-1995 to 00-00-1995**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
SmartMedia Tools: Bridging the Gap Between Generic Applications and Domain-Oriented Systems	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Department of Computer Science,University of Colorado,Boulder,Co,80309**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**10**	

SmartMedia Tools: Bridging the Gap Between Generic Applications and Domain-Oriented Systems

Markus Stolze *Tamara Sumner*
Department of Computer Science and the Center for LifeLong Learning and Design
University of Colorado
Boulder, CO USA 80309-0430
Email: [stolze, sumner]@cs.colorado.edu

ABSTRACT

Our intuitions regarding the usability and utility of generic versus domain-specific applications may be too simplistic. Empirical studies indicate that professional practitioners need both: (1) the rich, flexible formatting and editing features associated with generic tools, and (2) the supportive functionality offered by domain-specific tools. The problem is that today's tools are either generic (and flexible) or domain-specific (and supportive), but not both. SmartMedia Tools change this situation by supporting practitioners to bridge this gap and work at different points along the flexibility / support continuum. SmartMedia Tools enable practitioners to begin with generic graphic elements and to gradually enrich SmartMedia with domain-specific vocabularies and relationships using graphical refinement, naming, prototype definition, and attribute refinement.

KEYWORDS: Domain-specificity, End user modifiability, Generic Applications, Incremental Formalization, Tailorability, Task-specificity

INTRODUCTION

Many HCI researchers have advocated the benefits of domain or task-specific systems over generic software applications [4, 11, 14]. Domain-specific systems embody a model of the entities to be manipulated and the tasks to be performed and use the model to provide active support to the user. An example is a kitchen design environment enabling users to construct floorplans from objects such as cabinets and stoves, which can then be analyzed by the system for compliance with design guidelines [6].

Generic applications like word processors, graphics packages and databases do not focus on a particular task or domain. Instead, these applications support the creation and manipulation of a particular type of representation, such as documents or drawings, by providing a wide range of formatting features and flexible editing commands.

Domain-specific software is believed to be more useful and usable than generic software because users interact with familiar entities and do not need to build up domain entities from other lower-level operations [6, 10, 11, 14]. However, recent empirical studies indicate that these intuitions and even the dichotomy between generic versus domain-specific may be too simplistic.

Nardi and Johnson found that domain-specific software is often too rigid and limited to support the rich set of tasks that professionals need to perform [12]. In their study, professional slidemakers preferred collections of generic graphic tools to slide making-specific software because of the generic tools' greater power and flexibility. However, preferences for generic, domain-specific, or a combination of both types of tools were highly dependent on the user's specific goals, which varied widely.

Sumner studied the use of collections of generic applications by design communities [18, 19]. She found that practitioners were using generic applications in very domain-specific ways. Over time, design communities created graphic vocabularies for expressing important domain concepts and well-defined representations for making important concepts and relationships visible. She observed that the flexibility and formatting features of the tools enabled designers to continually evolve their vocabularies and representations to better support changing work practices. However, her analysis showed that the tools' generic nature had several negative side effects such as introducing cognitive and manual burdens on constructing and maintaining designs and hindering iterative design.

As these findings indicate, the question of whether generic or domain-specific software is better suited for many areas of professional practice is too simplistic. In fact, *what many practitioners need are tools that bridge the gap between these two extremes by combining elements of both.* Specifically, tools need to provide: (1) the rich, flexible formatting and editing features associated with generic tools, (2) the supportive functionality offered by domain-specific tools, and (3) computational mechanisms that support bridging the gap between these two endpoints.

Towards this end, we have created SmartMedia Tools enabling practitioners to work continuously along a

spectrum of generic (flexible) to domain-specific (supportive) graphic representations. SmartMedia enables practitioners to begin with generic graphic objects and to gradually enrich their tool with domain-specific vocabularies and relationships.

This paper begins by reviewing empirical findings to derive design requirements for SmartMedia Tools. Next, a scenario illustrates how SmartMedia enables practitioners to enrich generic tools with domain-specific concepts. Finally, we compare this approach with other work in the areas of end user tailoring and incremental formalization and discuss the implications of SmartMedia for group work practices.

HOW TO BRIDGE THE GAP? DERIVING REQUIREMENTS BY EXAMINING PRACTICES

We are specifically interested in providing tools to support designers working in domains such as user interface design and software design. In our studies of three such design communities, workplace observations, interviews, videotape analyses, and analyses of design representations and tools, were used to understand long-term changes in design practices and representations [18, 19]. The major result revealed how design communities gradually construct and evolve their domain over time by defining important domain objects, creating and evolving representations for viewing these objects, and establishing relationships between objects and representations. The observed design process was labeled *domain construction*.

Domain construction both illustrates the need to bridge the gap between generic and domain-specific systems and offers insights into how systems might achieve this. Key findings from these studies – designers gradually evolve specialized representations, domain objects at different levels of explicitness are intermingled, evaluating and transforming design representations is difficult – have determined requirements for tools to support the observed domain construction process. These findings are reviewed here to motivate design requirements for SmartMedia Tools.

Finding #1: Designers gradually evolve specialized representations.

Similar to others [2, 9], we observed that designers rely heavily on diagrammatic, pictorial, and other forms of visual representation (see Table 1). However, standard or pre-defined representations were not used. Rather, these designers considered communicating the evolving design to other design stakeholders as a crucial part of their job and continually tried to define new and improve existing representations to enhance the communication process. As a result, all communities ended up creating specialized design representations tailored to their particular needs (see Table 1).

These representations evolved as designers continually refined their form and their content. What was particularly striking was the interplay between refining form and content; the two were inseparably interwoven. Often, designers would begin to refine parts of the representation by changing graphical elements such as font, color, shape, or position. These refinements served to make previously implicit or tacit information visible and explicit to themselves and other stakeholders.

Table 1: Evolution of domain representations and objects in three user interface design communities.

Comm -unity	Evolution of Representations	# Objects Graphic	Explicit	Formal
1	Unstructured text	3	0	0
	Structured text +	9	6	5
	Tables	7	5	0
2	Graphic templates +	4	0	0
	Hypermedia outline	5	5	4
3	Text +	5	5	0
	Simple tables	5	4	0
	Initial flow charts +	5	4	0
	Complex tables	13	13	13
	Later flow charts +	9	7	0
	Complex tables	15	14	14

Some of these graphical refinements concerned the structure of representations. For instance, community 1 started out using unstructured text design representations. This representation eventually became very structured as graphical delimiters and other markings were introduced that later evolved into explicit subcomponents.

Many of these graphical refinements changed the content of representations by introducing new domain objects or modifying existing domain objects. Typically, new representations contained few, if any, explicitly represented domain objects. Over time as stakeholders interacted, these representations became populated with more domain objects, often depicted at greater levels of detail. As an example, the initial flow chart representation used by community 3 depicted only five domain objects, while later versions of the flow chart representation depicted nine objects (see Table 1, # Objects Graphic column). What was particularly interesting was the process by which domain objects came into being. Rarely did designers decide they needed a "menu" or some other object and then proceed to define that object in detail. Instead, domain objects and their parts emerged gradually while refining existing representations.

Figure 1 shows three stages in the graphic evolution of a voice menu representation used by community 3. The representation on the left is a "simple phrase table" used to represent any system action, not just voice menus, that expect a user response. Font size, font style, and position are used to distinguish parts of a menu such as the title, phrase, possible actions, and expected system responses. The representation in the middle shows the first pass at a graphic representation specifically for voice menus. This node from the initial flow chart representation distinguishes parts such as identifiers (PO.01), titles, and legal commands (1, security code). The representation on the right shows a refined voice menu representation. New parts have been reintroduced (i.e., the phrase) and parts have been graphically and semantically refined (i.e., the shaded title indicates the title is spoken in the interface).

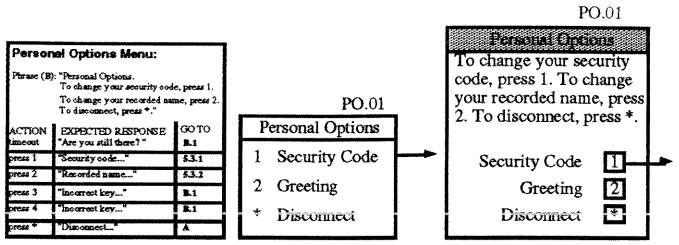


Figure 1: Graphic evolution of a voice menu representation.

These observations indicate that tools providing a canned model of a particular type of design representation could not accommodate the changing communication needs of these communities. Instead, tools need to provide the flexible editing commands and rich formatting features typically associated with generic applications to support the observed *graphic refinement* process.

Finding #2: Objects at different levels of explicitness are intermingled.

Graphic refinement was not the only way these designers evolved their representations. Sometimes additional steps were taken to make the objects present in the representations more explicit. The "# Objects" column in Table 1 identifies three levels of explicitness found in the design representations. As discussed in the previous section, the refined domain objects minimally had a distinguishable graphic look (see "graphic" column).

Many times after graphically refining an object, designers would also create a name for the object. Naming objects served communication needs by helping to ensure that stakeholders were, literally, referring to the same graphic object. Some names were implicit (i.e., only used in conversation) and some were explicit. By our definition, names are only explicit when there is some written key or other record explaining that items with a particular look and parts have a specific name. Explicit names helped ensure consistent interpretation of design representations by all design stakeholders.

In some cases, a subset of the named, explicit objects were formalized even further (see "formal" column). Shipman defined formalization as the process of "identifying machine-processable aspects of information" such as types, attributes, and values [17]. In some cases, designers had enriched their tools with awareness of some of their classes or types of domain objects by defining (1) name <-> graphic look mappings and (2) part <-> whole relationships (i.e., attribute <-> class relationships). One benefit of such formalization was that it promoted the consistent production of design representations by allowing previously defined objects to be reused when creating new representations. However, this benefit only seemed to be taken advantage of when the system being used made named classes directly available as reusable palette items; i.e. integrated into the system's standard generic tool selection instead of being in a separate toolbar or area.

We also observed that even after objects had been formalized (made machine-processable), their evolution was not over. Designers continued to refine both the look and content of formalized domain objects by adding new attributes, modifying existing attributes, and removing attributes. As the voice menu example shows (Figure 1), evolution was not a simple, linear march towards progressively elaborate objects. Many steps focused on simplifying existing representations by removing or refining existing attributes; e.g., the "phrase" attribute in the table representation (left) did not appear in the initial node representation (center).

These observations point to several mechanisms that tools must provide to enable practitioners to bridge the gap between generic and domain-specific systems. First, systems must allow practitioners to *name classes* of domain objects. Second, defining classes should follow the *prototype definition* model. In this object model, an existing object instance is used as a template for creating the desired class. Third, once a class has been created, it should be available for reuse in the system's standard palette. That is, systems should support *mixed palettes* intermingling objects at different levels of explicitness. Finally, systems need to provide an underlying object model that is flexible enough to allow already defined and used object classes to continue to undergo *attribute refinement*.

Finding #3: Evaluating and transforming design representations is difficult.

Designers did not simply construct representations, they also evaluated representations to see if various design or project goals were being met by the artifact being constructed. Evaluation refers to the activity of analyzing a representation to see if it conforms to various criteria and constraints [1]. For instance, the designers in community 3 continually evaluated their design representations for compliance with existing user interface guidelines. These designers used a combination of visual inspection and mental simulation to aid in their evaluation process. For experienced designers or small designs, these practices worked fine. However, as designs grew large and complex, relying on visual inspection became increasingly problematic as important features and relationships became difficult to spot. Also not all designers are equally experienced; many newly hired designers lacked detailed knowledge of the interface guidelines thus did not know some of the relevant criteria. Thus, as noted by others [1, 8], relying solely on experience and practices is fraught with potential error and designers need tools that support their evaluation activities.

The basic design process followed by these communities was construct-evaluate-iterate. In all communities, a single design representation was insufficient to support this process (see Table 1). As noted by Norman, a good representation emphasizes the important objects and relationships and de-emphasizes the less important things [13]. However, there are many important aspects of any particular design and no single representation can show them all equally well. Thus, these communities created multiple representations that made different aspects of the design more visible and readily apparent. Besides serving visual inspection and evaluation activities, multiple

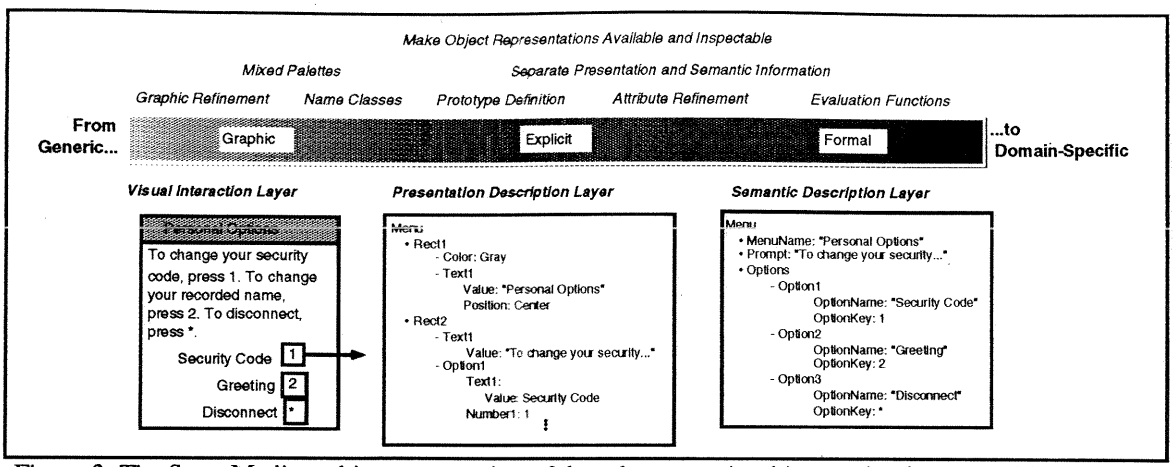


Figure 2: The SmartMedia architecture consists of three layers: a visual interaction layer providing standard drawing tools, a presentation description layer, and a semantic description layer. Special editors are provided for modifying object descriptions.

representations were sometimes created to facilitate iterative, collaborative design. For instance, the lead designer in community 2 created the hypermedia outline representation to make it easier to modify the textual parts of the design during collaborative design sessions with other stakeholders.

However, multiple representations created a transformation burden; i.e., members in all communities reported that going back and forth across the representations was hard work, both manually and cognitively. The problem stemmed from the highly interrelated nature of the representations. These representations shared many of the same domain objects; thus making changes in one representation required making changes in the related representation to maintain design consistency. Two design communities reported that this transformation burden negatively affected their iterative design practices. The problem was that the tools used by the designers did not support them at all in either locating related objects or maintaining consistency across representations. This lack of support occurred both when using different tools to create the different representations *and* when using the same tool to create the different representations.

The design communities we observed were all using generic software applications to create their design representations. Earlier, we saw that these tools offered many positive affordances such as graphical refinement. Some tools went a little further and supported designers to enrich tools with some awareness of important domain objects using techniques such as naming, defining classes, and mixed palettes. For the most part, where tools offered this functionality, designers took advantage of it. However, none of the graphics packages used by these communities supported this level of enriching. As discussed previously, all these tools broke down when it came to supporting aspects of their practices such as evaluation and transformation activities. These design communities could have benefited from many of the active support mechanisms associated with domain-specific tools.

Specifically, tools must enable practitioners to take the next step and further enrich their domain objects with *behaviors and evaluation functions* to support mental simulation and evaluation activities. This requires *making the domain object definition available* for inspection and modification. As the transformation examples indicate, the domain object definition also needs to be accessible to other tools. However, since the essence of the transformation process is to change the visual representation of aspects of the domain objects, *presentation information needs to be separate from semantic information.* This separation is necessary to support graphic refinement processes on the same domain object in multiple representations.

In summary, we have enumerated several design requirements for systems to bridge the gap between generic applications and domain-oriented systems. The following two sections will show how these requirements are reflected in the design and use of SmartMedia tools.

SMARTMEDIA TOOLS: TOWARDS SYSTEMS THAT ARE BOTH FLEXIBLE *AND* SUPPORTIVE

We have developed a series of SmartMedia Tools [15] combining positive aspects of both generic and domain specific applications. The goal is to make systems that are both flexible and supportive by turning the current dichotomy of generic versus domain-oriented systems into a seamless continuum. Towards this end, SmartMedia Tools embody a specialized architecture and corresponding tools for refining domain objects both graphically and semantically (see Figure 2).

In generic graphics applications like MacDraw[1] and flowcharting tools like Inspiration[1], graphics objects can be accessed and manipulated only at the visual interaction layer using direct manipulation. Similar to these applications, SmartMedia Tools also provide a visual interaction layer supporting the direct manipulation of graphic objects. However, graphic objects in SmartMedia tools have two additional user-accessible representations: a system-supplied textual description of the object's presentation in the interface (the "presentation description") and a user-definable textual description of the object's meaning in the domain

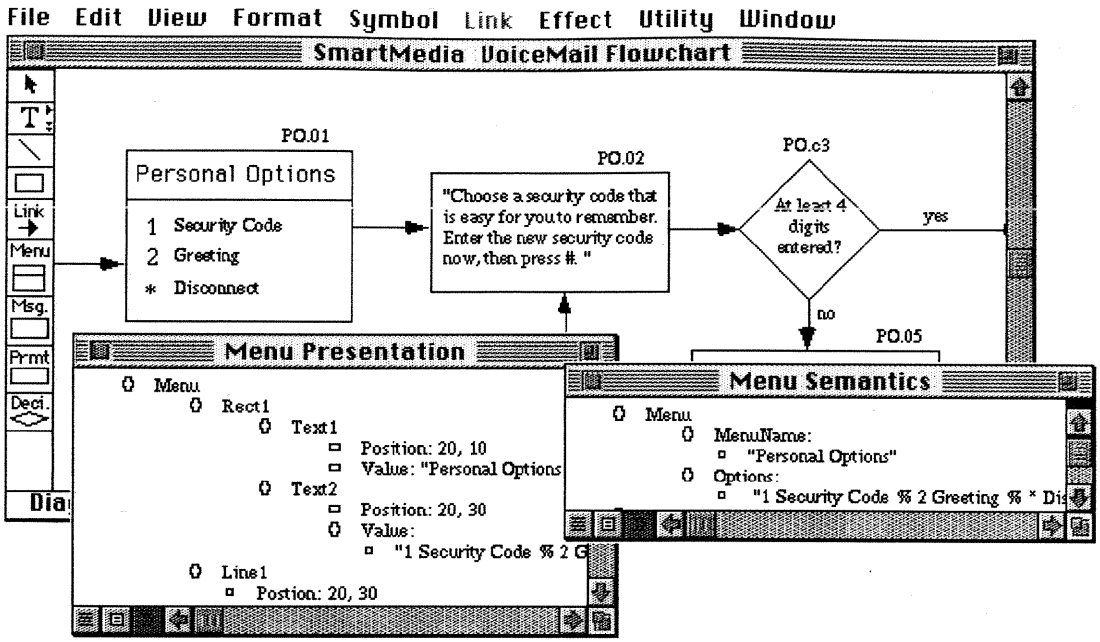


Figure 3. SmartMedia Flowcharting tool after initial tailoring (i.e., graphic to explicit transition).

(the "semantic description"). The presentation description summarizes the visual properties of objects (e.g. shape, size, color) and the semantic description specifies the domain-specific meaning of a graphic object (e.g. that a number represents a menu command). This description is saved for use by other applications in an open, object-oriented database.

Special editors are provided that enable users to inspect, extend, and modify the object descriptions directly. Relationships between parts of the presentation and semantic descriptions, such as evaluation functions, can be defined using spreadsheet-like functions. The representations are integrated in that changes in any of the object representations are reflected in all other object representations. Thus, changes performed by direct manipulation in the visual interaction layer are mirrored in the object's presentation description. At any time, users can select a graphic object in the visual interaction layer and define a class based on its properties. Once defined, objects are available in the palette for future reuse.

The SmartMedia architecture enables SmartMedia Tools to seamlessly support the transition of domain objects from graphic to explicit to formal objects. SmartMedia Tools meet the requirements developed in the previous section in the following ways:

- Graphical refinement is supported by offering the general functionality of graphics packages and a palette supporting the reuse of previously defined domain objects.
- Naming and the incremental formalization of graphic objects is supported by prototype definition and the integrated presentation and semantics object editors.
- The definition of user-defined analyses and transformations functions is supported through spreadsheet-like functions as well as by making the semantic object descriptions open and accessible to other external applications.

In the following section, we will use a fictional scenario to illustrate this functionality in a concrete context.

SCENARIO: USING AND EVOLVING SMARTMEDIA

In this scenario, we use voice dialog design as our example domain. Voice dialog applications are systems with phone-based interfaces such as voice mail systems. Our scenario is informed by Sumner's empirical study of designers working in this domain [19]. She described how designers of voice dialog applications continually adapted and improved their design representations. Here we will discuss how a SmartMedia Tool could have supported these designers to develop and evolve domain-oriented design support tools in parallel with their design representation.

Stella is a designer of phone voice dialogs at a major phone company. Until recently, she was using a textual description of the voice dialog interface as her design representation. However, this textual representation was getting unworkable for the larger designs she is working on now. From a previous job appointment, Stella has some experience using flow charts as a design representation. Stella uses a SmartMedia flow charting tool for experimenting with the new representation.

From implicit drawings to explicit design objects

Initially Stella uses the generic flowcharting functionality of the tool to produce her designs; i.e., she uses the provided boxes, ovals and arrows as her design objects. Doing these designs, she experiments with using different shapes for different interface components. Over time, she gains confidence that flowcharts are an appropriate

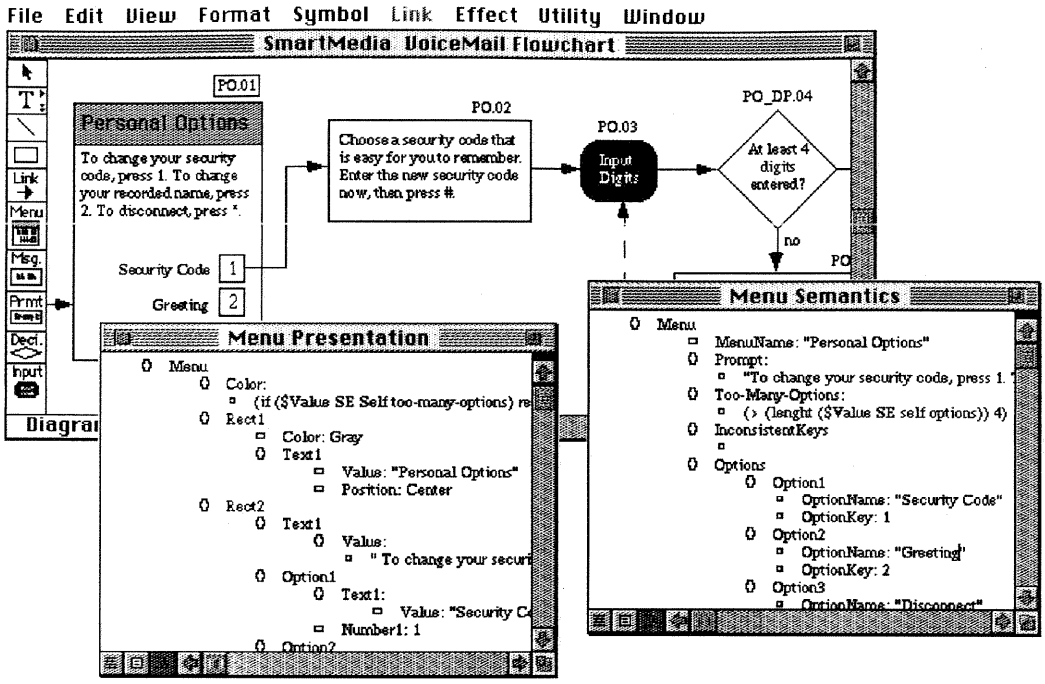


Figure 4. SmartMedia tool after more tailoring (i.e., explicit to formal transition).

representation for the design of voice dialogs. Stella inspects her previous designs and detects that she is implicitly distinguishing domain-specific entities like voice menus, messages, and decision points. While doing the designs she was relatively consistently using different visual formats for each of these concepts, but she was not explicitly classifying the objects or labeling their subcomponents. Stella decides to adapt the tool to reflect the specific objects and operations she has evolved to better support her design practices (see Figure3).

When creating new objects, Stella has to minimally define the presentation description and, optionally, the corresponding semantic description. For example, in her previous designs, Stella had graphically distinguished that voice menus had three subcomponents or parts: a menu name, a prompt, and options. She uses a voice menu from an existing design as a prototype for presentation description part of the class definition (see Figure 3, top left and bottom left). Parallel with the presentation description, Stella also defines the corresponding semantic description. She specifies that 'menu' objects have attributes like the menu name, the prompt and options (Figure 3, bottom right). The newly defined object is added to the system palette (Fig. 3, top left) and can now be used to create future designs more consistently.

From explicit to formal design objects

During the following weeks, Stella uses the newly defined objects for producing her designs. She also discusses her new design representation with her colleagues, customers and sub-contractors. In response to various problems and suggestions, Stella iteratively evolves and refines the presentation and the semantic description of the domain objects, and she also introduces the 'input' object as a new palette item. The result of this transition from explicit to formal domain objects is shown in Figure 4. The graphic look of domain objects has been elaborated to show more information in a way that emphasizes the important elements.

Also the semantic description has also been elaborated on further. The semantic information about the menu has become more structured through the introduction of new attributes (see Figures 3 and 4). Also new computed attributes were introduced in the semantic description (e.g., the too-many-options attribute) and the presentation description was extended using spreadsheet functions so that changes in the semantic description are automatically reflected in the interface of the visual interaction level.

DISCUSSION

In the previous scenario we have shown how a designer extended and adapted a SmartMedia tool that initially only offered generic flowcharting functionality to give domain-specific support in the context of voice dialog design. The domain-specificity resulted from the definition of palette items which are customized to reflect important concepts in the domain – visually (through the visual interaction layer), semantically (through the semantic description layer), and computationally (through evaluation functions and transformation by external applications). While the empirical evidence presented earlier indicates the validity of this approach, several important questions need to be considered:

- How is the SmartMedia approach different from existing forms of end user tailoring found in other commercial and research applications?
- Will practitioners tailor in the way we have described?
- Will providing this flexibility negatively impact group work practices by promoting divergent practices?

Comparison to other approaches

We use "tailoring" to refer to the entire range of adaptations supported by SmartMedia; i.e., from graphical refinement to the definition of evaluation functions. This approach to end-user tailoring is different from the kind of tailorability that can be found in applications like Word[1], Emacs, and Excel[1]. This should not be too surprising because SmartMedia tools focus on supporting the definition of domain-specific *graphical* design representations. These applications are focused on document production and tabular analysis; to our knowledge, there are no commercial graphical applications supporting similar levels of end-user tailorability. Thus SmartMedia tools can be seen as a proposal on how graphic applications should be made end-user-tailorable. Still there exist similarities, as well as differences in the tailoring facilities provided.

A key difference with commercial applications is SmartMedia's focus on enriching tools with awareness of *domain objects*. Word, like SmartMedia tools, supports the naming of objects (i.e. paragraphs) and the mapping of names and visual appearance through the definition of "styles." This, however, is all the domain-oriented information that can be attached to a paragraph. Word does not enable users to specify that domain objects may have multiple subparts or to associate domain-specific behaviors or functions with defined objects. Emacs is another example of an editor that offers many opportunities for tailoring. However the approach to customization is very different from the one used in SmartMedia tools. Customization in Emacs consists of the re-definition of standard functions and the definition of new functions. Emacs does not support the naming of objects, nor does it support the creation of user-defined palettes. Excel and other spreadsheets support the naming of cells and regions and the definition of functions that use this information. However in spreadsheets, only instances can be named, not classes of objects. Thus, palettes of domain objects cannot be created and there is no support for creating design representations by combining reusable building blocks.

Several researchers have investigated end-user modification and programming in the context of building more flexible design environments [3, 5, 17]. Modifier provides a special interface and knowledge-based assistance to support designers to add new domain objects to their design environment [5]. Programmable design environments provide a domain-enriched dialect of Scheme that enables designers to define new domain objects and operations on the objects [3]. Both approaches differ from SmartMedia in that they require designers to explicitly formalize domain knowledge; i.e., render it machine-processable, before it can be used. That is, they do not enable designers to *incrementally formalize domain objects during use.*

The most similar approach is that of Shipman [16, 17], who pioneered many ideas and techniques associated with incremental formalization. In many respects, we follow in the footsteps of these ideas. However, there are important differences in both emphasis and approach. Shipman emphasizes using incremental formalization to support hypermedia knowledge-base evolution, not the development of domain-specific graphic design representations. He has focused more on providing computational mechanisms that scan for either textual or graphical patterns and suggest possible formalizations to the user. We have focused more on looking at how people *incrementally formalize in practice* and providing functionality that supports these practices. We feel that these two approaches are complementary and that, ultimately, both are needed.

Will practitioners tailor as described?

In the end, this is an open question to be answered by observing SmartMedia in use. However, our studies of design communities indicate that there is a need for this type of tailoring and that *forms of it are already occurring* to some extent. We also include two other observations as promising indications. First the motivation seems to already be present as we observed how these communities continually strive to improve upon their existing practices, and in many cases also their tools. Second, all three design communities had individuals who had taken on local developer roles [7]; thus an organizational infrastructure is already present to support further tailoring activities.

Impact on group work practices

Many people are concerned that providing such extensive tool tailorability will negatively impact group work practices. In many groups and organizations, smooth functioning depends on shared conventions, tools, and practices. A primary concern is that practitioners will tailor their tools in radically individualized ways and this will lead to diverging, and even chaotic, work practices and working environments. These are valid concerns that, again, will not be fully understood until we have observed SmartMedia tools in use over a long period of time. However, we believe that this is an unlikely outcome; that instead, tool tailorability promotes convergence in work practices. For instance, Trigg and Bodker studied a work group tailoring Word Perfect to support their work practices [20]. They found that over time, systematization emerged as the group took advantage of the tool's tailorability to co-evolve their tools and work practices. As another example, the three design communities we observed were using generic off-the-shelf applications which offered hundreds of features and many forms of customization. These communities had plenty of opportunities to diverge. However, they did not. Instead *we saw convergence*, as they enriched their tools and evolved practices to promote the production of shared, standardized design representations.

IN SUMMARY

We observed that the dichotomy between generic or domain-specific software is too simplistic for many areas of professional practice. Instead, many practitioners, such as designers, need SmartMedia Tools enabling them to work continuously along this spectrum. An analysis of three design communities was used to derive design requirements for SmartMedia tools and a scenario was used to illustrate how SmartMedia features fulfill the stated requirements. SmartMedia Tools enable practitioners to gradually enrich generic graphics tools with domain-specific knowledge to support the creation and analysis of specialized design representations. During tool use, generic graphic objects are

incrementally formalized through processes of graphic refinement, naming, prototype definition, and attribute refinement to better support evolving work practices.

ACKNOWLEDGMENTS

We thank the L3D group at the University of Colorado for providing a rich history of thinking on these issues and a ready forum for further discussions. We particularly thank Stefanie Lindstaedt, Jonathan Ostwald, and Kurt Schneider. This research was supported by ARPA under grant No. N66001-94-C-6038.

[1] Product Credit and trademark notifications for the products referred to are given here: Excel and MS Word are registered trademarks of the Microsoft Corporation. MacDraw is a registered trademark of the Claris Corporation. Inspiration is a registered trademark of the Inspiration Software Corporation.

REFERENCES

1. Bonnardel, N., "Criteria Used for Evaluation of Design Solutions," Designing for Everyone and Everybody (Proceedings of the 11th Congress of the International Ergonomics Association), Paris (July), 1991.

2. Curtis, B., H. Krasner and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, Vol. 31, pp. 1268-1287, 1988.

3. Eisenberg, M. and G. Fischer, "Programmable Design Environments: Integrating End-User Programming with Domain-Oriented Assistance," *Human Factors in Computing Systems (CHI '94)*, Boston, MA (April 24-28), 1994, pp. 431-437.

4. Fischer, G., "Domain-Oriented Design Environments," in *Automated Software Engineering*, Ed., Kluwer Academic Publishers, Boston, MA., 1994, pp. 177-203.

5. Fischer, G. and A. Girgensohn, "End-User Modifiability in Design Environments," *Human Factors in Computing Systems (CHI'90)*, Seattle, WA (April 1-5), 1990, pp. 183-191.

6. Fischer, G. and A. C. Lemke, "Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication," *HCI*, Vol. 3, pp. 179-222, 1988.

7. Gantt, M. and B. Nardi, "Gardeners and Gurus: Patterns of Cooperation Among CAD Users," *Human Factors in Computing Systems (CHI '92)*, Monterey, CA, 1992, pp. 107-117.

8. Guindon, R., "Requirements and Design of DesignVision, An Object-Oriented Graphical Interface to an Intelligent Software Design Assistant," *Human Factors in Computing Systems (CHI '92)*, Monterey, CA (May 3-7), 1992, pp. 499-506.

9. Guindon, R., H. Krasner and B. Curtis, "Breakdowns and Processes During the Early Phases of Software Design by Professionals," in *Empirical Studies of Programmers: Second Workshop*, G. Olson. S. Sheppard and E. Soloway, Ed., Ablex Publishing Corporation, Norwood, New Jersey, 1987, pp. 65-82.

10. Lewis, C. and G. M. Olson, "Can Principles of Cognition Lower the Barriers to Programming?," *Empirical Studies of Programmers: Second Workshop*, 1987, pp. 248-263.

11. Nardi, B. A., *A Small Matter of Programming*, The MIT Press, Cambridge, MA, 1993.

12. Nardi, B. A. and J. A. Johnson, "User Preferences for Task-specific vs. Generic Application Software," *Human Factors in Computing Systems (CHI '94)*, Boston, MA (April 24-28), 1994, pp. 392-398.

13. Norman, D. A., *Things That Make Us Smart*, Addison-Wesley Publishing Company, Reading, MA, 1993.

14. Repenning, A. and T. Sumner, "Agentsheets: A Medium for Creating Domain-Oriented Visual Languages," *IEEE Computer (Special Issue on Visual Programming)*, Vol. 28, pp. 17-25, 1995.

15. Schneider, K. and M. Stolze, "SMArT CASE: Supporting Co-Improvement of Process, Tools, and Notations," *Submitted to: International Conference on Software Engineering (ICSE-18)*, Berlin, Germany (March 26-30), 1996.

16. Shipman, F. M., C. C. Marshall and T. P. Moran, "Finding and Using Implicit Structure in Human-Organized Spatial Layouts of Information," *Human Factors in Computing Systems (CHI '95)*, Denver, CO (May 7-11), 1995, pp. 346-353.

17. Shipman, F. M. and R. McCall, "Supporting Knowledge-Base Evolution with Incremental Formalization," *Human Factors in Computing Systems (CHI '94)*, Boston, MA (April 24-28), 1994, pp. 285-291.

18. Sumner, T., "Designers and their tools: Comparing two models of design support systems," University of Colorado at Boulder, Ph.D. dissertation, Dept. of Computer Science, 1995.

19. Sumner, T., "The High-Tech Toolbelt: A Study of Designers in the Workplace," *Human Factors in Computing Systems (CHI '95)*, Denver, CO (May 7-11), 1995, pp. 178-185.

20. Trigg, R. and S. Bodker, "From Implementation to Design: Tailoring and the Emergence of Systemization in CSCW," *Conference on Computer Supported Cooperative Work (CSCW '94)*, Chapel Hill, North Caroline (October 22-26), 1994, pp. 45-54.