Implementation of Automated Interaction Design With Collaborative Models

To appear: "Interacting With Computers"
Original submission May 5 2002

Robin R. Penner and Erik S. Steinmetz
Iterativity, Inc., 118 E. 26th St., Minneapolis, MN, 55404
robin@iterativity.com, erik@iterativity.com

Abstract
This paper summarizes the current status of an ongoing research program to explore automated alternatives to the current manual method of designing, implementing, and delivering user interfaces to complex digital control systems. Using examples from two implementations of the model-based interface automation approach that resulted from this research, we explore the models and collaboration required to perform on-demand user interface design. We first discuss the need for automation of the user interface design process and place the work into a research context. Using examples from two implemented systems, we then review the object-oriented models and processes that we used to support interaction design automation. Our findings support the application of model-based automated design approaches in digital control system domains, and particularly emphasize the need for rich semantic support for automated design.

Key Words: User Interface Design, Interaction Design, Model-Based Automation, Interaction Design Automation

INTRODUCTION

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, Landauer, 1995 and Nielsen, 1993). Processes and artifacts for performing user interface design have been successfully developed and are in widespread use (e.g., Galitz, 1999; Mandel, 1997; Mayhew, 1999; Rosson & Carroll, 2002), 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 digital systems have been difficult to achieve for many reasons, including the need for specialists to perform the design work, rapidly changing technology, increasing interaction options, difficulty quantifying procedures and results, and the difficulty inherent in producing and adopting useful specifications, standards, or guidelines. In addition, when standards and processes do exist, there is often little management support for their application by software engineers involved in real-world development efforts. In most

cases, user interface design is not considered an integral part of the development process until late in the cycle. Physical, cultural, and managerial separation between user interface designers and application implementers can compound the problem.

These general problems with the delivery of usable systems are exacerbated in the various domains based on digital process control. In applications like factory control, building systems management, flight systems control, and operations management, user interfaces are usually individually developed for each application, installation, process, and, often, each user. This is an expensive approach, requiring skilled human labor. In addition, such systems pose a number of problems that may result in less than optimal user interface designs. First, each digital control system is unique. Each system, even those with the same set of functions, has a different set of components, organizations, tasks, users, computing systems, control programs, and interaction devices. Second, these individual features change over time, sometimes very quickly, and all evolve with the advances in digital control technology. This frequently necessitates changes in at least the user interface portions of the control software, and often requires completely new user interfaces to be developed, installed, and mastered. Third, the users and tasks in this environment are diverse, and multiple separate software applications are required in order to support them. Even with proper usability engineering applied to each application, user interfaces to various systems and functions will differ, and each site that needs a particular application will require manual user interface specialization.

In response to these problems, we have been involved in a long-term research and development program to reduce the cost and improve the usability of digital control interfaces (Penner, 1992, 1993, 1994, 1996, 1999; Penner & Soken, 1993, Penner & Steinmetz, 2000, 2002). Initially, we limited our research domain to the relatively simple one of building management; we have continued the program in the more complicated domain of military operations planning and control. The result of this research has been the design and implementation of several online model-based systems that create, display, and modify all user interfaces that are required by each user of a particular type of digital control system. All aspects of the user interface are dynamically created on demand, specialized for the situation, the specific users, and the interface devices they choose to use.

The approach that we developed, called MAID (Model-based Automation of Interaction Design) has allowed us to explore the allocation of the various functions that are required to drive automated interface generation, as well as to experiment with mechanisms for arranging each model and accomplishing each required function. Systems built using the MAID approach allocate knowledge about domain semantics, interaction design, and interface presentation into separate, collaborating models. Each model contains knowledge about a particular aspect of interface design, with the details distributed among hierarchically organized objects. The system uses this knowledge to dynamically create and manage all the user interfaces that are needed by any user.

We will discuss examples from two implemented systems to illustrate the architecture, design decisions, and evolution of this approach. The first system, called DIGBE (Dynamic Interface Generation for Building Environments; Penner & Steinmetz, 2000, 2002), is a fully functional system that provides on-demand user interfaces to building management control systems with multiple users and interaction devices. The second, AID (Automated Interaction Designer), is under development; currently, it provides automatically designed and presented user interfaces

for situation monitoring of military planning environments, and is being extended to support the full range of military analysis and operations interactions.

RESEARCH CONTEXT

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 (for example, see Szekely, 1996) 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 (Byrne et al., 1994). Computer Aided Design of User Interfaces (CADUI) (Vanderdonckt, 1994; Vanderdonckt & Puerta, 1999) 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.

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. The model-based approach we took to automated interaction design was influenced by the findings of numerous other researchers in this area. For example, SAGE (Roth & Mattis, 1990) generated 2D static presentations of relational data, and emphasized the importance of multi-dimensional representations of data and the effect of user goals on effective design. BOZ (Casner, 1991) 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 (Seligman & Feiner, 1991) had a separate presentation component, and included representation knowledge of users, tasks, and contexts. Later work on IBIS included an emphasis on data characterization (Zhou & Feiner, 1996) and hierarchical decomposition of visual lexicons (Zhou & Feiner, 1997). Investigations in the area of User Interface Design Environments (UIDE) also informed the development of the MAID approach, including modeling of user interface design knowledge (Foley, Gibbs, Kim, and Kovacevic, 1988), separation of design environments into multiple layers for the domain, the interactions, and the presentations (Wiecha et al., 1989), the use of declarative user interface design models (Szekely, Luo, and Neches, 1993), and the use of domain models to provide the underlying semantics for user interface design (Puerta, Eriksson, Gennari, and Musen, 1994).

AUTOMATED INTERACTION DESIGN

The DIGBE program had several explicit goals, including:

Improve the quality of human computer interfaces to digital control systems,
Reduce the need for human-supplied user interface software development efforts,
Integrate diverse applications and services,
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, and

Support evolution in control systems, user functions, user interface design knowledge, user interface devices, and human-computer interaction paradigms.

The current AID program has several *additional* goals, including:

Reduce the number of operators required to control the automation,

Incorporate emerging principles of good design to improve user interfaces to military operations domains,

Expand domain modeling to complex mobile situations,

Incorporate uncertainty, planning, collaborative behavior, and external automated reasoners,

Refine modeling of interaction design knowledge, and

Expand role structure to provide additional semantic constraints.

Figure 1 shows a screen shot of a DIGBE-generated user interface for the manager of the environmental control system in a large building. The DIGBE user interface allows managers, operators, and technicians to fully control, monitor, and configure a building's digital control systems, without any prior user interface designs. The basic tasks for users are presented as panes of an application window, and, for the example in Figure 1, include system navigation (left pane), hierarchically navigable system monitoring (top right) and detail data interactions with the selected object (lower right).

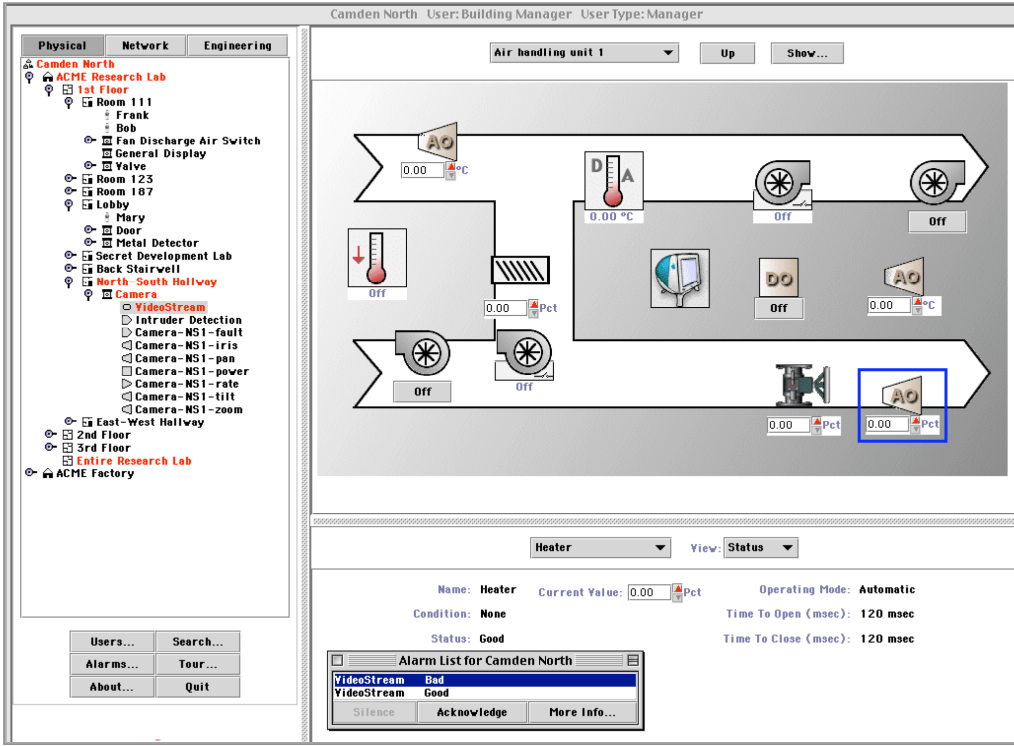

Figure 1. Example DIGBE-Generated User Interface

The AID system incorporates many of the same design principles applied in DIGBE. An example from a preliminary AID user interface is shown in Figure 2. The AID system is currently usable by situation monitoring personnel in military situations. Using a simulation as the source of situation information, it designs and presents visualizations of the unfolding situation, and provides interfaces to allow users to configure and control simulation objects. Although the AID system has very sparse domain, interaction, and presentation models, it is capable, even in the early stages, of automatically generating rule- and constraint-based interactions with appropriate choices of tasks, formatting, coding, information allocation, widgets, etc.

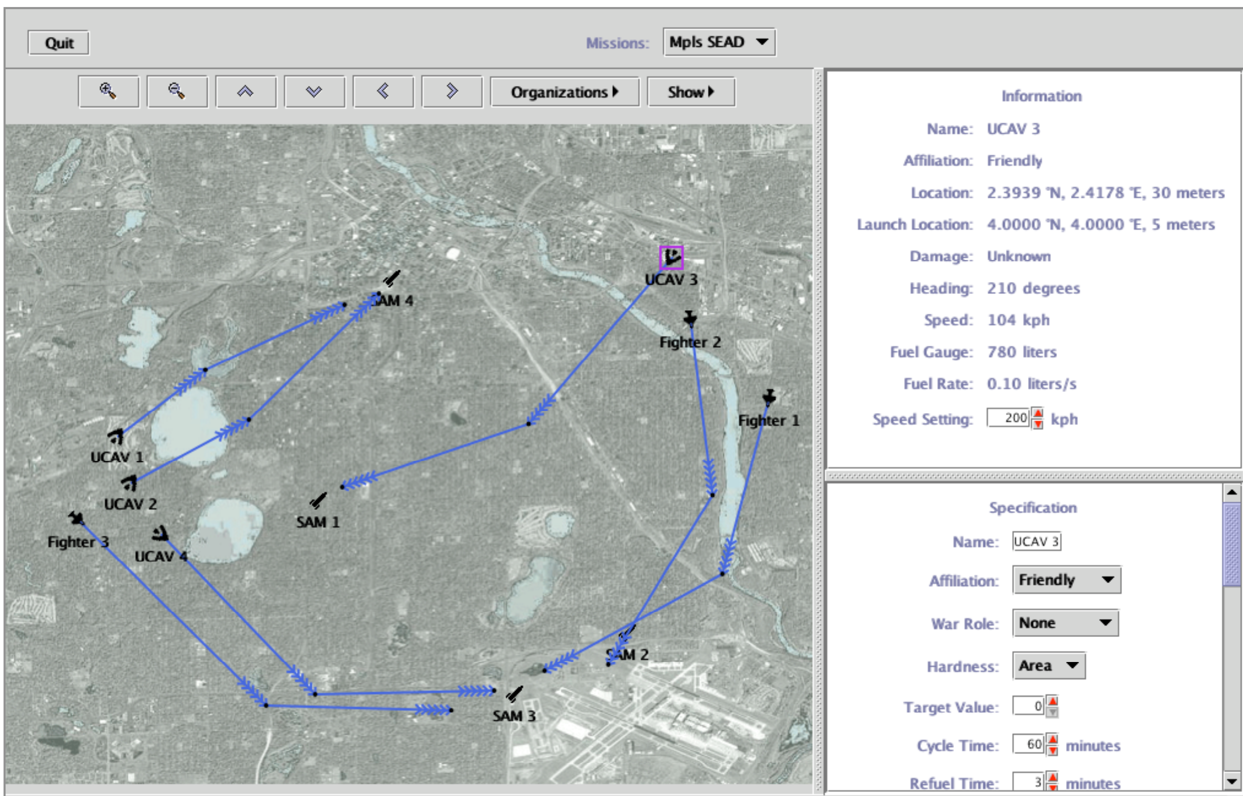

Figure 2. Example AID-Generated User Interface

These systems are able to provide specialized user interfaces for each situation and user because they use internal models of good design (as we currently understand them) to create and tune each interface. Responsibility for the various aspects of an interface design is distributed into different models that collaborate to produce the user interfaces. Figure 3 illustrates how, using the MAID approach, the functions of situation representation, interaction design, and interface design are partitioned into three different object-oriented components, each with particular responsibilities in the collaborative design effort.

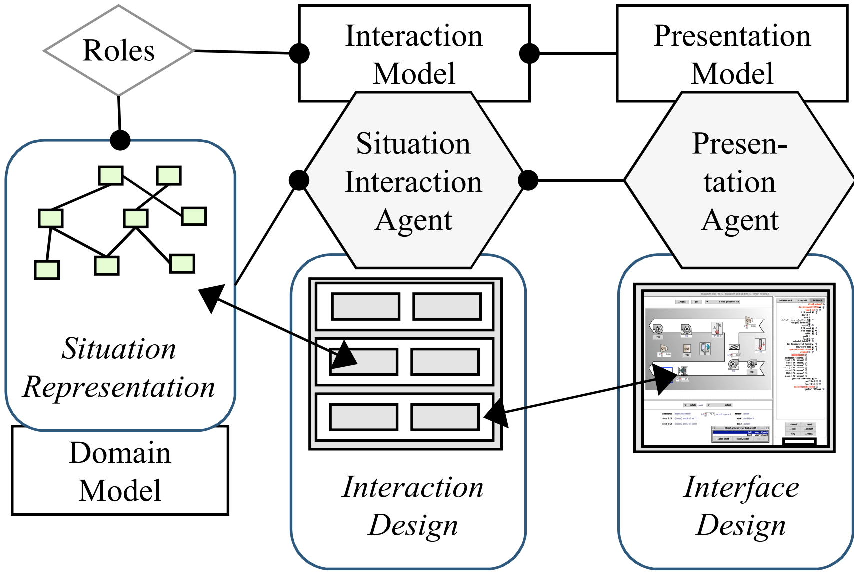

Figure 3. Model-Based Automated Interaction Design Architecture

Object-oriented models provide the knowledge base for each component. The **Domain Model** (at the lower left of the diagram) provides the semantic basis for interaction design, allowing control system independence and an object-oriented representation of control system components. The hierarchical, task-driven **Interaction Model** (top center) provides the design knowledge required for automatic composition of appropriate tasks and abstract interactions, allowing dynamic application of the principles of usability engineering to support abstract interactions between people and systems. The **Presentation Model** (top right) possesses knowledge about how to select and format concrete user interfaces to express interaction designs, allowing hardware and operating system independence, and a clear separation between abstract interactions and concrete interfaces.

Under the control of software agents, each model is used as an exemplar database to produce dynamic models of the ongoing situation, the current interaction design, and the current presentation. An object-oriented **Situation Representation,** containing a representation of the situation-specific system in the real world, is created from the domain model. All interaction participants share dynamic situation representation, ensuring shared knowledge for grounded collaboration. The **Interaction Agent** starts the design process for a particular user by creating an interaction object, which specializes itself using a compositional productive process to create an **Interaction Design**. The **Presentation Agent** (under the direction of the Interaction Agent) selects the appropriate Presentation Model for the currently accessed device (CRT/keyboard or handheld, in the current systems). It uses the objects in the presentation model as templates, selects and specializes them as necessary, and presents the Interaction Design as an **Interface Design**. As the user interacts with the situation through the generated user interface, or as the situation changes, the entire automated interaction design system continues to dynamically support the necessary user-system conversation.

Situation Modeling

In a MAID system, the semantics of the situation that is the topic of the human-system dialog are contained in the situation representation. The situation representation is dynamically created from the objects defined in the domain model, which has three basic types of classes: things, data, and actions.

The “things” in the situation are created from a specialization of the domain class Artefact. They represent the inter-related individual system components that are present. In DIGBE, only concrete objects and actors were represented in the domain model. Work on the AID program has extended the domain model to include less tangible things like organizations, information, and situations. An object can be added to the situation model in a number of ways: because it is part of the system description (either a static description or the online description a control object broadcasts), because it is added by a user, or automatically as part of the creation of another object. For example, a flight control system would automatically be created for an aircraft, and an analog output would be created for a temperature sensor that is created for an air-handling unit.

A domain model consisting only of inter-related and hierarchically organized “things” provides important semantic information about objects and their relationships. However, it doesn’t capture the semantics of their data. To do this, we explicitly modeled data classes to represent this type of information. Each thing in the domain model is automatically provided with a set of data objects, representing its properties (such as the name, current value, permissible settings, or operational limits). When a thing is created, its data properties are automatically instantiated as one or another type of data object in the situation. We have identified basic data types required to represent the properties and data of objects in use in digital control systems, including: accumulator (simple counter), continuous (range), discrete (setting choices), file (data file), state (boolean), text, time, video, and location. For example, the data objects associated with an analog sensor (e.g., a temperature or humidity sensor) include status (a state), access level (a discrete), various continuous data values representing the sensor’s current value and alarm limits, and file data objects representing pointers to history files.

Figure 4 shows a portion of the domain model artefact hierarchy as well as a small portion of the domain data hierarchy. There are many more objects in the domain models of AID and DIGBE than are shown in the figure, and each object has many more relationships than are shown. Unified Modeling Language (UML) notation (Jacobson et al., 1999) is used in this diagram, with outlined arrowheads indicating a superclass and standard arrowheads indicating a relationship. As Figure 4 shows, an unmanned aircraft (UAV) is a type of Aviation, which is a type of Combat Equipment, which is a type of Equipment, which is a Physical Object, which is a Tangible Object, which is an Artefact (or “thing”). UAVs that are instantiated into the situation model inherit all the properties and relationships of their superclasses, unless they override them. So, all UAVs have their own sub-objects and properties (like their speed setting), but also inherit those of their parents (like the speed and heading sub-objects of Physical Object).

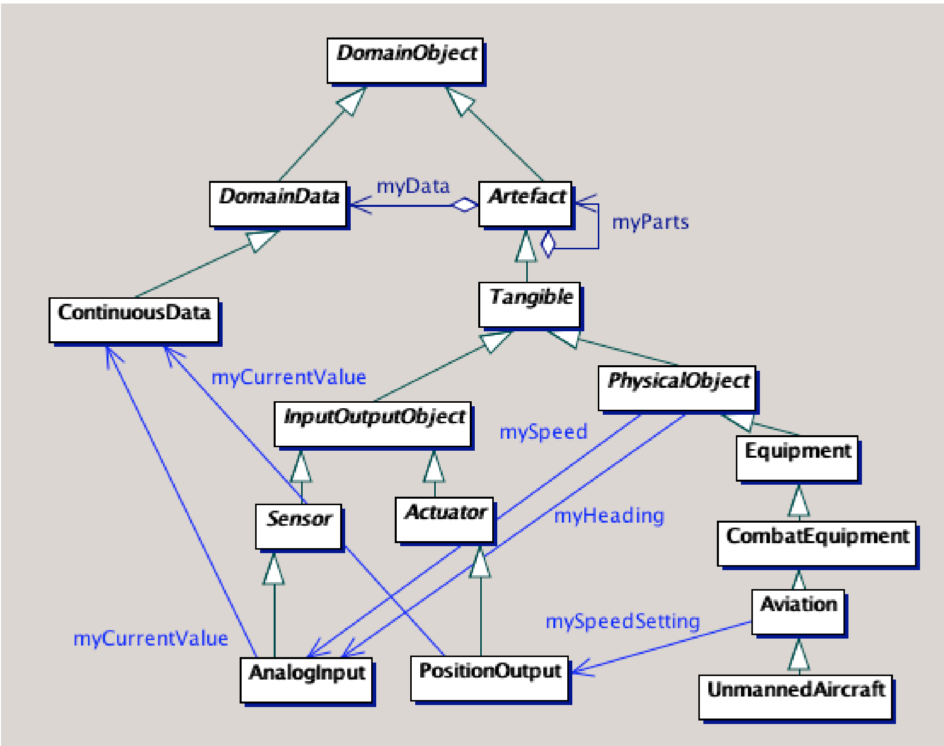

Figure 4. Portion of AID Domain Model

In addition to containing things and their associated data, the domain model includes both an implicit and an explicit representation of actions. Changing data values is an implied action that is allowed by data objects, and control actions to actuators are not represented as explicit actions in the domain, as command of control equipment is handled through objects representing actuators and actuator behavior. Other actions, however, are possible in the situation that act directly on processes or information. Modeling these actions as situation objects allows information about domain-specific actions to be conveyed without having to manually build new interface components. Examples of actions currently modeled include those required by user interface and task processes (e.g., Quit) that cannot be modeled through actuators.

Interaction Modeling

The automated design capabilities of DIGBE and AID stem from the knowledge that their interaction models possess about interaction design, and the ability to use the semantic information in the situation representation to drive the compositional design process. The interaction model is organized as a (hierarchical) cellular automaton driven by local behavior rules and affordance constraints (Penner & Steinmetz, 2002). Its purpose is to model the process of designing the interactions between the user and the information contained in the current situation.

The interaction model in DIGBE is made up of five levels, each composed of objects from the next lower level:

Applications (e.g., Building Management), composed of tasks;

Tasks (e.g., Monitoring), composed of subtasks;

Sub-Tasks (e.g., Selection), composed of elements;

Elements (e.g., Value Over Time), composed of primitives; and

Primitives (e.g., Labeler), the basic units of an interaction.

In DIGBE, the generation of user interfaces is user-based. When a user of a certain type asks for an interaction with a specific control system (represented by a situation model), the Interaction Agent creates an application that selects its component tasks based on the type of user that has logged in. In contrast, the AID system has a requirement of reduced users and a more fluid transfer of control between human and automated entities, and the task component has high semantic variability. Because of this, the task layer has been moved from the interaction model to the domain model in AID. Tasks are represented as hierarchical, situated processes with associated information (e.g., assignments, orders), data (e.g., completion status, priority), actions, and entities (actors and objects). In the interaction model, various views onto the situation are defined, which provide the basis for supporting human performance of domain tasks.

Figure 5 illustrates the inter-relationship of the classes in the AID interaction model.

Figure 5. AID Interaction Model

In addition to the deletion of the task layer, the most significant differences between the AID model and the DIGBE model are the general flattening of each level and the incorporation of the interaction agent directly into the interaction model. In AID, the Interaction Design Agent is the highest-level concrete interaction object. It is made up of Interaction Views, which are either directly requested by the user, or selected based on the status of tasks assigned to humans in the domain, and matched to the capabilities of the currently active human user. These views are composed of Interaction Elements, which are in turn composed of Interaction Primitives.

The interaction models in both DIGBE and AID are *self-composing productive systems*. When an instance of any interaction object is created for a particular user and a particular domain object (the *referent* of the interaction object), that instance is responsible for adapting to the communication needs of the referent and the user. It uses internal rules and constraints based on current practice in task analysis, requirements management, and interaction design, as appropriate to the responsibilities of that object. Interaction objects also select 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. Using this process of self-composition, an entire interaction (or only the parts that need to be changed) is created or modified in real time when needed. Each interaction design is specialized dynamically to suit the user, the objects in the real world, the interaction devices, and the required tasks. In use, each dynamically responds to user input (or changes in the system) by redesigning specific parts of itself, as needed.

To illustrate this, consider the situation in which a user needs to monitor the actual speed and possibly change the speed setting for a UAV. This is accomplished through an operational user logging on, navigating to the UAV (using a dynamically created map of the system), and selecting the UAV on a graphic map containing representations of the available UAVs (also dynamically created). In this case, the need is recognized and accomplished by the user. It can also be generated in the domain model as an unfinished task, and assigned to the user. This could happen, for example, if the UAV required supervisory control and the situation determined that a speed change was required. The result in both cases is the same: an information view containing the data for the UAV and its basic controls is created in the interaction design, and that information view begins to compose itself to best suit the UAV object and the interaction needs of the user.

Each of the view objects in the interaction model contains only certain sub-objects, to provide a particular view of the referent. This selection of which data needs to be included in an interaction is accomplished through the use of static objects called ***Usage Roles***. Usage roles provide the semantic information that allows the interaction design to select only the data that contributes to that view of the object. When a view is provided with a referent, its job is to select all of the data associated with that referent that matches the usage roles of the view. The information view, for example, is defined as containing data that provides identification, location, values, or setters for that referent.

Usage roles provide semantic information about the purpose of the information represented by each data object, in the context of the thing it is associated with, and independent of the type of data it represents. When a UAV, for example, is created, it creates a number of subparts, including an analog input data object to represent the sensed speed of the UAV (as illustrated in Figure 4). It also creates a position output to represent the actuator that commands the speed. Both the sensor object and the actuator object have an associated data object of type Continuous to represent their current values. In the case of the sensor, because the UAV uses this data as a sensed value, it sets the usage role of that data object to the “value” usage role. In contrast, the actuator sets the usage role of its associated continuous data object to the “setter” usage role. We have defined a number of static usage roles, including value, setter, identity, description, parameter, comparison, evaluation, location, and plan.

However, usage roles don’t specify which interaction elements to choose to represent each selected data object. This information isn’t provided statically, either, in many of the views. For

example, the Information View (see Figure 5) is defined generically, as being composed of a number of Interaction Elements (an abstract class; see the Information and Specification Views in Figure 5). Selection of the appropriate specialized class to represent each piece of data that needs to be included in the interaction is accomplished through the use of ***Representation Roles***. Representation roles are static roles that provide information about the *kind* of thing referenced by each data object in the domain model. They are pre-assigned to domain data classes and to interaction elements, and instantiations retain the representation role assigned to them.

Representation roles are used at the element level to choose between different interaction components to represent a specific item of data. State Data objects have a boolean representation role, for example, to indicate that any data object that is a specialization of a state data object represents boolean data. Continuous Data objects, in contrast, have a Number representation role, to indicate that they represent numerical, continuous data. When a view requires an element to represent an interaction with one of these objects, it first looks at its own defined treatment of objects with each type of usage role. In each view, some of the usage roles are also defined as static within that view, indicating that objects with those roles appear in that view but can't be changed by the user.

For objects of a statically defined usage role, the view selects a static element to represent the data. For objects whose usage role is not defined as static for that view, the view uses the representation role of the domain data that is the referent of the element that needs to be created, and matches it to an interaction element with the same role. So, for the speed setting current value, which is represented by a Continuous data object with a "setter" usage role (not defined as static in this view) and a "number" representation role, a Number Element is selected, since that element also has a "number" representation role. Currently defined representation roles include action, boolean, discrete, file, number, object, string, state, static, and location.

An important principle that was applied during the development of the MAID systems is the object-oriented principle of decoupling, where each part of a system needs only limited access to the other parts. Implementation of the DIGBE system in a real world domain revealed a number of domain model requirements, particularly in the representation of data and in the semantic information about users, processes, references, and data functions, and the AID development process has revealed others required to support more complex, mobile situations. While there may be multiple ways of embodying this information and applying it to help guide interaction design, the use of roles in Java provides a useful mechanism for exploring domain independence.

Roles in MAID provide the interface between the interaction model and the domain model, and form a lower limit on the amount of information necessary to adequately construct an interaction. Roles create the ontology necessary for the user to interact with the world as described by the domain model. As long as the domain model abides by this ontology, it may be constructed as specifically or generally as demanded by the situation, and the data sources controlling it. We are currently experimenting with modeling parts of the interaction itself in the domain model. The existence of the user, the interaction device employed by that user, and the existence of the interaction process (AID itself) has been made as an explicit part of the domain model to allow better modeling of the rules which determine a user's tasks and their engagement with those tasks.

Figure 6 shows an example of the design of an Information View for a UAV, along with its component elements and the primitives for one of the elements. Each data item associated with

the UAV that has a usage role of identity, location, value or setter is created as a particular type of element, chosen to match the representation role of the data it represents. The situation object “Speed Setting”, for example, is a continuous data object, assigned the setting usage role when it was created as part of the UAV (since it provides an interaction that is used to set a speed). Continuous data objects have the “number” representation role. Because the interaction element that also has a number representation role is the Number Element, the Information View creates an instantiation of a number element into the interaction design, to represent the interaction with the speed setting of the UAV. The referent of the number element is set to the speed setting data object.

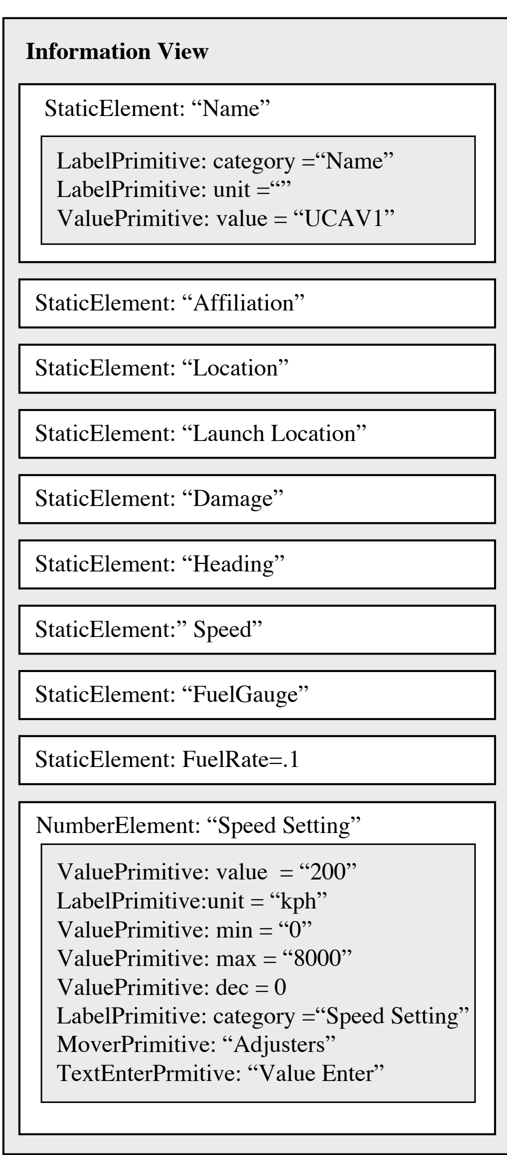

Figure 6. Example Interaction Design for an Information View Referencing a UAV

Interaction elements have a pre-defined set of primitives, which are selected if the parts of an element they represent are present in the situation. The interaction element selects the

appropriate primitives for the data object that is its referent, and fills the properties of the primitives with the appropriate values from the domain object. In Figure 6, there are two elements shown expanded into primitives; the first (a static element) and the last (the number element representing the speed setting). For a speed setting, the number element contains a number of primitives, including two label primitives providing the name of the category (in this case, "Speed Setting") and the units ("mph"). Four value primitives represent the current value (200), the decimal places (0), the maximum speed value (the maximum speed for this UAV), and the minimum speed value. A text enter primitive provides the ability to enter a new value, and a set of mover primitives allows the value to be scrolled. Each of these primitives has an appropriate referent in the domain; for example, the referent of the category label primitive is the object name data object for the referent of the number element.

A third type of role is the ***Object Role***, which identifies the type of thing an object is (to identify the subclasses of artefact, including physical object, organization, information, actor, input/output, process, and situation). These roles are used by the interaction views to specialize themselves into various specific views onto the system that are required for task support. For example, a physical thing or organization can be the focus of a two or three dimensional map view, but information is a better focus for textual or auditory information views. Object roles are also used by views to further constrain their contents; for example, information views contain all data sub-objects that match particular usage roles, but they also contain the values and setters for their sub-objects with the input/output object role.

Figure 7 provides a diagram of the relationships between domain model objects and interaction model objects and the various types of roles. This system of roles provides a useful separation between the knowledge in the interaction model and the knowledge in the domain model. As long as a domain model codes its objects and data according to the roles of those objects and data, the interaction model can be used with any domain model.

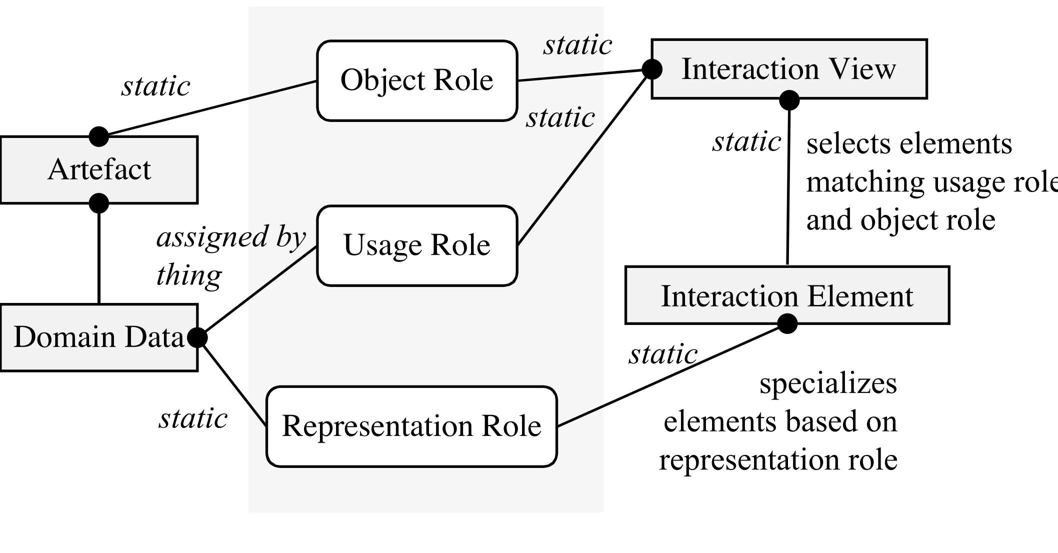

Figure 7. Relationship Between Domain Objects and Interaction Objects

Presentation Model

The presentation model is separate from the interaction model and the interaction design, in order to provide adaptation to multiple types of user interface devices. Each type of device has a defined Presentation Model; the appropriate model is selected based on the device currently in use. The Presentation Model converts the interaction design into a presentation by selecting user interface components like windows (to enclose the interaction), frames (for views or subtasks), and widgets (for elements), and specializing them according to the parameters of the objects in the Interaction Design.

In addition to the device-specific knowledge about translation from interaction objects to presentation objects, each presentation model also contains a number of heuristics that specify how to select and specialize presentation objects, and how to code and present the interaction object information (for example, how to line up widgets, add appropriate status coding, and represent objects graphically). Figure 8 presents an AID user interface design for the interaction design shown in Figure 6.

Figure 8. Example User Interface Design for Information Display with UAV Referent

In this example, the number element in Figure 6 (the speed setting for the UAV) is represented as a number spinner, which is the corresponding presentation component for a number element in the Presentation Model for a standard computer console. The category label primitive "Speed Setting" is used to label the number spinner, a colon is added to the label, and the entire label is aligned with other interactors in the presentation. A text box represents the text enter primitive, and up/down arrows the movers. The high and low limits for the speed setting are provided by the maximum and minimum value primitives, and the text box and the movers are set to accept only values within those bounds. Finally, the actual value, contained in a value primitive, is placed in the text box.

The AID system currently has very sparse presentation knowledge. It is being expanded to include presentation function knowledge that supports better visualization and interaction with the situation. For example, we are developing contiguity heuristics to guide the grouping of presentation elements within views, providing views with grouping rules based on the distance between situation objects. We are also developing grouping heuristics to provide complex presentations, which would allow AID to group the sensed speed and controlled speed settings in a way that better allowed comparison, using more complex interactors (e.g., sliders) and coding limits visually. The choice of a visual interactor to represent a complex speed control can be driven by various properties in the situation that are accessible by the presentation objects. For example, the number element's boundary values can be used to choose between a modifiable pie display and a slider. In the same way, the semantic information provided by other properties of the interaction element can be used to help design the presented user interface. For example, if the unit label primitive of a number element is "kph", the speed value and settings could be presented as traditional cockpit speed displays.

Automated Interaction Design in Use

After the interaction design is fully composed, and the user interface presentation has been composed to support it, the system maintains the interactions and responds to changes in either the situation or in the presentation (caused by input from the user). For example, consider the situation in which the user changes the speed setting for a UAV. Figure 9 illustrates the communication paradigm in AID, showing the sequence of events that occur when the user changes the speed setting value in the interface by entering a new value into the number spinner. The number spinner presentation tool receives the "change value" message from the user (step 1), and sends an "invoke" message to the interaction primitive that represents the active part of the presentation tool (step 2). This primitive (either the one representing the spinners or the one representing the text box, whichever was used to change the number) sends a "primitive changed" message to the number interaction element that it is part of (step 3).

If the interaction was with a primitive that affects only the interaction and not the domain (for example, selecting to view a different type of information about an object), steps 4 through 6 are skipped, and the domain would remain unchanged. However, the number element in this example has a referent that is affected by the user's action, so the number element notifies its referent in the domain (the continuous data object that represents the current speed of the UAV) through a "set value" message (step 4). In step 5, the continuous value in the situation changes its value, and notifies any of its event listeners (for example, all of the owners of the speed setting (the UAV itself), and other UAVs that need to be notified that the value has changed).

External changes enter the process just before step 5. Such changes can come from other system components or through physical changes in the situation, as, for example, when the UAV changes its own speed to avoid an obstacle. In this case, the same process holds, beginning at step 5. The domain object that changed notifies its event listeners, and tells the interaction element that represents it that it has changed.

After receiving a "domain changed" message (step 6) from the domain (or a "primitive changed" message from an enclosed primitive if the element causes only interaction changes), the interaction element decomposes itself and re-composes itself with a new set of primitives and primitive values that reflect the changes in its referent (steps 7 and 8). In this case, where the UAV speed setting has been changed, the value primitive that holds the actual value of the UAV

speed is set to the new value. The interaction element then sends an “interaction changed” message (step 9) to its interaction view (the information view).

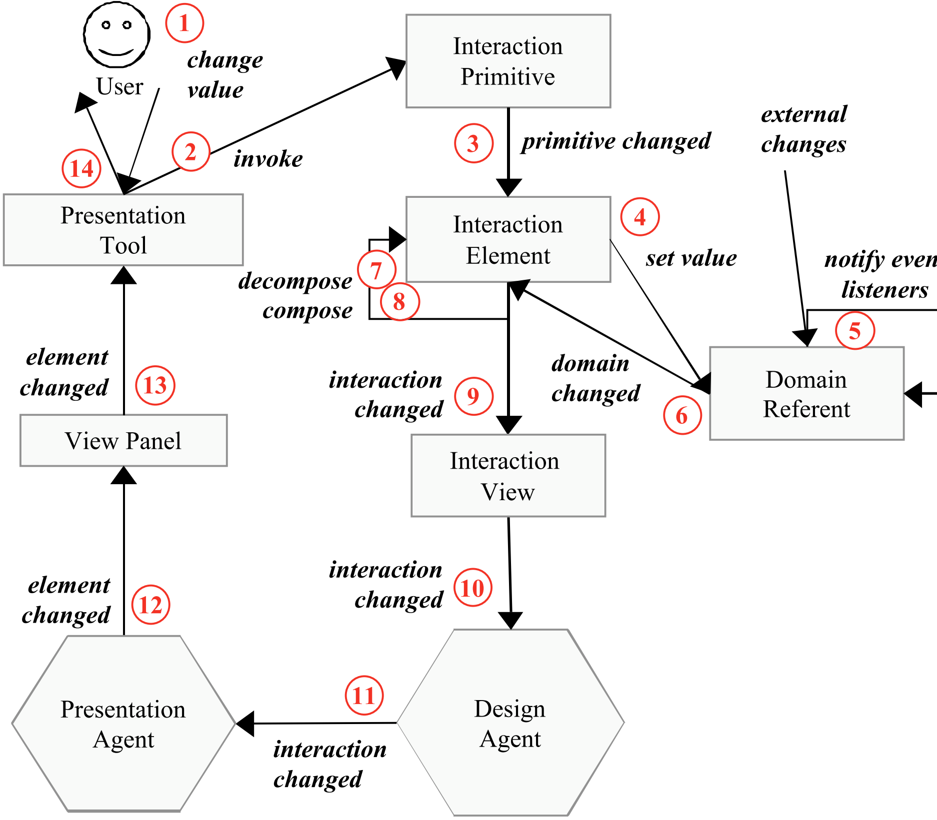

Figure 9. MAID Communication Paradigm

The change that the user made to the interaction doesn’t affect the interaction view, and so the “interaction changed” message is sent on to the Interaction Design Agent (step 10), who forwards it on to the Presentation Agent (step 11) for appropriate action. The Presentation Agent sends an “element changed” message to the presentation panel that represents the information view (step 12). The panel finds the appropriate tool matching that interaction element (the speed setting number element), and tells it to rebuild its values by sending an “element changed” message (step 13). The presentation tool then rebuilds itself with the appropriate information from the interaction element, and presents the changed information in the interface (step 14).

CONCLUSION

The implementations of model-based interaction design automation discussed here demonstrate the utility of this approach, particularly in the domains that involve complex multi-actor domains like digital control. We believe that with appropriate functional allocation to different aspects of collaborating models and a well-designed ontology for communication between these models, this approach can provide consistent, well-designed interfaces for a wide range of users and dynamic situations without the need for human intervention.

ACKNOWLEDGEMENTS

The work on DIGBE was supported by internal research grants from Honeywell, Inc. AID is supported by the Air Force Research Laboratory under Contract No. F33615-01-C-3151.

REFERENCES

Byrne, .M., Wood, S., Sukaviriya, P., Foley, J., Kieras, D. (1994) Automating User Interface Evaluation. *Proceedings of the 1994 conference on Human Factors in Computing Systems (CHI94, 232-237).*

Casner, S. (1991) A task-analytic approach to the automated design of graphic presentations. *Transactions on Graphics*, 10(2), 111-151.

Foley, J., Gibbs, C., Kim, W., and Kovacevic, S. (1988) Knowledge-based user interface management system. *Proceedings of the 1988 Conference on Human Factors in Computer Systems (CHI '88)*, 67-72.

Galitz, W. (1997) *The Essential Guide to User Interface Design*. New York: Wiley.

Jacobson, I., Booch, G., and Rumbaugh, J. (1999) *The Unified Software Development Process*. New York: Wiley.

Landauer, T. (1995) *The Trouble with Computers*. Cambridge, Mass.: MIT Press.

Mandel, T. (1997). *The Elements of User Interface Design*. New York: Wiley.

Mayhew, D. (1999) *The Usability Engineering Lifecycle*. San Francisco: Morgan Kaufman Publishers.

Nielsen, J. (1993) *Usability Engineering*. Boston: Academic Press.

Penner, R. (1992) *Consistent Honeywell Interface Design Concept for Building Management*. Minneapolis, MN: Honeywell Inc., Sensor and System Development Center, 1992.

Penner, R. (1993) Developing the process control interface. *Engineering for Human Computer Interaction*, Elsevier Scientific Press, 317-334.

Penner, R. (1994) Multimedia interfaces for process control. *Proceedings of the Energy-Sources Technology Conference, ASME petroleum Division*, 375-383.

Penner, R. (1996). Multi-agent societies for collaborative interaction. *Proceedings of the 1996 40th Annual Meeting of the Human Factors and Ergonomics Society, Part 2 of 2.* Philadelphia, Pennsylvania, Sept. 2, 762-768.

Penner, R. (1999) DIGBE: Adaptive user interface automation. *AAAI Spring Symposium*, 98-101.

Penner, R. and Soken, N. (1993) Consistent Honeywell Interface. *Scientific Honeyweller*, 106-109.

Penner, R. and Steinmetz, E. (2000) DIGBE: Adaptive user interface automation. *AAAI Spring Symposium*, Stanford, CA, 98-101.

Penner, R. and Steinmetz, E. (2002) Model-based automation of the design of user interfaces to digital control systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans. Vol 32*, No. 1, January, 41-49.

Puerta, A., Eriksson, H., Gennari, J., and Musen, M. (1994) Model-based automated generation of user interfaces. *Proceedings of the National Conference on Artificial Intelligence*, 471-477.

Rosson, M and Carroll, J. (2002) *Usability Engineering*. San Francisco, Morgan Kaufmann Publishers.

Roth, S. and Mattis, J. (1990) Data characterization for intelligent graphics presentation. *Proceedings of the Conference on Human Factors in Computing Systems (CHI '90)*, 193-200.

Seligman, D. and Feiner, S. (1991) Automated generation of intent-based 3D illustrations. *Computer Graphics. 25*(4), 123-132.

Szekely, P. (1996) Retrospective and challenges for model-based interface development. *Proceedings DSV-IS96*, Berlin, 1-27.

Szekely, P., Luo, P., and Neches, R. (1993) Beyond interface builders: Model-based interface tools. *Proceedings of INTERCHI '93*, 383-390.

Vanderdonckt, J. and Puerta, A., eds. (1999) *Computer-Aided Design of User Interfaces II*. Dordecht, The Netherlands: Kluwer Academic Publishers.

Vanderdonckt, J. (1994) Automatic generation of a user interface for highly interactive business-oriented applications. *Proceedings of CHI94*, 123-124.

Wiecha, C., Bennett, W., Boies, S., Gould, J., and Greene, S. (1989) ITS: A tool for rapidly developing interactive applications. *Transactions on Information Systems, 8*(3), 204-236.

Zhou, M., and Feiner, S. (1996) Data characterization for automatically visualizing hetrogeneous information. *Proceedings of IEEE Information Visualization '96*, 13-20.

Zhou, M., and Feiner, S. (1997) Top-down hierarchical planning of coherent visual discourse. *Proceedings of the 1997 International Conference on Intelligent User Interfaces*, 129-136.