Goals and Objectives for User Interface Software

Bill Betts, David Burlingame, Gerhard Fischer, Jim Foley, Mark Green, David Kasik, Stephen T. Kerr, Dan Olsen, James Thomas

This written report summarizes the discussions and conclusions of the goals and objectives group at the ACM/SIGGRAPH Workshop on Software Tools for User Interface Development. The report is organized into the following sections:

- Section 1 – Overview of group goals and discussions
- Section 2 – Definition and characteristics of a UIMS
- Section 3 – Criteria used to develop a taxonomy of a UIMS
- Section 4 – Tasks and tools for user interface development
- Section 5 – Suggested topics and areas of research

1. Overview of group goals and discussions

The members of the group have been both developers and end users of user interfaces. This experience provided two perspectives on the nature of user interface software. Both of these views played a major role in our discussions and the conclusions presented here.

The two primary goals of the group were to enumerate the various tasks or actions involved in the development of user interfaces and to determine the software tools that could provide automated support for these actions.

Prior to discussing the activities and software tools required for user interface development, the group discussed these topics:

- Definition of a UIMS
- Characteristics of a UIMS
- Criteria to be used in developing a taxonomy of a UIMS

The group also defined several areas and topics of research on user interfaces and UIMS that should be investigated by the academic and industrial research community.

This report documents these discussions and conclusions.

2. Definitions and characteristics of a UIMS

A User Interface Management System (UIMS) is a tool (or tool set) designed to encourage interdisciplinary cooperation in the rapid development, tailoring and management (control) of the interaction in an application domain across varying devices, interaction techniques and user interface styles. A UIMS tailors and manages (controls) user interaction in an application domain to allow for rapid and consistent development. A UIMS can be viewed as a tool for increasing programmer productivity. In this way it is similar to a fourth generation language, where the concentration is on specification instead of coding. A UIMS can be described from two different viewpoints: the viewpoint of the software developer and the viewpoint of the end user.

A software developer regards a UIMS as a tool that provides support for the definition of the user/application dialogue, imposes external control on the application, provides support for the presentation of the application's output and includes an interactive component providing support for the interaction between an application and an end user. A software developer requires that a UIMS provide a user interface with the following characteristics:

- Consistency
- Support for a range of users from novice to expert
- Support for error handling and recovery
- Support for tailorability and extensibility of application

The use of a UIMS by software developers encourages the development of better software by:

- Providing a consistent user interface between related applications
- Encouraging development and use of reusable software components
- Insulating applications from the complexities of the environment
- Shielding applications from the effects of the end user
- Supporting ease of learning and use of applications

From the viewpoint of an end user, the primary goal of a UIMS is to support the easy and effective use of an application. A UIMS provides the following advantages to an end user:

- Consistent user interface across applications
- Support for multiple levels of help or assistance
- Support for training
- Support for end user tailoring of the interface
- Support for extensibility of application

Developers of user interfaces must have software development/design skills, knowledge of human factors and experience in the area of human-computer interaction. Additional software development resources must be allocated for the design of user interfaces since this phase now becomes distinct from the application development process.

3. Criteria used to develop a taxonomy of a UIMS

This section of the report includes an enumeration and discussion of a proposed set of criteria to be used in the development of a taxonomy of a UIMS. The development of this taxonomy is required for effective evaluation of new and existing UIMSs. The inclusion of a criterion in this list does not imply that a UIMS should (or must) possess that characteristic. The criteria are organized into two groups: The first group is associated with an end user's view of a UIMS, and the second group is associated with a user interface designer's view of a UIMS.

User oriented criteria

The criteria associated with an end user's view of a UIMS are organized into these five categories:

- Dialogue tailorability
- Mechanisms for tailorability
- Interaction support
- Dialogue style
- Dialogue sequence

Dialogue tailorability

The first criterion is whether the UIMS supports the tailorability of the interface by the end user. There are at least two levels at which this tailorability can occur. The first level is the lexical level, where the user has control over where menus, windows and other interaction objects are placed on the screen, command names and some details of lexical feedback (for example whether a bell rings or the screen flashes in order to attract the user's attention). The

second level is in the structuring of the dialogue, or syntax of the interaction. At this level the user can modify existing commands and add new commands to the user interface. The new commands added to the dialogue could be combinations or old commands, or commands that add new functionality to the user interface. A UIMS may support either, both or none of these levels.

Mechanism of dialogue tailorability

The mechanisms provided for tailoring the user interface will determine the frequency and nature of the changes that can be made. For some user interfaces, such as automatic tellers, user tailorability is not desirable. In this case the UIMS should not automatically provide tailorability features. In some applications the user interface must be customizable to different user communities. These user communities could represent different language groups, countries or job functions. In this case one user (or a small group of users) will be responsible for all modifications. In other applications any user can potentially modify the user interface. This can cause some problems, and may not always be desirable. If each user has a custom interface, there could be problems with training and the transfer of knowledge between users. Customized user interfaces can cause problems when the user interface is updated to account for new program features. When this is done the modifications made by the user may no longer be applicable. The UIMS may provide features that simplify the migration of users from one version of the user interface to another. In addition to providing mechanisms for dialogue tailorability, the UIMS may also provide a means of controlling when these mechanisms can be used, and by whom.

A number of mechanisms for dialogue tailorability have been proposed. The most flexible mechanism is programming by example. Another mechanism supporting tailorability is simple textual specification by the end user. An example of using textual specification for tailorability is the definition of macros.

Interaction support

The degree of support provided by a UIMS for the interaction between an application and an end user is determined by the following capabilities:

- Help – An end user should be able to acquire help when necessary; however, the request for help should be unobtrusive.
- Embedded training – The success of an application depends on the ease of learning. Learning time is greatly reduced when training can be embedded in the user interface and can be dynamically accessed by the end user as necessary. An interface that allows an end user to simultaneously interact with an application and receive training is an effective method of embedded training.
- User error avoidance and recovery – Good user interfaces have a number of features that help the user recover from errors and unexpected actions. These features can be divided into three groups depending upon when they are used. The first group of features, called escapes, is used when the user is in the middle of specifying a command. They allow the user to gracefully back out of a command that he or she doesn't want to use. The second group, called stops, is used to terminate or suspend a command while it is executing. The third group, called undo, is used to reverse the effects of a command that has been executed.
- Defaults – The ability of the end user to use defaults as defined by the application results in a decrease in interaction time. In addition, defaults reduce the requisite knowledge that an end user must possess to understand the capabilities of an application.

Dialogue style

In order to cover a wide range of applications and user groups a UIMS should support a wide range of dialogue styles. Some of the dialogue styles that could be supported by a UIMS are:

- Simple command interaction
- Form interaction
- Question/answer interaction
- Graphical manipulation providing only lexical feedback
- Graphical manipulation providing semantic feedback
- Direct manipulation of graphical images

Ideally, a UIMS should support a range of dialogue styles from which a user can select. The sophistication of a user governs the type of dialogue style that is chosen.

Dialogue sequence

Support for free or random navigation through a dialogue enhances the usefulness and flexibility of the system. Some types of dialogue sequences that could be supported by a UIMS are as follows:

- Flat – In this type of user interface all the commands are accessible at each point in the dialogue. That is, there is no particular structure imposed on the commands.
- Hierarchy – In this type of user interface the commands are organized into a strict hierarchy. At each place in the hierarchy, only the commands at that point can be used. If the user wants to execute another command, he or she must move to the part of the hierarchy where the command is located.
- Hierarchy with limited deviation – In this case the commands are organized into a hierarchy, but the user is given more freedom in moving about the hierarchy. For example, the same command could appear at several places in the hierarchy, or there could be direct connections from one part of the hierarchy to another.
- Multi-threaded – In a multi-threaded dialogue, the specification of one command, or a sequence of commands, can be suspended while the user performs one or more unrelated commands. A common example of this is the use of a calculator to compute the value of an operand in the middle of specifying the command.
- Multi-threaded with multi-programming – This is an extension of multi-threaded dialogues that allow the user to execute several commands simultaneously. At any point in time the user may be taking part in multiple dialogue, with no dialogue suspended.

Designer oriented criteria

The previous criteria are used to characterize a UIMS from an end user's viewpoint. From a user interface designer's viewpoint, the following criteria are important in developing a taxonomy of UIMS:

- Rapid prototyping
- Reusable components
- Specification granularity
- Intelligent assistance
- Explicit semantics
- Evaluation/testing
- Open architecture
- Type of control provided (internal or external)

- Type of model supported (state, object, direct manipulation)
- Source of events/communication

Rapid prototyping

Support for rapid prototyping enhances the effectiveness of a UIMS since feedback from the end user can be obtained early in the software development life cycle. A UIMS that provides support for the transition from a rapid prototype to a final system is particularly effective since components of the rapid prototype can be reused in the final system. The support of reusability reduces the time and cost of software development.

Reusable components

A UIMS that provides support for the development and reuse of software components increases the productivity of the user interface designer. In addition, support for reusable software components encourages the development of fully tested, evaluated and documented user interface libraries. Other direct consequences of reusable software components are standardization of user interfaces and enhanced support for rapid prototyping.

Specification granularity

Specification granularity deals with the amount of detail a user interface designer provides when defining the user interface. There are two aspects to specification granularity. The first aspect is the amount of information that the user interface designer must provide before the UIMS can construct a user interface. At one end of this spectrum the designer only needs to list the operations (and their arguments) supported by the user interface and the UIMS will automatically construct a user interface. At the other end of the spectrum the designer must specify every detail of the user interface design (for example, the screen position of each interaction and display technique, and the details of the dialogue in the form of a transition diagram). The use of higher level specifications should simplify the production of user interfaces. The other aspect of specification granularity is the ability to switch from one level to another. If the first version of the user interface was produced from a high level specification, is it possible to use a lower level specification to modify fine details of the interaction? In other words, can the granularity of the specification be adjusted to support the operations the designer wants to perform.

Intelligent assistance

Intelligent assistance refers to the amount of assistance or direction that a UIMS provides in the definition of the user interface. A UIMS should encourage the creation of good designs by making tasks or activities that result in such designs easy to accomplish. Actions, activities and decisions that result in poor user interface designs should be either disallowed or difficult to accomplish. Intelligent assistance should also provide support for the management, organization and tracking of alternative designs of user interfaces.

Explicit semantics

This criterion determines the degree to which a UIMS allows the semantics of the application to be easily and formally represented in the elements of the user interface. The semantics of the application are expressed in the behavior of the objects and actions of the user interface and in the relationship between these objects and actions.

Support should be provided for the definition of the relationship between the semantics and symbology (visual representation) of the semantics. For example, the color red is associated with danger or stop, and the color green is associated with safe or continue; the UIMS should allow a user interface designer to easily define and subsequently modify this relationship without creating significant ripple effects through the user interface and the application.

Evaluation/testing

The level and degree of automated support for evaluation and testing that a UIMS provides a user interface designer is a criterion that should be addressed in developing a taxonomy of UIMS. A UIMS should provide support for testing and evaluation in all phases of the software development life cycle (requirements, specification, design, implementation and maintenance).

Open architecture

A UIMS that provides support for an open architecture by allowing the addition of other interaction techniques is more flexible, extensible and customizable by the end user than one that does not. The ability of the user interface designer to define an individual style of interaction and presentation is also an attribute of an open architecture. The style of interaction and presentation is an important commercial consideration. Most software companies have their own unique style that identifies them within the marketplace. A UIMS should be able to produce the style used by a particular company. This style should be defined once and then used in all subsequent user interfaces.

Type of control

Control of the application can be determined by the application (internal) or determined by the UIMS (external). A true UIMS provides support for external control, while a toolkit only supports internal control.

Source of events/communication

This criterion deals with the run-time communications between the user interface and the user, and between the user interface and other components of the program. The user interface must deal with the information entered by the user (via the input devices). The UIMS can view this communications in a number of ways. The simplest view is that the user must interact with only one input device, and the user interface selects the device he or she must use. An alternative to this approach is that the user can only interact with one device at a time, but the user can select the device. At the opposite end of the spectrum, the user can interact with several devices simultaneously. Note that this criterion is based on the logical view supported by the UIMS and not on how input is implemented by the UIMS.

A UIMS also supports communications between the user interface and the other parts of the application. The minimum level of interaction between the user interface and the application is the transmission of user requests, user data and application results. In addition to this normal channel, the user interface and the application program may be able to interrupt each other. This may occur when the user wants to terminate a computation that is in progress, or when an exception condition arises in the application that must be dealt with by the user. This type of communication assumes that the user interface and the application can be viewed as separate processes.

4. Tasks and tools for user interface development

The overall process of developing a user interface

follows the standard waterfall life cycle of software development. The individual phases of the waterfall life cycle are as follows:

- Requirements
- Specification
- Preliminary design
- Detailed design
- Implementation
- Testing
- Maintenance

Our discussions started with this life cycle model, but during the discussions it became clear that from a user interface tools point of view it is hard to separate the specification, preliminary design and detailed design phases. In the following discussion, these three phases will be discussed together.

Many of the activities within the life cycle and the necessary software support tools are unique to the development of user interfaces. This section summarizes these activities and tools, based on the structure provided by the waterfall life cycle model. All of these tools should be integrated to allow the sharing of information between the various phases of the software development life cycle.

Requirements phase

The primary goal of the requirements phase is to develop an understanding and subsequent documentation of the existing environment or problem domain. This goal is achieved by developing a user process model and a data model of the system. These models describe the user's views of the problem domain (both the objects of interest and how they can be manipulated), and the underlying abstractions used by the application developers. In most cases the individuals involved in this process are the end user, who can be viewed as an expert with a thorough understanding of the problem domain, and a user interface designer, who is responsible for collecting, organizing and documenting the end user's knowledge. The user interface designer must have a good understanding of human behavior and user interface technology, allowing him or her to match the user's needs with the best possible user interface.

The wording of the above statements suggest a strong similarity between the tasks of user interface design and knowledge engineering. Both of these activities attempt to formally describe the operations performed by an expert. In our case we want to use this knowledge to assist in the design of good user interface. The knowledge engineering literature could provide ideas for possible tools and techniques that could be used in the requirements phase of user interface design.

Various tools provide support for the activities involved in the development of an understanding of the problem domain. The user interface designer needs tools for knowledge collection, acquisition and management. In addition, tools are needed that allow a designer to illustrate or model to an end user the designer's perceived understanding of the problem domain. The definition or description of the problem domain could be in terms of objects, object relationships, actions, action relationships (inverses, mutual exclusion, dependencies) and visual representations of objects. Tools must be developed that provide support for this process and enable a designer to evaluate the consistency and completeness of the definition.

Rapid prototyping tools are also useful during this phase of software development, because they provide a tangible and executable version of the final system. Rapid prototypes are effective means for the user interface designer to generate feedback from the end user. However, the designer must be careful to prevent the user from prematurely accepting the view provided by the prototype as the only solution. Ideally, the user interface designer should develop several prototypes that illustrate alternative design strategies.

Specification and design phase

The information gathered in the requirements phase is used to produce a system specification, which is a formal description of the user interface to be developed. The requirements identified in the previous phase should be reflected in this specification. That is, for each requirement it should be possible to identify the parts of the specification which satisfy the requirement. When the requirements change it should be possible to identify the parts of the specification that must be updated. Tools that support this tracing of the specifications from the requirements are necessary to perform adequate verification of the specifications. These tools could also assist with updating the specifications when the requirements change.

Once the specification of the user interface is complete, the design of the user interface can begin. User interface design could benefit from a number of tools. One of these tools is rapid prototyping. There are a number of properties that a prototyping tool should have. It should be able to use the specification of the user interface as a starting point for the development of the prototype. This will have the benefits of reducing the number of notations that the designer must learn, and facilitate tracing the design back to the requirements. An effective prototyping tool should allow previously defined software components to be reused in the new prototype, thus enabling a user interface designer to quickly develop a large number of alternative designs. Prototyping tools that allow a user interface designer to refine a prototype into a completed design are more effective than those that do not allow continual refinement. Finally there should be a well defined, and potentially automated, route from the prototype to the implementation of the user interface. Once an acceptable design has been produced, there should be no need to recode the design in another notation, with the possibility of introducing errors.

Automated software tools that support the evaluation of user interface designs need to be developed. There are two aspects to this type of tool. One aspect is the detection of features that could give the users problems when they are interacting with the user interface. This type of evaluation could deal with screen layout, command syntax, consistency of the user model and usage of input devices. This type of tool could produce a rating for the user interface, or it could warn the designer of potential problems. The other aspect of design evaluation is adherence to style manuals. If a style manual has been defined for a collection of user interfaces, this tool would ensure that all the designs agreed with the manual.

Tools that can automatically produce the design of a user interface from its specification could have a significant impact on the design of user interfaces. There are several ways in which this type of tool could be used, depending upon its sophistication. An automatic design tool could be used to produce the first draft of the design. The output of

this tool should be in a form that is acceptable to a UIMS, allowing the designer to move directly from a specification to an implementation of the user interface. The designer would then refine the design until an acceptable user interface is produced. In some cases the first draft may be acceptable. Another possible use of an automatic design tool is in the evaluation of different design strategies. For example, the design tool could be instructed to design menu based, direct manipulation and command based interfaces for the same program. The designer could then evaluate each of these approaches to determine the best one. This tool could also be used to produce different interfaces for different user communities.

Finally, a tool for tracing the design back to the specification should also be developed. This tool could be used in the evaluation of the design, and updating the design when the specification changes. If automatic design tools are used, a separate tracing tool may not be required.

In summary, in the design phase we need tools for rapid prototyping, the evaluation of designs, automatic design and tracing the designs back to the specification. In addition, the design tools should encourage the definition of good user interfaces by encouraging decisions that result in a good interfaces, and discouraging decisions that result in a bad ones.

Implementation phase

The implementation of the user interface design could benefit from reusable software components. Software tools that provide support for reusability must include adequate mechanisms for the definition, storage and retrieval of reusable components. Methods to categorize and express the semantics of reusable components must be provided by the software tool, thus enabling a user interface designer to determine the existence and functionality of reusable software components.

An ultimate long-term goal in the design and implementation of user interfaces is to develop a set of tools or an environment that will enable the requirements, specifications, design and implementation phases to be highly automated. A user interface designer should be able to interact with this environment to accomplish all of the following tasks:

- Description of the problem domain
- Rapid prototyping
- Evaluation of the design including human factors criteria
- Consistency and completeness analysis
- Automatic implementation of the user interface

Testing phase

Testing of user interfaces should address both the functionality and performance of the user interface. There are two times at which testing must be performed. The first occurs during the user interface development process. In this case, all the features of the user interface must be exercised to ensure that they are functioning correctly. The second time is when modifications are made to the user interface after it has been released. In this case, the modifications must be tested to ensure that they have been performed correctly, plus all the other features of the user interface must also be tested to ensure that the modifications have not invalidated previously correct parts of the user interface. Tools should be developed to aid with this type of regression testing. For non-interactive programs a set of input data and the corresponding correct results can be used for regression testing. In the case of user interfaces this is not always possible. Most of the input comes from graphics devices such as a mouse. Mouse positions can be logged, and then played back as a means of testing, but if the screen layout is changed the logged data could invoke different operations than originally intended. Similarly, if the syntax of commands has changed, logged input data will be useless for regression testing. Determining whether the user interface is producing the correct results can also be difficult. Most of the important information produced by the user interface takes the form of images on the screen. It is very difficult to construct programs that automatically analyze the images on a screen to determine if the correct output was produced. The development of automated tools for the regression testing of user interfaces is an important research topic.

If the implementation of the user interface can be traced to the specifications of the user interface, then the automatic generation of test data may be possible. In addition, the evaluation of the test results is an activity that could be partially automated through the use of pre- and post-conditions.

The removal of errors is the ultimate goal of the testing process; therefore, tools should be developed to support debugging. An ideal debugging tool allows the simultaneous testing of the user interface and viewing of the software support code.

The user interface must also be evaluated against human factors criteria. If prototyping has been used in the requirements and design phases, then a significant amount of human factors evaluation will have already been completed by this phase of the life cycle. Human factors evaluations should be performed in all phases of the life cycle.

An effective user interface evaluation should include the logging of the user's interaction with the user interface. This logging should be correlated with the tasks being performed, thus allowing the user interface evaluator to determine where and when excessive user entry errors, requests for assistance and pauses occur. Tools that provide support for the automated analysis of the logs enable the user interface evaluator to effectively identify areas of poor design.

A considerable amount of detailed information can be produced when logging user interactions, therefore, tools must be developed to process this data and generate summarized information and statistics concerning user interaction. Such tools can act as filters for the user interface evaluator, thus shielding the evaluator from low-level details.

In addition, a feedback mechanism should be included in the design of the user interface. This feedback mechanism allows the user to provide critical comment on the user interaction process.

Maintenance phase

The maintenance phase requires careful consideration since a modification to the user interface may have a direct and adverse effect on the end user community. Modifications to the user interface are particularly problematic when users are able to customize the interface. Through customization, multiple views of the user interface can develop; user interface maintenance must take into account every possible view.

Tools must be developed that indicate to a maintainer the extent to which a modification will affect the user interface. A user interface maintainer can then evaluate,

prior to making a change, the impact that the modification will have on the end user community.

5. Suggested topics and areas of research

The information generated from a workshop affects the direction of subsequent research on the topic. One of the goals of this workshop was to suggest specific areas of user interface and UIMS research that should be addressed by industry and academia.

In general, the predominant areas of research are the development of tools that provide support for the development/evaluation of user interfaces, the development of higher-level languages for the definition and implementation of user interfaces and the development of formal taxonomies/categorizations of UIMS. Nine specific research topics are as follows:

1) Determine how artificial intelligence and knowledge engineering can support the definitions of the conceptual model of the user's problem or system being developed. In the development of a system such as a user interface, research should define a method, tool or environment to enable a knowledge engineer to present the end user with a description or conceptual model of the system. Modeling is particularly important in the development of a user interface.
2) Define and develop data structures that will provide adequate support and effective interfaces for user interface software tools.
3) Develop user interface design tools that can acquire and use higher-level knowledge about user interface concepts for particular application domains. This higher-level knowledge includes the objects, actions and relationships in specific problem domains. Design tools should allow the definition and reuse of primitives in specific application domains, thus enabling user interface designers to begin defining user interfaces in terms of abstract objects instead of lower-level programming language concepts.
4) Determine the characteristics of good user interface designs and how these characteristics can be quantified and measured. Tools should then be developed that will encourage user interface designers to define good user interfaces by making easy those decisions that result in good user interfaces and making difficult those decisions that result in bad user interfaces.
5) Develop prototyping tools that will facilitate feedback from the end users of the interface. The feedback resulting from the end user's interaction with the prototype can be used to further refine the requirements of the user interface.
6) Develop a formal taxonomy or categorization of UIMS functionality with respect to the control and data models provided by UIMS. As additional examples of UIMS are developed, a taxonomy will be important in evaluation.
7) Conduct research in the area of UIMS and distributed systems. The majority of work on UIMS has been concerned with supporting the development of applications executing on single user workstations or single processor computer systems.
8) Investigate the automatic generation of user interfaces from high-level specifications. The higher-level specifications should be defined in terms of objects, actions and relationships between objects and actions.
9) Investigate the use of conventional tools and techniques that have been defined to support the software development process. These tools include prototyping, testing, debugging, evaluation and configuration management tools.

Tools and Methodology for User Interface Development

Jim Rhyne, Roger Ehrich, John Bennett, Tom Hewett, John Sibert, Terry Bleser

1. Task force organization and objectives

The members of this task force were: John Bennett, Terry Bleser, Neil Corrigan, Roger Ehrich, Tom Hewett, Przemyslaw Prusinkiewicz, Hank Ramsey, Jim Rhyne, Kurt Schmucker, John Sibert and John Wiberg. We jointly prepared an annotated outline for this report during the workshop. The conversion of this outline to prose, after we had left Seattle, was done by Jim Rhyne, Roger Ehrich, John Bennett, Tom Hewett, John Sibert and Terry Bleser. In the process, we were again reminded of the inadequacy of present editing, networking and text formatting tools for such a task.

The objective of the task force was to characterize in a broad way the understanding of procedures for enduser interface design and development. Discussion often focused on tools to assist designers and implementers, paying particular attention to research topics of current and future interest. There was a consensus that enduser interface software proceeds through phases of requirements/solutions, design, implementation and testing common to other software and hardware developments.

The primary focus of research has been on implementation, and little progress has been made on methodology and tools for the other phases. We also lack integrated tools for use throughout all of the phases. Consequently, most of the group's effort went to requirements/solutions and design. The group members felt that the most important testing of an enduser interface was its scrutiny by the actual endusers and by human factors experts, and that this testing must occur at the earliest stages of the development, rather than at the end. Auditing of the final software is also essential, but should not be a problem when the design of the interface is cleanly specified.

Perhaps the most important software development bearing on the enduser interface has been the User Interface Management System (UIMS). Since Kasik published his research paper with this title [14], the pace of research into enduser interface development and tools has greatly accelerated. We begin with an assessment of the state of this development.