Model-Based Automation of the Design of User Interfaces to Digital Control Systems

Robin R. Penner and Erik S. Steinmetz

Abstract—**Digital control systems, like those controlling the functions of buildings or industrial processes, pose a number of special problems for good user interface design. The general problems of providing usability, common to all systems, include difficulty in accessing and applying principles of good design. In addition, digital control systems can have multiple users, with multiple roles, and each installation has different configurations of systems, controls, and user interface devices. Providing interactions for the users of building control systems is often achieved by manually implementing each required display. This is an expensive solution, which often produces less than optimal results. We address these problems through the automation of user interface design. Our solution, called DIGBE (Dynamic Interface Generation for Building Environments), separates domain knowledge, interaction design, and presentation heuristics into multiple collaborating models. Each model contains knowledge about a particular aspect of interface design, and uses this knowledge to dynamically create each user interface that is needed to support the users of a control system. DIGBE demonstrates that it is possible to automatically and dynamically create consistent and individualized user interfaces from model-based design knowledge.**

Index Terms—**Building management systems, cooperative systems, design automation, user interfaces, user modeling.**

I. Introduction

In the day-to-day operations of a building, digital control systems provide numerous automated and semi-automated distributed functions, including heating, cooling, ventilation, access control, and security. Human users may need to interact with the components of each function. The building manager may interact with the system by setting desired comfort and consumption ranges (room and zone temperatures, electricity usage), entering or modifying occupancy schedules, and managing access codes and user databases. Other users of the building systems, like building engineers, security guards, or occupants, perform some of these management functions, and additionally operate equipment (such as escalators and thermostats). Technicians who install and service the building management system interact with the digital control system to install, set-up, fine tune, diagnose, and repair functional components. Other technicians may modify configurations and install replacement parts and systems, or add new functionality. In specialized or complicated buildings, other user roles and tasks might be required, like scheduling operating rooms or maintaining sterile environments.

Manuscript received August 17, 2000; revised October 19, 2001.

The authors are with the Iterativity, Inc., Minneapolis, MN, 55404 USA (e-mail: robin@rpenner.com).

Publisher Item Identifier S 1083-4427(02)01481-9.

Each of the interactions with the building automation systems must be provided through user interfaces between the automation functions and the user. Current systems largely provide these user interfaces through individual single-function controls (thermostats, valves, keypads) and computer stations. The computer stations may be dedicated displays, with data tables and mimic displays (diagrams that mimic the organization of system components), or they may be multi-use PCs with specialized programs providing scheduling spreadsheets, access code databases, or configuration applications. Integration of functions, components, or data is not widespread in building automation systems; user interfaces for each system typically do not provide the same interface styles or interaction designs. Switching between these different user interfaces often causes errors and user confusion. In addition, the usability of many of the user interfaces that are provided is poor, and many are outdated, resulting in difficulty accessing and using the functions that users need from the building automation systems. User interfaces to individual functions and systems are hard coded, and so may not get updated when additional automation functions are added or when system components change.

Buildings, by their very nature, are unique. This uniqueness presents additional difficulties when attempting to create good user interfaces to the building's distributed control systems. Each building has a different set of users, tasks, systems, equipment, and requirements. Small buildings differ from large buildings, forced air systems differ from boiler systems, and automated security installations differ from buildings that are staffed with human security guards. Even within similar buildings with similar systems, the components of each particular system will not be the same, and the environment within which they operate and the special parameters required for individual installations and operations cannot be predicted in advance. There will be differences in the number of floors, the layout of each floor, available ductwork, system placement, and system distribution, which make it necessary to individually create any mimic displays that are required. This is an expensive procedure; hand crafting of individual displays for building management systems generally accounts for an average of one third of the total installed cost of a new building management system.

Research in the last several decades has contributed to the development of a solid practice of usability engineering, and has clearly demonstrated that the design of the user interface to an automated system determines the ability of humans to use it (see, for example, [1] and [2]). Processes and artifacts for performing user interface design have been successfully developed over the last several decades (e.g., [3]–[5]; surveyed in [6]), and

integrated user interface design and testing software is readily available. In practice, however, fully usable systems often remain both undefined and unrealized. Uniformly good user interfaces to complex systems have been difficult to achieve for many reasons, including the need for specialists to perform the design work, rapidly changing technology which increases interaction options, difficulty quantifying procedures or results, and the difficulty inherent in producing useful specifications, standards, or guidelines. In addition, when standards and processes do exist, there is often little management support for their application by software developers. User interface design is very often not considered an integral part of the development process until late in the cycle, and the separation between user interface designers and application implementers can compound the problem.

Even when performed under optimal conditions by experienced and talented professionals, user interface design as it is currently practiced may no longer be sufficient to meet the complex needs of the users of current control systems. Increases in computing power and in the prevalence of distributed computer systems make it difficult to predict in advance which tasks and what information users will need when they interact with such systems. This makes it difficult to design a user interface or interaction structure in advance to suit the user, the task, the information or objects involved in the task, the application or system providing the task functions, and the device and operating system the user has chosen to embody the user interface. It may be impossible to predict the interactions that will be required for a particular system and user, the information that will be necessary, or the information that will be available. It will also be impossible to predict or control the hardware and software capabilities that will exist in local and distributed forms, or the experience and capabilities of the human participants.

In other work ([7]–[9]), we describe how we used usability principles and processes to cognitively engineer (through task and scenario analysis, integrated user involvement, and functional prototyping with iterative development) a user interface design framework for building managers, operators, and technicians, including full design specifications for environmental and security domains. This system provided a general, tested user interface framework for building management systems hosted on PCs, including specification of the displays required for each role and task. Since the components and functions required at each installation are different, general rules for widget selection and layout, culled from the literature, were detailed and illustrated in a style guide [10], available to application developers as a paper document and as an electronic hypertext document.

This effort would have been sufficient (at least, until it needed to be revised as the state of the art in user interface design progressed) if the applications that were being designed were not required to differ based on system configuration or user role. Each special case could not be addressed within the limits of a style guide, resulting in the requirement for implementers to make design choices, often to the detriment of the "user experience". In addition, this solution does not provide integration between other building systems and the environment and security systems, and does nothing to alleviate the costly burden of creating specialized displays and controls for each installation. To explore other approaches, we undertook a five-year research program to explore and prototype automated solutions to providing user interfaces to complex digital control systems. This program had several explicit goals, including the following.

- Reduce the need for user interface software development efforts.
- Maximize the appropriate application of user interface knowledge.
- Minimize end user and design engineer configuration requirements.
- Provide for multiple types of functionality, data, and users.
- Support reuse of user interface designs.
- Support multiple types of hardware devices.
- Support evolution in control systems, user functions, user interface design knowledge, and user interface devices and interaction paradigms.

We hypothesized that a system with multiple collaborating models could be constructed, capable of automatically designing, presenting, and managing the interface between a user and a control system. To test whether such a system would meet our goals, we designed and iteratively created an operational prototype called DIGBE (Dynamic Interaction Generation for Building Environments [11]). We used the tested user interface framework and style guide that we had developed for building management as the basis for the knowledge that our system contains. DIGBE is a fully functional system, based on solid design principles, for *dynamically* and *adaptively* managing the basic tasks of building management (including configuration, monitoring and control of security and environmental systems, management of users, and data analysis), through the collaboration of three models. The **Domain Model** provides the semantic basis for interaction design, allowing control system independence and an object-oriented representation of control system components. The hierarchical, task-based **Interaction Model** provides the design knowledge required to design appropriate tasks and interactions, allowing dynamic application of the principles of usability engineering. The **Presentation Model** converts interaction designs into user interfaces, allowing hardware and operating system independence.

II. Related Research

Researchers and practitioners have attempted to address the problems of automating usability engineering for several decades. The popularity of graphical user interfaces in the 1980s spurred an emphasis on user interface management systems (UIMSs), which, it was hoped, would result in the ability to automatically design and present user interfaces. However, in the mid-1990s, industry and university researchers generally concluded [12] that automated user interface generation is too difficult, because it depends on human knowledge of task structures and domain requirements. Many of these UIMS researchers chose to concentrate on approaches in which automated, model-based systems provide a critique of designs developed (mainly) by human designers [13].

Despite the lack of emphasis on totally automated generation, the related work on computer-assisted design has had much to offer in the areas of user and task modeling, and also informs the issues of separation of processes between domains, interactions,

and presentations. SAGE [14] generated (two–dimensional) 2-D static presentations of relational data, and emphasized the importance of multidimensional representations of data and the effect of user goals. BOZ [15] took a task analytic approach to interface automation, included some rules of composition that responded to situational parameters, and separated interaction content from user interface format. IBIS [16] had a separate presentation component, and included representation knowledge of users, tasks, and contexts. Later work on IBIS included an emphasis on data characterization [17] and hierarchical decomposition of visual lexicons [18].

Investigations in the area of user interface design environments (UIDE) included modeling of user interface design knowledge [19], separation of design environments into multiple layers for the domain, the interactions, and the presentations [20], and declarative user interface design models [21]. Mecano [22] used domain models to provide the underlying semantics for user interface design. Recently, computer aided design of user interfaces (CADUI) [23], [24] has been suggested as a more efficient emphasis than automated generation. CADUI provides incremental design aiding through a task-based (rather than a widget-model-based) mechanism, and includes an automatic presentation layer. Many researchers are investigating the various aspects of CADUI, contributing to the development of powerful tools to assist human usability engineers.

Driven by the high costs of delivering good user interfaces for each installation of a complex control system, some industrial practitioners have been developing pragmatic solutions to the problem. In the domain of manufacturing information systems, a generated user interface has been demonstrated [25] which combined user, task, and information modeling with templates to produce dynamic, adaptive interfaces for controlling automated manufacturing parameters. Our approach contained these elements, but also incorporated an adaptive model of the design of interactions for subtasks and interaction elements that can compose and modify themselves to suit different types of users, data, or task situations.

III. Dynamic Interaction Generation for Building Environments

The DIGBE prototype was developed in Java, and runs on most hardware platforms. There are two software components apparent to a user of a DIGBE building management system. First, a general DIGBE application is available as a standard desktop application on a multi-use PC. A user, generally a building manager, starts up that program, and is presented with a simple dialog that allows the user to identify the type and location of the control system database, the name of the system, and the system-level user's name and password. In response, DIGBE creates an executable to allow log-on to the control system. This executable is available as another application, titled with the name of the chosen control system. When a user executes this application, DIGBE presents a log-on screen requesting the user type and password.

After successful log-on, DIGBE designs and presents (in real time) the user interface appropriate to the task set of that user. Each of the tasks required by the user's role is presented as part of an integrated application, specialized to the user's role and access privileges. The user interface is also automatically adapted to best suit each specific task, the objects of interest, the interface device, and the value and type of data. As the interaction progresses, the system continues to dynamically adapt to these factors by designing, presenting, and managing the ongoing user interface. The DIGBE research implementation provides for three user roles (manager, operator, or technician), can interpret any building management control system database in its known formats, and can present fully functional building management user interfaces on either a CRT or a handheld user interface (simultaneously, if desired). It requires no pre-built user interfaces to any of its functions, but creates each interaction and interface as required.

Fig. 1 shows a screen shot of a DIGBE-generated user interface for a manager of a large building environmental control system. The basic tasks for this user are presented as panes of an application window, and include system navigation (left pane), system monitoring (top right) and detail data interactions (lower right). With these displays, the user is automatically presented with the display of (or access to) all necessary tasks and system components, through a consistent and "usability engineered" user interface, specialized to the particular building and combination of control systems.

DIGBE is able to provide a well-designed, specialized user interface for each situation, because it uses internal models of good design (as we currently understand them) to create and tune each interaction. DIGBE's knowledge is stored in object-oriented models, and is separated into domain information, information about user tasks and interaction requirements, and information about how to select and present interactions on different types of hardware. The models in DIGBE allow it to

- create an object-oriented **application model** (containing a representation of the site-specific digital control system), using a **domain model** (which defines an ontology of the objects, relationships, actions, user roles, data roles, and data types in the domain);
- create a representation of the **interaction design** to support a specific user's interaction with the control system, using an **interaction model** (which contains compositional rules for creating interactions to support task structures);
- format and present that interaction as a **user interface**, using a **presentation model** (which contains information-coding heuristics and hardware-specific widget selection information);
- manage and automatically update the representations (and therefore the user interface) in real-time, as the interaction progresses.

The remainder of this section provides a brief overview of the structure, function, and operation of each of the three main models in DIGBE.

A. Domain Model

The domain of building management is fairly well understood, and numerous domain models with information about

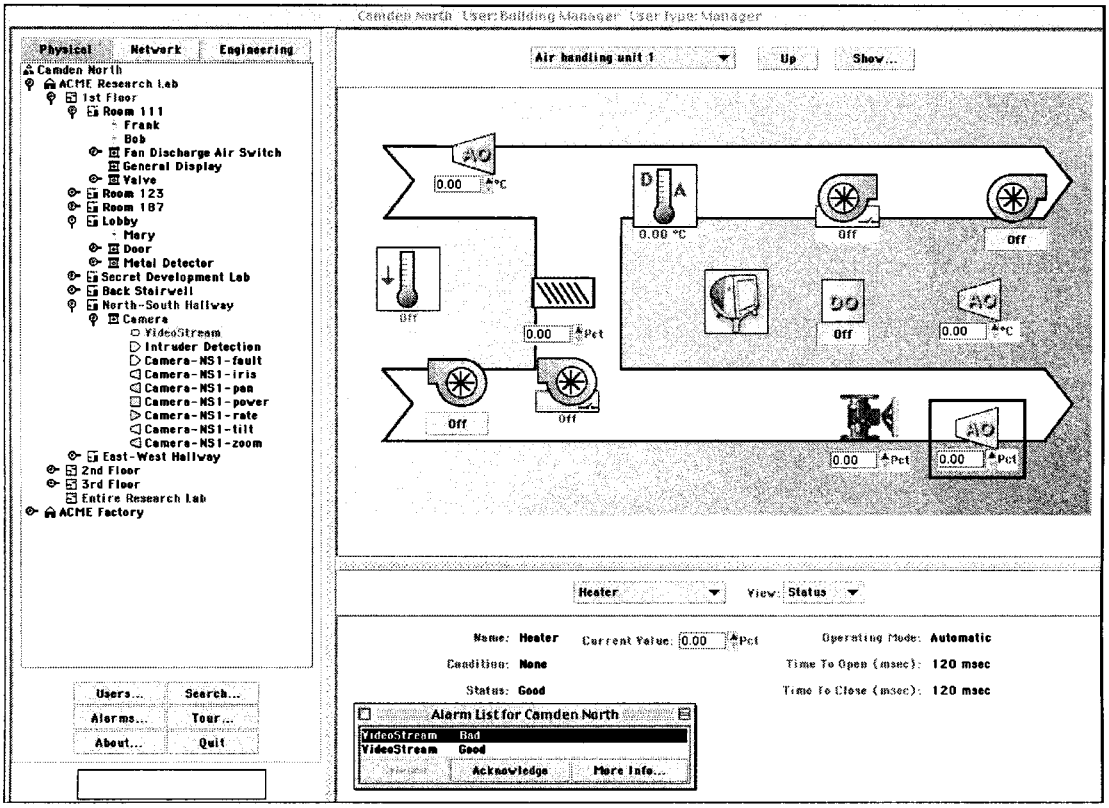

Fig. 1. DIGBE-generated user interface for a building manager.

data types and information roles have been constructed (e.g., [26]). The domain model required to support automatic user interface design in DIGBE, like other representations of buildings and their control systems, contains information about control objects, their location within the building structure, and their hierarchical organization within control systems. The types of artifacts in the DIGBE domain model are shown in Fig. 2. In this and other diagrams, we conform to the Unified Modeling Standard [27], and represent object oriented inheritance relationships as arrows (with the arrow head pointing to the parent, and the child referred to as the specialization). Composition (or "has-a") relationships are shown as lines with a circle at one end (with that end indicating the owner or container in the relationship, and the other end the subpart).

The artifacts and their organization in the DIGBE domain model mirror the organization that the users of building management systems hold about their systems; many databases currently in use have similar structures. Sites have three sub-hierarchies: their logical engineering structure (panel groups, containing panels that physically contain controllers), the network structure (C-buses, which are networks of controllers, each controlling multiple plants), and the physical structure (buildings, containing areas, then rooms with equipment and occupants). Plants and equipment may have associated points, which are the input and output components of control systems. There are a number of different types of points possible in a control system, including sensors, actuators, and counters, with specializations of each type of point to include the basic types of input/output objects present in buildings.

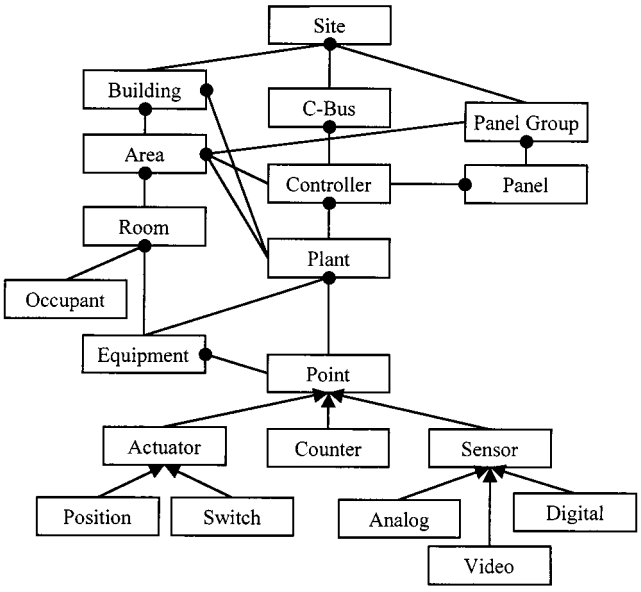

Fig. 2. Domain model classes in DIGBE.

When the DIGBE application is provided with a pointer to a database containing digital control configuration information for a building (or set of buildings, called a "site"), and it has knowledge of the schema for that database, it does two things. First, it converts the objects that are defined in the database into corresponding domain model objects, then instantiates them into an application model that it creates to represent that building system.

A very small building with only one simple controller would have an application model with the artifacts shown in Fig. 3. The highest-level object represents the site "My Store." It has one building, one panel, and one control network. The control network contains a heating and ventilation control (HVAC) plant, which has a selectable mode (heat/auto/off). There are two rooms that also contain equipment that is part of the HVAC plant; the "Back Room" contains equipment called "Furnace," with an on/off switch, and the "Sales Room" contains equipment called "Thermostat," which has a sensor for the actual temperature and a controller to set the desired temperature.

An application model consisting only of physical artifacts provides important semantic information about objects and their relationships. Other domain information, however, is necessary to support the automation of interface design. The problem lies in the different types of data that are present in different aspects of the control system, and the different types of interactions required with different types of equipment. While sensors cannot be controlled, only read, actuators need to be both readable and controllable. Different types of actuators have different control and display requirements; for example, a temperature control has a continuous range with high and low limits, while a heat mode switch is a three-state control. The automated interface design system needs to be able to select the appropriate interfaces for different kinds of data types; for example, on a standard PC user interface, state objects like on/off switches require very different interactions and coding than, for example, video objects.

To provide some of the required data semantics needed to support automatic interface design, we also include a category of objects in the domain model called Domain Data. Each artifact in the domain model has a number of properties, such as their current value, permissible settings, or operational limits. We have identified eight basic data types required to represent the properties of objects in use in digital control systems of the type we are considering: accumulator (simple counter), continuous (range), discrete (setting choices), file (data file), state (Boolean), text, time, and video. When an artifact is created, each of its data properties is also instantiated as one or another type of domain data object. For example, all artifacts, when they are instantiated, also instantiate a text data object, to represent their name. The data objects associated with an analog sensor (e.g., a temperature or humidity sensor) include status (a state data), access level (a discrete data), various continuous data values representing their current value and alarm limits, and file data objects representing pointers to history files.

During the development of DIGBE, we found that the design of appropriate interfaces is also enhanced by deeper knowledge of the uses to which that information may be put in context. To provide this information to the interaction design model, we provided additional semantic data markers in the DIGBE domain model, in the form of Java classes called Roles. The Role classes in DIGBE are singleton classes (only one instance of a class is ever present in a virtual machine). These classes provide the language spoken between the domain model and the interaction model, much as a set of enumerations would act in the C programming environment. Because the language consists of software objects instead of simple numbers or strings, however,

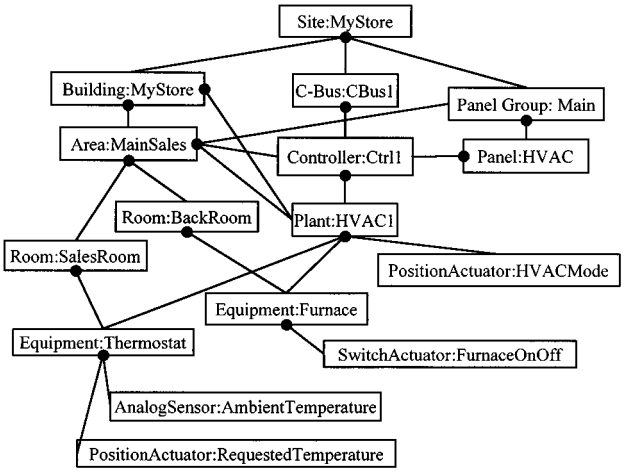

Fig. 3. Example application model for a simple building.

it becomes much more flexible: the language can represent an ontological hierarchy through inheritance, and actions (function calls) can be attached to these language objects to give them object-oriented semantic knowledge.

There are four main types of roles provided in the domain model: agent roles, process roles, referential roles, and data roles. *Agent roles* are associated with a user when that user is created or re-defined, and currently include engineer, guard, maintainer, manager, or operator. These roles are used by the Interaction Model to select the task set for the initial user interface design. Because of the agent role of operator assigned to a particular security guard, only operations tasks, and not maintenance, management, or technical tasks, will be presented to this user.

Process roles are currently limited to the two main processes that DIGBE has knowledge of: building environmental processes and security processes. These roles are assigned to both plants and users when they are created, and are used to constrain information for particular users. For example, a particular security guard has the process role security. When the interaction model composes the interaction design to support this user, only objects from plants that the user is concerned with will be selected, so that the security operator will not be shown or have access to environmental control system components.

Referential roles are used to provide information about the kind of thing referenced by each object in the domain model. They are statically preassigned to each object in the domain model, and their instantiations retain the referential role assigned to them. all artifacts are assigned an object referent role, to indicate that they refer to objects in the world. Data objects are assigned individual referential roles that are used by the interaction design objects to provide a basis to choose between different user interface components; for example, radio buttons, rather than number spinners, are better choices for data that refers to Boolean states. Using referential role interfaces provides the semantic information that allows DIGBE to make such choices. State data objects have a Boolean referential role, for example, to generically represent that any data object that is a specialization of a state data object refers to Boolean

data. Continuous data objects, in contrast, are associated with a number referential role, to represent that they operate on numerical, continuous data.

Data roles are associated with the data objects in the application model when a particular data object is instantiated. Data roles provide semantic information about the purpose of the information represented by the data object, in the context of the artifact it is associated with. For example, the objects representing the current value of a temperature sensor, a fan switch, and thermostat are the data objects continuous, state, and continuous, respectively. Each of these current values is assigned a data role that indicates the type of information they represent for the object they are associated with. The temperature sensor provides a measured value, and is given the data role of value. The fan switch, in contrast, is a setting that determines how the system will operate; the current value for the fan switch is assigned an operational parameter data role. The thermostat isn't a measured value, but a desired setting, and so is given the data role of setting. These data roles are used to constrain the subtasks to apply only to information that is needed in the current situation.

The DIGBE domain model objects are used as templates when DIGBE builds the application model that represents the processes, structures, and objects that are present in the system. Using an architecture like DIGBE, we have found that a domain model needs to support the following semantic information:

- control system objects and their hierarchical relationships;
- data objects, to represent the semantic information about data values associated with objects;
- role information, to guide the selection of appropriate user interface elements, to provide differential access to different types of building processes, and to allow interaction design based on the purpose and form of information.

B. Interaction Model

DIGBE's interaction model contains the information required to design interactions between a user and a control system, based on the requirements of the current situation as they are represented in the application model. We built a five-layer hierarchical model for DIGBE, with the top level representing applications that are required by the major roles of building management system users. Fig. 4 shows the organization of the five levels; using UML notation, this figure shows that the components of each level are objects from the next lower level. In DIGBE, applications are composed of tasks, which are composed of subtasks. Subtasks are composed of elements, which, in turn, are composed of primitives.

DIGBE has three applications: building management, building operation, and building technician. When a user logs on to a DIGBE control system, the role associated with that user determines which application is selected as the basis for the user interface. Based on our analysis of the requirements of building management user interfaces in earlier work, we had developed task analyzes for each of the users in a building management system. The basic tasks that each user requires for each application make up the second layer in the DIGBE interaction model, the task level. Fig. 5 presents the task hierarchy in the DIGBE interaction model. Objects that are

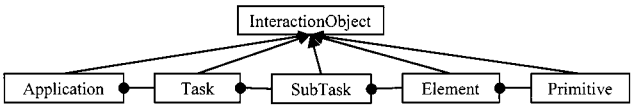

Fig. 4. Interaction model hierarchical levels.

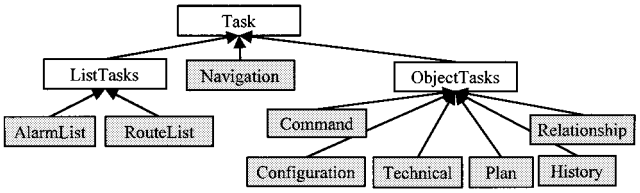

Fig. 5. Interaction model tasks.

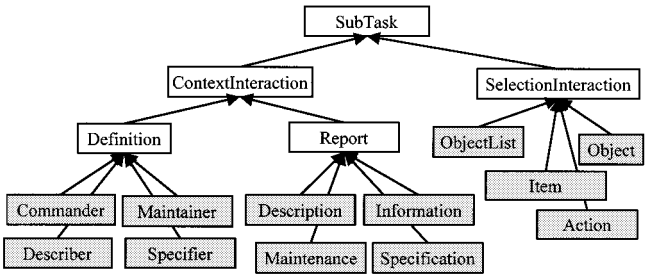

Fig. 6. Interaction model subtasks.

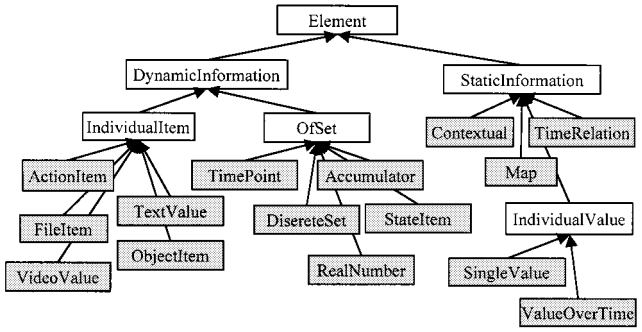

Fig. 7. Interaction model elements.

shown with shaded boxes are terminal objects, which can be instantiated into an interaction design, while objects in white boxes are abstract, and only their terminal specializations can actually be instantiated.

Tasks are composed of subtasks; the DIGBE subtask hierarchy is shown in Fig. 6. Subtasks are composed of elements, shown in Fig. 7, which are in turn composed of primitives, shown in Fig. 8.

The interaction model is a *self-composing productive system*. When an instance of any interaction object is created for a particular user and a particular set of domain objects, that instance is responsible for adapting to the current situation and producing its own parts. Objects fine-tune their sub-components for the specific context, based on appropriateness to the situation and the user. Each subcomponent then tunes itself by selecting and tuning its own subcomponents. Fig. 9 shows the compositional nature of the interaction model. To simplify the diagram, we only include the technical, command, and configuration tasks,

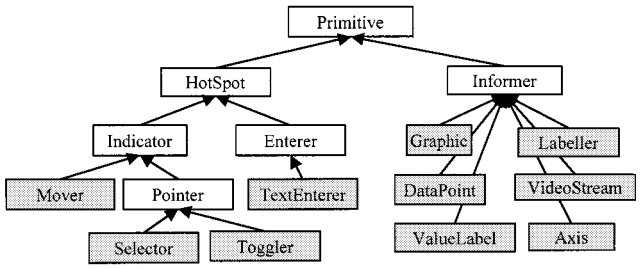

Fig. 8. Interaction model primitives.

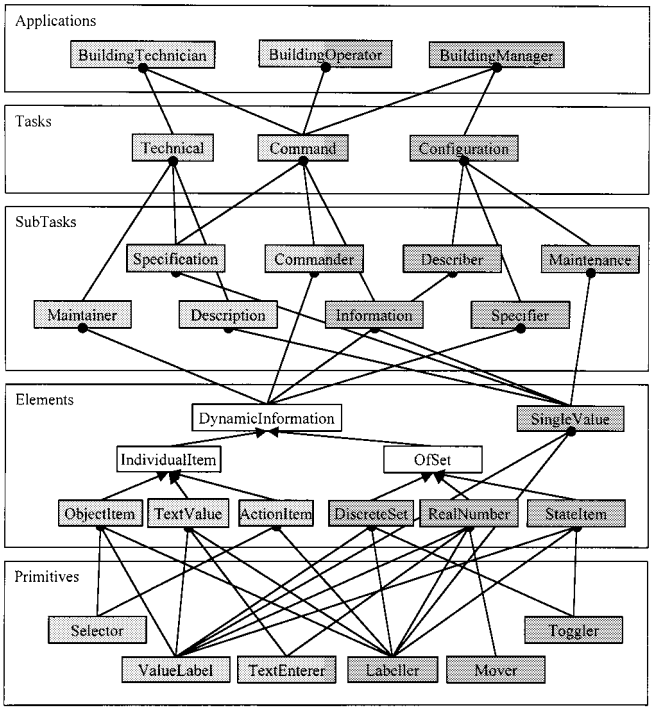

Fig. 9. Portion of the interaction model, showing compositional relationships.

and have removed any abstract objects that are not compositional subparts of the interactions included in the figure.

When a user requests a DIGBE application by logging on, the system selects the appropriate application, and instantiates it into the interaction design that it is building. This application object then determines the tasks it requires for that user, and selects the appropriate task objects to be instantiated into the interaction design. These task objects also compose themselves, by selecting the subtasks that are defined for their tasks. Subtasks, in turn, select and specialize their component Elements, which, finally, select and instantiate appropriate primitives. Using this process of self-composition, an entire user interface (or only the parts that need to be changed) is automatically produced or modified in real time, when needed. An example of a generated interaction design is shown in Fig. 10. The example shows only the configuration task for a building manager interested in configuring a freeze stat (a thermostat that monitors for freezing temperatures). To simplify the example, only the specifier subtask is expanded to show its component elements, and only the StateItem element that represents the operating mode for the freeze stat is expanded to show component primitives.

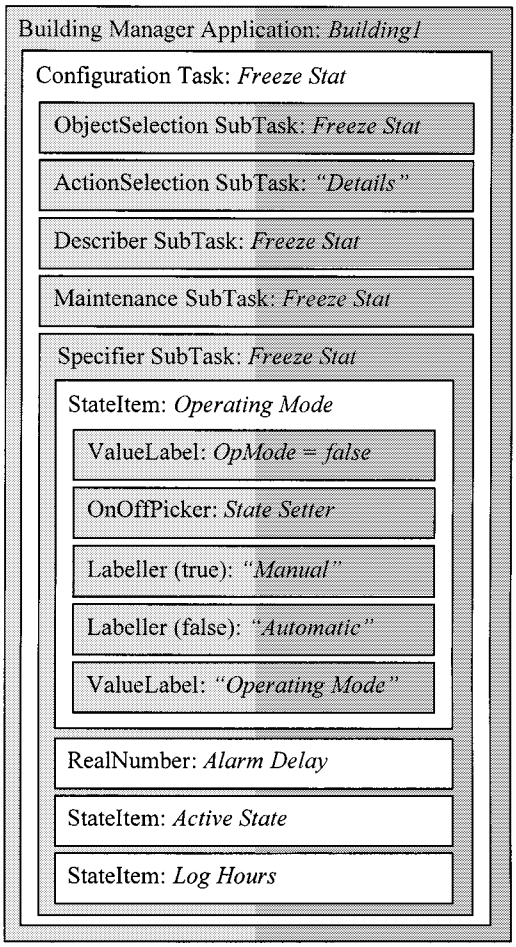

Fig. 10. Portion of interaction design for configuration task.

Of particular interest is the mechanism by which subtasks in the DIGBE interaction model select and specialize their elements. First, the subtask uses the role of the data in the system to select only the objects that have the same role as that subtask. The specifier subtask, for example, searches for data objects that are associated with the freeze stat that have the operational parameter role, and creates an editable element for only those data objects. The specification subtask, on the other hand, presents the same information as noneditable elements.

As shown in Fig. 9, the elements that are subcomponents of the maintainer subtasks are only defined generically, as dynamic information elements. because this object is an abstract object in the interaction model, the dynamic information element specializes itself to fit the data object of interest, using the referential role associated with that data object. For example, the data representing the operating mode of the freeze stat is associated with referential role of Boolean, as is the element StateItem, so a StateItem is selected to represent that particular data object.

C. Presentation Model

The presentation model in DIGBE is separate from the interaction model and the interaction design in order to make DIGBE applicable to multiple types of user interface devices. To prove the DIGBE concept, the prototype is capable of composing interfaces for users on handheld Palm devices or CRTs, with sep-

arate presentation models for each of these two devices. The DIGBE presentation model converts the interaction design into an actual presentation, by selecting user interface components like windows (for applications), frames (for tasks), and widgets (for elements). It also contains a number of heuristics that allow it to line up widgets, add appropriate coding (selecting, for example, icons to represent objects in mimic displays, and to appropriately color code alarm status of objects in navigation hierarchies). Fig. 11 presents a portion of a user interface presentation for the specifier subtask, showing only the ObjectSelection (labeled "Freeze Stat"), ActionSelection ("View: Details") and Specifier SubTasks.

Separation of the presentation model from the interaction model provides device independence, allowing the same interaction to be presented in the most appropriate fashion for a particular device. For example, although the interaction design element that represents alarm status is turned into a color code when the user interface device is a CRT, the handheld device we used for the prototype implementation of DIGBE did not display color. The presentation model for this type of device, instead, converts the element that represents alarm status into "bold coding", in which the symbol or name of an object in alarm is shown with bolder lines than normal.

In the DIGBE proof of concept prototype that we developed, there is one application model for each site, and one active Interaction Design for each user logged onto the control system. A user may have multiple user interfaces at one time; when they access a DIGBE application from a different device, a user interface is generated for them, regardless of whether another user interface is currently being generated; both use the same interaction design, differing only in the Presentation Model that is used to tune the interaction to each device.

IV. Conclusions and Future Research

We have discussed the models that DIGBE uses to automatically design interactions that dynamically adapt to the user and to the task and data environments. DIGBE demonstrates that interaction design knowledge can be modeled as an automated real time constraint- and affordance-driven process, and that this process can be separated from those necessary to present interactions on user interface devices and from those necessary to internally represent information about domain objects.

Additional areas of research are suggested by the current work. An extension of the DIGBE architecture to allow the incorporation of independently developed models of tasks, users, interaction design processes, and domains would greatly increase the utility of the system. In addition, integration of a modularized, compositional approach with efforts whose primary emphasis are the development of robust models would demonstrate both the real-world application of the models and the scalability of an approach like the one taken with DIGBE. One promising example is the emerging work on task characterization [28]. This research has been successfully applied to automated visual discourse synthesis, using a model of presentation intents that is matched to a model of the visual tasks that achieve them. DIGBE attempts to include this sort of information through its dynamic specialization mechanisms,

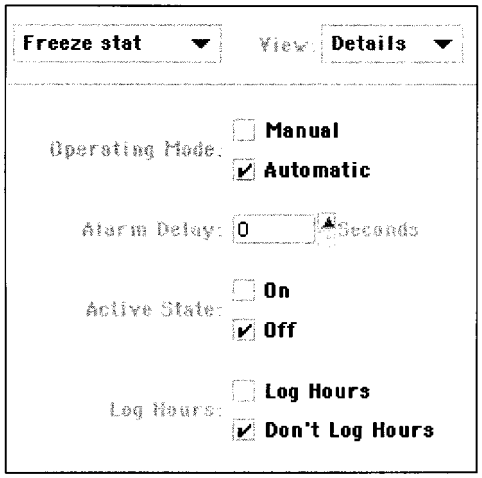

Fig. 11. Presentation of specifier subtask for freeze stat.

but the information is embedded in the interaction objects and not fully constrained by the roles in the domain that correspond to the intent structure in [28].

Additional future research needs include improvements to the modeling of actions available to the user (both in the domain and the interface). In addition, a system like DIGBE would benefit from the development of domain semantic reasoning mechanisms to guide visual layouts, the ability to dynamically add objects to the domain, and mechanisms to accommodate objects which cannot be fully classified (and therefore given roles) in the current domain structure. Further research is also required to provide the information infrastructure that is needed to make such an approach viable, including automatic discovery of domain information and extension of the modeled domains to determine scalability. In addition, to provide ease of evolution, such a system would require the addition of several complex programmer interfaces. These interfaces, which DIGBE could itself generate, would allow user interface design specialists to refine the design knowledge possessed in the different types of models.

In complex control domains, automated interface design could become an economically beneficial alternative to pre-designed user interfaces requiring expensive configuration. DIGBE demonstrates that, although full automatic synthesis of any interface at any time may not yet be feasible, it is both possible and useful to automatically generate interfaces to complex systems domains with known task structures.

References

[1] T. Landauer, *The Trouble with Computers*. Cambridge, MA: MIT Press, 1995.
[2] J. Nielsen, *Usability Engineering*. Boston, MA: Academic, 1993.
[3] D. Mayhew, *The Usability Engineering Lifecycle*. San Francisco, CA: Morgan Kaufman, 1999.
[4] W. Galitz, *The Essential Guide to User Interface Design*. New York: Wiley, 1997.
[5] T. Mandel, *The Elements of User Interface Design*. New York: Wiley, 1997.
[6] B. Myers, "User interface software tools," *ACM Trans. Comput.–Hum. Interact.*, vol. 2, no. 1, pp. 64–103, March 1995.
[7] R. Penner, "Developing the process control interface," in *Engineering for Human Computer Interaction*. New York: Elsevier, 1993, pp. 317–334.

[8] R. Penner and N. Soken, "Consistent honeywell interface: Tools for developers," *Sci. Honeyweller*, pp. 106–109, 1993.
[9] R. Penner, "Multimedia interfaces for process control," in *Proc. Energy-Sources Technology Conf., ASME Petroleum Div.*, 1994, pp. 375–383.
[10] ——, *Consistent Honeywell Interface Design Concept for Building Management*. Minneapolis, MN: Honeywell, Inc., Sensor Syst. Develop. Ctr., 1992.
[11] R. Penner and E. Steinmetz, "DIGBE: Adaptive user interface automation," in *AAAI Spring Symp.*, Stanford, CA, Mar. 2000, pp. 98–101.
[12] P. Szekely, "Retrospective and challenges for model-based interface development," in *Computer-Aided Design of User Interfaces*, J. VanderDonckt, Ed. Namur, Belgium: Presses Univ. de Namur, 1996.
[13] M. Byrne, S. Wood, P. Sukaviriya, J. Foley, and D. Kieras, "Automating user interface evaluation," in *Proc. 1994 Conf. Human Factors in Computing Systems (CHI94)*, 1994, pp. 232–237.
[14] S. Roth and J. Mattis, "Data characterization for intelligent graphics presentation," in *Proc. Conf. Human Factors in Computing Systems (CHI '90)*, 1990, pp. 193–200.
[15] S. Casner, "A task-analytic approach to the automated design of graphic presentations," *Trans. Graph.*, vol. 10, no. 2, pp. 111–151, 1991.
[16] D. Seligman and S. Feiner, "Automated generation of intent-based 3D illustrations," *Comput. Graph.*, vol. 25, no. 4, pp. 123–132, 1991.
[17] M. Zhou and S. Feiner, "Data characterization for automatically visualizing hetrogeneous information," in *Proc. IEEE Information Visualization'96*, 1996, pp. 13–20.
[18] ——, "Top-down hierarchical planning of coherent visual discourse," in *Proc. 1997 Int. Conf. Intelligent User Interfaces*, 1997, pp. 129–136.
[19] J. Foley, C. Gibbs, W. Kim, and S. Kovacevic, "Knowledge-based user interface management system," in *Proc. 1988 Conf. Human Factors in Computer Systems (CHI '88)*, 1988, pp. 67–72.
[20] C. Wiecha, W. Bennett, S. Boies, J. Gould, and S. Greene, "ITS: A tool for rapidly developing interactive applications," *Trans. Inform. Syst.*, vol. 8, no. 3, pp. 204–236, 1989.
[21] P. Szekely, P. Luo, and R. Neches, "Beyond interface builders: Model-based interface tools," in *Proc. INTERCHI '93*, 1993, pp. 383–390.
[22] A. Puerta, H. Eriksson, J. Gennari, and M. Musen, "Model-based automated generation of user interfaces," in *Proc. Nat. Conf. Artificial Intelligence*, 1994, pp. 471–477.
[23] J. Vanderdonckt, "Automatic generation of a user interface for highly interactive business-oriented applications," in *Intelligent User Interfaces*, M. Maybury and W. Wahlster, Eds. San Francisco, CA: Morgan Kaufmann, 1998, pp. 516–524.
[24] J. Vanderdonckt and A. Puerta, Eds., *Computer-Aided Design of User Interfaces II*. Dordecht, The Netherlands: Kluwer, 1999.
[25] R. Monafred, A. Hodgson, B. Bowen, and A. West, "Implementing a model-based generic user interface for computer integrated manufacturing systems," in *Proc. Inst. Mech. Eng.*, vol. 1212, 1998, pp. 501–516.
[26] P. Torcellini, "Supervisory control for intelligent building systems," Ph.D. dissertation, Purdue Univ., West Lafayette, IN, 1992.
[27] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. New York: Wiley, 1999.
[28] M. Zhou and S. Feiner, "Visual task characterization for automated visual discourse synthesis," in *Proc. CHI98: Human Factors in Computing Systems*, 1998, pp. 392–399.

Robin R. Penner received the Ph.D. degree in cognitive science and robotics from the University of South Florida, Tampa, in 1989.

She has over 20 years of industry experience in artificial intelligence, usability engineering, and software engineering, primarily applied to the design and implementation of experimental adaptive systems. She is currently the president and chief cognitive scientist for Iterativity, Inc., Minneapolis, Minnesota. Her main research interests include decision-aiding automation, cognitive modeling, and user interface design.

Erik S. Steinmetz received the M.S. degree in computer science from the University of Minnesota, Minneapolis, in 1999.

He has formal training in artificial intelligence, computer science, philosophy, and mathematics. He is currently the Vice-President and Lead Software Architect for Iterativity, Inc., Minneapolis. His major research interests include agent communications, adaptable user interfaces, and human decision aiding.