# Towards Accessible Automatically Generated Interfaces Part 1: An Input Model that Bridges the Needs of Users and Product Functionality

J. Bern Jordan(✉) and Gregg C. Vanderheiden

University of Maryland, College Park, Md., USA
{jbjordan,greggvan}@umd.edu

**Abstract.** Automatic model-based generation of user interfaces is a potential strategy to enable individuals with disabilities to control products and services with an interface that fits their specific needs. Most of the existing work and models have been focused on mainstream users and have evolved into complex, multilayered approaches. In this paper, we describe a new, simpler input model, the FIN-USI model, which is a bridge between the basic input a system/device needs for functionality (the Functionality Input Needs; FINs) and the basic input that users provide as input in an abstract, modality independent manner (User-Sensible Inputs; USIs). An abstract model of a user interface can be made up of FIN and USI elements. Each FIN and USI element consists of a type of input and characteristics that are applied to that input type. Input elements may be grouped for functionality or usability reasons. All the components of the model are described and examples of application are given in this paper.

**Keywords:** Abstract user interface model · Accessibility · Functionality needs · Input requirements · Personalization · Universal design · User input · User needs

## 1 Introduction

People with disabilities often find it difficult or impossible to complete tasks with user interfaces (UIs) that do not meet their needs or are not operable with their abilities. There are many potential strategies for creating accessible interfaces for people with disabilities [1]. Built-in interfaces may be designed using a universal/inclusive design process to try to accommodate as many users as is commercially feasible [2]. Software and web UIs may be designed in accordance with accessibility standards, such as the Web Content Accessibility Guidelines (WCAG) 2.0 [3] and application program interfaces (APIs), such as IAccessibile2, AT-SPI, UIAExpress, NSAccessibility, UIAccessibility, MSAA, UIA, and others [4–6] so that assistive technologies can be used. A manufacturer may also go so far as to create several alternative UIs for different expected user populations. All of these strategies require deep knowledge of accessible design and the resources necessary for development and testing with all of the different types, degrees, and combinations of the disabilities targeted. Furthermore, these accessibility strategies also focus on common disability groups—they are not personalized. This makes accessibility difficult to extend


J. Zhou and G. Salvendy (Eds.): ITAP 2017, Part I, LNCS 10297, pp. 129–146, 2017.
DOI: 10.1007/978-3-319-58530-7_9

to people who have needs and preferences that are uncommon or conflict with those of other users.

Automatic, model-based generation of user interfaces is potentially a powerful strategy for creating interfaces that fit each person's needs and preferences [7]. Using underlying models of a UI, an interface generator on a person's Personal Alternative User Interface (PAUI) device (such as a mobile phone running special interface software) could create an interface tailored to the user [1]. The PAUI's interface might have features for the person, such as a dynamic braille display, text-to-speech, or alternative input modalities. The one-size-fits-one approach of automatic interface generation could then provide a usable interface even if no manually-designed interfaces for the device are accessible to the individual. This paper does not specifically focus on the generation of UIs, but instead on models that could support UI generation.

Models for the generation of UIs have been developed as standards and as the output of research projects. Many of the research-oriented models suffer from being too complex and difficult to implement for their resulting benefit [8, 9] and have not been used by industry in any meaningful way. Models that have been successfully tested with users (e.g., [7, 10]) have tended to be simpler. Unfortunately, models that have been developed to this point have generally been focused on relatively limited types of interaction and input, such as input that can be handled by form-like interfaces, rather than considering a broader range of what various systems might require for input. There is a need for a simple model for user interface generation that covers a broader range of user input that systems need for operation, yet is practical enough for industry use. In this paper, we describe such a model and its development.

## 2 Background

Model-based UI generation relies on a separation between the user interface and the functionality that is to be controlled. This separation of concerns was introduced with the Seeheim model [11] and is now a common software architecture pattern that underlies Model-View-Controller [12], Model-View-Presenter [13], Model-View-ViewModel [14], and other patterns. In COUSIN [15], an early user interface management system, this separation was emphasized to automatically generate a UI from an abstract definition of the functionality. Several potential advantages were found with this approach: reduced effort for UI design, UIs with more user support, easier involvement of human factors experts, more consistent UIs, and the ability to provide multiple UIs [15]. This separation between the user interface and functionality also potentially allows for alternative UIs to communicate with the underlying system functionality. Despite seemingly potential benefits, the automatic generation of UIs has made little headway in mainstream UI development.

### 2.1 Models for User Interface Generation

A number of models have been developed to facilitate the generation of user interfaces. Many of the models that have been developed can be categorized using the CAMELEON reference framework [16, 17], which foresees models at four levels:

- *Concept & Task Models* define the tasks a user is expected to perform and the domain objects with which a person will interact.
- *Abstract UI Models* define interface elements independently of specific interaction modalities.
- *Concrete UI Models* specify the interactors (widgets) that will be used, which are targeted to a particular platform, environment, and modality.
- *Final UI* is the runnable user interface that can be executed as binary code or an interpreted language.

A user interface generator may take models at any of these levels and may incorporate other models and information about the user, the context of use, and the environment to generate the final UI. Fully automatic generation of UIs is challenging [9]. Because of this, many of the model-based interface generation research programs have limited their work to semi-automatic generation of user interfaces [8]. This hand-tuning of interfaces is unacceptable as a scalable solution for providing customized or close-fitting access to people with disabilities. Most model-based user interface projects have focused only on mainstream users, leaving the creation of interface generators for people with disabilities as future work, with two notable exceptions. In the SUPPLE project, a constraint-based approach was used to layout graphical user interfaces (GUIs) for people with physical disabilities [7]. In the MyUI project, a pattern-based approach was used to create UIs for older people interacting with specific applications on a television [18].

Early model-based systems were developed to generate interfaces because, at the time, creation of UIs was a particularly difficult process. Modern UI toolkits and visual editors were not yet available. With the Mickey system [19], basic menus and dialog boxes were generated from specialized function signatures and comments in the application logic code. The Jade system [20] utilized a basic textual specification of seven types of inputs.

As frameworks were further developed, they generally became more complex. The MASTERMIND system [21] was one of the first to include a task model, which was modeled in terms of goals and preconditions. Since then, many model-based UI generation projects have included a task model or other higher-level model of the interface. The TRIDENT system [22] relies on task models in the form of an Activity Chaining Graph. ConcurTaskTrees [23] is a hierarchical task model that is included as a component of the TERESA [24] and MARIA [25] projects. The UsiXML project uses FlowiXML [26] as a task model, which is itself an extension of ConcurTaskTrees. The task models can be helpful in improving an interface by providing information that can be used in grouping and navigation [27]. Producing task models also requires more effort, which may not be worth the investment for manufacturers.

There are some notable exceptions to the general trend towards increasing model complexity. The SUPPLE system [7] represents abstract UIs with a combination of primitive input types and container types. The Personal Universal Controller (PUC) system [28] defines interfaces in a similar way, with a limited set of input elements and a tree interface structure that can be adjusted based on dependency information. Both systems were tested with users and showed performance and other benefits over manufacturer and other one-size-fits-all UIs [7, 10].

A widely-used type of model at the Concrete and Final UI levels are GUI toolkits. Many GUI toolkits are available. Toolkit widgets may support input from multiple modalities

(e.g., keyboard, mouse, and touch input), which can make them easier to use by people with disabilities. Toolkits for GUIs may also have built-in support for accessibility APIs. Assistive technology, such as screen readers, can use these APIs in their functioning to present an alternative UI to users. Without accessibility APIs, using assistive technology can be more difficult or impossible. There are strategies, such as off-screen models [29] that can be used, but these are more difficult and may only be available for the most commonly used applications. However, basing an adapted or alternative interface on an accessible graphical user interface may result in layers of interface. For example, a person with a screen reader listens to and interacts with an audio representation (layer 1) of a graphical interface (layer 2) to control a system's functionality (layer 3). This layering and indirection can lead to unnecessary complexity and confusion.

Some interface models at different levels of abstraction have been standardized. The HyperText Markup Language (HTML) is ubiquitous on the web and evolved from a language initially for the publication and exchange of documents to one that supports complex applications. Markup in HTML5 [30] can be extended with scripts to enhance the type of applications and interactions that can be created, but this can lead to inaccessible interfaces if care is not taken. XForms [31] offers a potentially more flexible way of specifying forms with richer processing logic than HTML, but is used much less in practice. To better support assistive technology in dynamic HTML, WAI-ARIA [32] was developed and can be used to provide compatible assistive technology with enhanced metadata about elements' roles and states. The Universal Remote Console (URC; ISO/IEC 24752:2014) is a system that can discover and allow remote control of compatible target devices through an interface socket [33]. This socket is defined in an XML language with a resource layer that defines different groupings and dependencies and uses XML Schema [34] data types. Interfaces for a URC controller may be manually designed or automatically generated for simple target systems [35]. Although not a system for user interface generation, the Common Access Profile (ISO/IEC 24756:2009) uses a two-sided approach for assessing the matching quality of a system's capabilities with a user's needs. However, this approach is focused on modality-matching and does not accommodate individual user interface elements.

All the models and projects discussed so far work for form-like interfaces, where users interact with buttons and enter data into sequential fields in a time-independent manner. Form fill-in is only one of a range of interaction styles [36]. Not all interactions with systems fit this paradigm.

### 2.2 User Input Beyond Forms

Several taxonomies of user input have been created (forms are a subset of each). In an early attempt to categorize input, six elemental tasks were proposed for graphical computer systems: Select, Position, Orient, Path, Quantify, and Text [37]. These elemental tasks could potentially be accomplished through different interface techniques. This list of elemental tasks is not a list of perceptual tasks that people do on a computer, but was proposed instead as a different way to think about and categorize input for those systems. Other taxonomies of input characterize input devices rather than the tasks one might perform. Buxton focused

on continuous input devices and characterized their properties sensed and number of dimensions supported [38]. This work was extended to other devices and dimensions of input in [39].

For creating interfaces for people with disabilities a modality-independent approach, like that of [37] is more appropriate. However, it is still important to consider these other, broader taxonomies of input as well. Novel and modality-dependent input may be used to provide input that is outside the form input paradigm.

## 3 The FIN-USI Model

There were four goals for the model as it was developed.

1. *Allow for the creation of UIs that meet the needs of users.* All users have needs and preferences when interacting with UIs. People with disabilities may have different needs from others. A PAUI and UI generator must use the model and create an interface that meets these individual needs.
2. *Model a broad range of input that systems need for proper functioning.* Users need to be able to control functionality of a variety of systems and should not be limited to systems that only have form-like input requirements.
3. *Require as few base components as possible.* Complex models are more difficult to implement for both manufacturers and PAUI developers. With a limited vocabulary of required inputs that must be supported, PAUI developers do not need to create as many interactors and UI generation rules for users who have unique input needs.
4. *Allow for extensions to improve usability.* Expressiveness is limited with a small vocabulary. Optional extensions can enable richer interface generators that support a wider range of interactors that are tuned for particular types of inputs.

The user interface needs of all users fall into three broad categories [1]. All users, including people with disabilities, must be able to *perceive* all information, must *understand* that information, and must be able to *operate* all controls necessary for all functionality. Interfaces generated via a suitable UI generator on a PAUI would cover the user needs of perception and operation (i.e., a generator should only use interactors that use a modality that the individual can perceive and interactions that the individual can operate). The user's ability to understand the input they are to provide and how to do so is a function of both the interface model and generated interface.

### 3.1 Model Development Process

To create a simpler yet more comprehensive model than those that currently exist, the authors started by examining a variety of sources and literature. By better understanding the limitations and successes of other models, the best features of these models could be incorporated into the new model. The model was developed in an iterative process. First a potential model was proposed, and then real-world interfaces were modeled. When the model could not accommodate a real-world interface, the model was revisited and revised. The FIN-USI model is the result.

### 3.2 Description of the FIN-USI Model

A user interface is a bridge between the user and the functionality they wish to control. The FIN-USI model is an abstract model of input that is provided through a user interface. Like a bridge, the FIN-USI model has two ends. The Functionality Input Need (FIN) end models the input needs of the functionality of the device, software, or system. For example, what does the tuner part of a television or the refrigeration functionality of a refrigerator need as user input in its most basic, modality-independent form? The FINs are very different from what a user interface needs a user to provide. A user needs to be able to find and push the Channel Up and Down buttons on a specific remote control to tune the television, but the TV only needs to know the channel number (the FIN). With refrigerators, the user needs to find and twist a knob on the inside of one, or push buttons on another to set the temperature, but both refrigerators only need to know the settings and do not care how it is entered. If users cannot fulfill the FIN using an available user interface (because the available UI does not fit their needs), and no alternate interface is available to them, then they will be unable to use that functionality.

The User-Sensible Input (USI) end is a model of abstract, modality-independent interactors that a user needs to be able to perceive, understand, and operate in order to fulfill the system's FINs. The FIN and USI components of the model are related by transformation functions which translate user-understandable input into the system-understandable input that the device requires. In this way, USIs fulfill FINs. The double-ended aspect of the FIN-USI model is unique among interface models and accounts for situations where the needs that a system has differs from what a user needs and understands. The relationship between USI and FIN elements (where each element is an input component) is shown in Fig. 1.

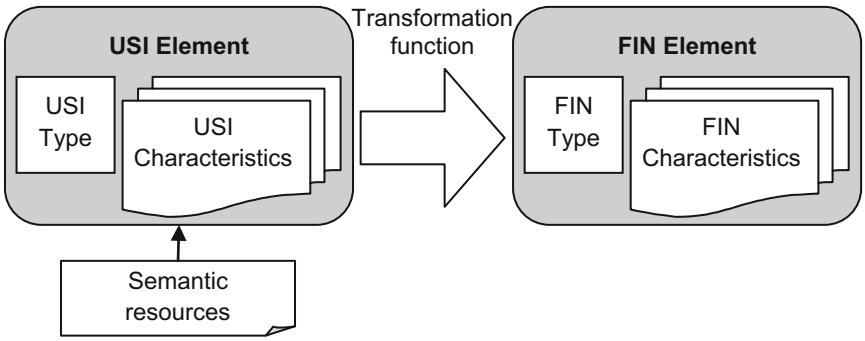


**Fig. 1.** The relationship between a USI and FIN Element and their subcomponents. The value that a user inputs to a user-sensible input (USI) element is transformed to fulfill a functionality input need (FIN). Semantic resources are applied to USI elements to make the purpose of the input understandable to users. An individual FIN or USI element is comprised of a type and any applicable characteristics.

Both the FIN and USI models have three major components:

- *Types*: The FIN Types are the raw data or other input that a system needs, whereas the USI Types are user- understandable forms of data and other input.

- *Characteristics*: The FIN/USI Characteristics define the manner in which the FIN/USI types must be provided.
- *Groups*: The FIN Groups identify data that must be grouped together for submission on the functionality end, while the USI Groups organize and provide context to the user input.

In short, the FIN/USI types are what a system needs and a user must provide, the FIN/USI characteristics are how the inputs are provided, and the FIN/USI groups are how the inputs are organized and submitted.

**FIN Types.** The FIN Types are a hierarchical organization of the types of input that functionality needs (see Fig. 2).

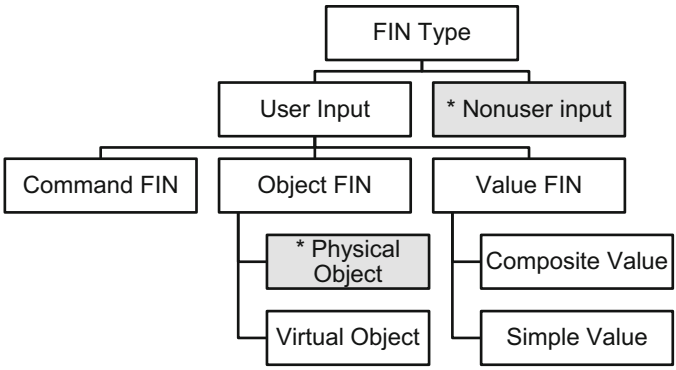


**Fig. 2.** Hierarchical class diagram of the upper levels of the Functionality Input Need (FIN) Types. Classes that are not the focus of the paper (Nonuser and Physical Object Input classes) are marked with asterisks and shaded. They are included in the FIN model for completeness.

The input to a system may come from the user or from non-user sources (e.g., data from sensors, such as GPS, from network sources, or other channels). The User Input FIN type can be further broken down into Command, Object, and Value FIN types. With Command FINs a system requires a request or command to be issued by the user for the system to enact some functionality. With Object FINs a system requires physical (objects, environments, or events in the real physical world—for example, a copier needs paper) or virtual objects (data or files) for functionality. Physical objects (such as using a biometric scanner, loading paper into a copier, having a voice recorded for speaker recognition, providing printed text for optical character recognition and translation, or use of a specific physical control) have inherent barriers to some people with disabilities that cannot easily be ameliorated with software-based changes to an interface. Value FINs are variables or data that a system accepts as input and are among the most common types of input. These values are understandable to the system and may or may not be understandable to users in their raw form. Simple Value FINs correspond to atomic data types that a system accepts, including Boolean, text, and scalar values. Composite Value FINs are comprised of other Value FINs. For example, an ISO-8601-format date-time FIN is a Scalar FIN with a representation like "2007-01-06T15:20:55-06:00". Another system might use a different date-time scale, such

as a UNIX-epoch-date-time FIN (also a Scalar FIN) with a representation like "1168118455". As another example, systems that need the user to input a color might internally use different representations of color with different color models: for example, RGB, RGBA, HSV, HSL, CMYK, and others (which are all Composite Value FINs with multiple Scalar FINs along different dimensions) or color spaces such as Pantone, which can be described with ID numbers (which is a type of Text FIN). Value FIN types are practically unlimited because they must meet the specific needs of a system's functionality which may vary broadly between systems.

An interface generator based solely on FINs does not meet Goal 1 (because FINs may not be understandable in raw form to users) or Goal 3 (because there are a practically infinite variety of FIN types). Instead, it is proposed that an interface generator use the USI Types, which are limited in number, and which can be converted with transformation functions to fulfill the FIN Types.

**USI Types.** The USI Types are a hierarchical organization of types of basic input that are understandable to users (see Fig. 3). It is restricted to user input that can be provided through software-based interactors. The USI Type model does not fulfill the Nonuser Input and Physical Object FIN Types because those needs cannot be met with the software-based interactors of PAUIs. The USI types can be used to fulfill FINs directly or with transformation functions (which are discussed later).

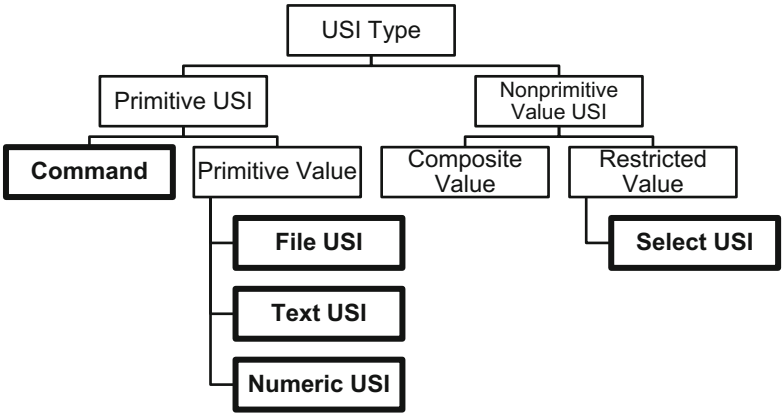


**Fig. 3.** Hierarchical class diagram of the User-Sensible Input (USI) Types. Classes marked in bold (Command, File, Text, Numeric, and Select USIs) are base USI types.

There are two categories of input in the USI model: Primitive and Nonprimitive USIs. Primitive USIs are atomic inputs that cannot be subdivided further into multiple user-sensible inputs. Nonprimitive USIs are composed of other Value USIs, and some are included as base USIs, which must be supported by all UI generators because they are very common in user interfaces.

A Primitive USI can be either a Command USI or a Primitive Value USI. The Command USI directly fulfills the Command FIN as a request or command issued by the user. In UIs, there are many ways by which users may issue a command: for example, they may click a

button or menu item on the screen, press a shortcut key combination on a keyboard, make a gesture or swipe on a touchscreen, or enter a command at a prompt.

The Primitive Value USI class is further subdivided into three types of input: File, Text, and Numeric USIs. A File USI is a reference to a data file or an entity in a specific file that a system needs as input. The File USI directly fulfills the Virtual Object FIN. A Text USI is a string of character data that a system needs to function, and directly fulfills the Text FIN. A Numeric USI is any rational number that can be represented in decimal form. With appropriate transformations, Numeric USIs can be used to fulfill many types of Scalar FINs.

The USI Types are further extensible by creating Nonprimitive USIs in two ways: by specifying validation rules which restrict the values of one of the base types (Restricted Value USIs) or by defining Composite USIs which are composed of multiple Value USIs. A Composite USI is a group of related Value USIs that can be used to model more complex FIN types that can be decomposed into individual atomic values. For example, a system may need a date in ISO 8601 extended format, such as "2009-04-04." While the date could be directly entered by the user in that form as a Text USI, in many cases, it would be better to use a Composite USI with the date split into three atomic Primitive Value USIs: year, month, and day.

Restricted Value USIs can be defined by restricting the values of Primitive Value USIs. There are many ways of restricting values, but one of the most important and common restrictions is enumerating all possible values so that users must select from that list—a Select USI. A Select USI can fulfill a Simple Value FIN directly (e.g., selecting a mode from a list) or through a name-value pair transformation function if that is more understandable to users (e.g., a user picks names from a list which are then transformed to the database ID numbers associated with each name). In mainstream UIs, there are many ways to make a selection, including choosing an option from a drop-down menu, clicking radio buttons or checkboxes on a form, keying a code for an item from a vending machine, or using a touch-tone phone to choose an option in an automated voice menu system.

Interface generators must at a minimum support all the base USI types. Interface generators may also optionally support some extended USI types with specialized interactors, especially for common types of entry with widely-used interface generators. For example, a Date Composite USI (which contains the primitive USIs: year, month, and day), could have its own interactor for some platforms and PAUIs. A specialized date-interactor might look like a calendar picker for some users and a set of dropdown menus for others. Not all generators will have specialized interactors for a given extended USI, however. In these cases, users will use the interactors associated with the base USIs.

**FIN & USI Characteristics.** The FIN and USI models share the same characteristics, which relate to the manner in which input must be provided to the system in order to be accepted.

*Input Cardinality and Order.* Input cardinality is the count of the number of individual object or value inputs that a system requires or allows. Minimum cardinality is the minimum number of values/objects that are required for valid input. FIN/USI elements with a minimum cardinality of 0 are optional. The maximum cardinality characteristic

defines the maximum number of values or objects that can be input. For example, "Vote for up to 5 candidates," has a minimum cardinality of 0 and maximum cardinality of 5.

For multiple inputs, the order of the values may or may not be important to the system's functionality—the order dependent FIN/USI characteristic. For example, the order of input is important if users are supposed to rank their choices or provide a two-dimensional path, which is a series of XY values.

*Input Time Dependence.* Time dependence is an important characteristic of input that can be applied to all FINs and USIs. Input is time dependent if the meaning or success of the user's input depends on when the input is submitted. If the same input submitted at a different time or for a different length of time, yields a different effect, it is time dependent. There are three types of time dependence: (1) *none*, where there is no time dependence; (2) *momentary*, where only the moment of time at which the command is activated or value is submitted is important; and (3) *continuous*, where values require continuous control or adjustment.

*Input Validity.* To function properly, a system needs values that are valid. There are many possible ways to validate data ranging from very simple rules to extraordinarily complex algorithms that rely on formulas that relate multiple values and internal state variables. From the system's perspective, an invalid value simply needs to be replaced by a valid one, but things are more complicated from the user's perspective. Users need to know the location of invalid values, what they did wrong, and what types of values are expected. Note that validation properties may be different between a FIN and related USIs where there is a transformation between different FIN and USI value domains.

Validation can always be handled by notifying the user what inputs have invalid values with a helpful error message. However, it is better to prevent user input errors than to have the users fix errors after making them [22]. Many interactors prevent users from entering invalid data: for example, a slider widget does not allow a user to enter a value that is outside of a defined range. Despite the practically infinite number of methods of validation, there are some that are very common. Numeric values may be bounded by maximum and minimum values and further validated by conforming to a fixed step size between subsequent valid values. Text values may be validated by being compared to minimum or maximum string lengths and to regular expression patterns. Enumeration, which forms the Select USI base type, is a particularly common and important method of validation that allows for convenient interactors that list the available choices.

**FIN & USI Groups.** All but the simplest user interfaces have groups or container elements (such as screens, windows, dialog boxes, toolbars, tabs, and fieldsets) that are used to structure and organize the elements. Some functionality requires several inputs to be provided simultaneously—these are FIN groups. USI groups may also group simultaneous input, but they can also provide structure to make a UI potentially less cluttered and easier to understand and use. Groups can be nested to further organize the input.

There are three types of groupings in the model. A *semantic group* is a USI-only group that contains USI elements that are grouped in a logical manner to provide context and aid understanding. A *simultaneous group* is a set of time-dependent FIN or USI elements that

must all be available to the user at the same time. A *submission group* is a set of non-time-dependent, value FIN or USI elements that must be submitted simultaneously to the system at the same time for validation, processing, or other reasons. Both simultaneous groups and semantic groups agglomerate elements that should be in the same interaction context (e.g., controls that must be on the screen at the same time or that make sense as a coherent whole). In contrast, submission groups do not require all elements to be in the same interaction context; the elements may instead be in a sequence, such a wizard interface, if that is what a user needs or prefers.

**USI Semantic Resources.** The USI elements are input that a user understands. However, a user also needs to know the purpose and context of their input. The functionality does not need these semantics, but users need them to understand an interface. This information can be provided to users through semantic resources: instructions, helpful error messages when invalid data is submitted, group titles, and input labels. These semantic resources are task specific and may also be specific to a particular language, culture, and output modality. Semantic resources for a USI element can potentially be provided in multiple versions, which might have different lengths and modalities (e.g., abbreviated text suitable for a small screen or dynamic braille; long, descriptive text; and icons as labels). An interface generator could choose the most appropriate resource from the multiple versions.

Labels and titles are an important way to help users understand the current context and what input they are to provide. Labels are required in accessibility standards (e.g., [5, 18]). Each USI element should have a label so the person knows the purpose of the element's input. A Composite USI must have a label for the composite and each Primitive Value USI (e.g., "Birthday" for a date Composite USI and then "Year," "Month," and "Day" for the primitive inputs).

**Transformations: FIN-USI Relationships.** A USI and a FIN may have a one-to-one relationship or there may be multiple FINs or USIs that are related to each other. A USI is sensible to users by definition; however, systems only understand the FIN-form of input, which may or may not be the same as the USI form. A transformation layer is necessary to convert one or more USIs to one or more FINs. It is envisioned that the transformation layer would be provided by the manufacturers, but third parties could potentially define alternative transformation functions from USIs to FINs. The transformation layer is not a new concept; in software, it is common to check, transform, and normalize user input into something that is useful to the system.

There are several possible relationships between FIN and USI elements. With a direct relation, the USI is the same as the FIN (identity transformation). Other USIs and FINs may have a one-to-one relationship but require a transformation from one value domain to the other. Many-to-one relationships are also possible. In the most frequent case, multiple USIs may be transformed and combined into a single FIN (e.g., the components of a date into a single date entity in the proper format). It is also possible that a single Text USI be parsed to fulfill several FINs, although parsing may be problematic if the user's input is not in the expected form. It is also possible to define multiple, alternative USIs to fulfill a FIN. An interface generator would pick the one that is the best fit for the user. For example, a system

might internally represent time in a 24-hour format: one USI might be the original 24-hour format while an alternative USI might allow for 12-hour time entry with an AM or PM designation.

### 3.3 Application of the FIN-USI Model

The two-sided nature of the FIN-USI model makes explicit the separation between the input needs of systems' functionality and what the user must do to provide those inputs. The USI-side of the model may be used as a base vocabulary of modality-independent inputs for an abstract UI. The USI elements can straightforwardly be applied to typical form input widgets found in GUIs. The USI model can also be used to describe other types of input that do not fit into a form fill-in interaction style. See Table 1 for examples of the USI model applied to different types of input.

**Table 1.** The USI model applied to various user inputs.

| Input or task *GUI widgets* | USI type | Card. Min. | Card. Max. | Order Dep. | Time Dep. |
|---|---|---|---|---|---|
| Form input: Text<br>*GUI: single & multi-line text fields, command line prompt* | Text | 0 or 1 | 1 | – | None |
| Form input: Numeric<br>*GUI: text box, spinner, or slider* | Number | 0 or 1 | 1 | – | None |
| Form input: single selection<br>*GUI: drop-down list, listbox, or radio buttons* | Select | 0 or 1 | 1 | – | None |
| Form input: multiple selection<br>*GUI: multi-select listbox or checkboxes* | Select | $n_1$ ($n_1 \geq 0$) | $n_2$ ($n_2 \geq 2$, $n_2 \geq n_1$,) | False | None |
| Form input: Date (YYYY-MM-DD)<br>*GUI: text field, text fields & drop-downs set, or calendar picker* | Composite: Number, Select, Number | 0 or 1 | 1 | – | None |
| Drawing program: Path of xy-coordinates | Composite: Number, Number | 1 | ∞ | True | None |
| Teleconferencing: Hand-raising functionality | Command | – | – | – | Momentary |
| Live auction: Bidding | Number | 1 | 1 | – | Momentary |
| Voting: Select up to 3 candidates | Select | 0 | 3 | False | None |
| Voting: Select & rank up to 3 candidates (rank voting system) | Select | 0 | 3 | True | None |
| Basic driving: simultaneous control over (1) wheel angle, (2) brake pressure, & (3) throttle. | Simultaneous group: Number, Number, Number | 1, 1, 1 | 1, 1, 1 | – | Continuous (×3) |

*For each example, the table lists the USI Type along with the USI characteristics: cardinality minimum, cardinality maximum, order dependence, and time dependence.*

### 3.4 Inherent Task-Related Barriers to Access

It has been recognized that certain types of tasks cannot currently be made accessible to people across disabilities [40]. With current technology, we do not know yet how to

create interfaces or interactors that allow access to some kinds of input, for example time-dependent input.

Time dependence can be one of the main barriers to accessibility. With some forms of input (such as scanning), there is no known way for users to provide the input in the expected time frame with the expected level of accuracy, especially when input is time dependent in a continuous manner. These barriers may be caused by a wide variety of factors that cannot easily be mitigated by changing an interface's interactors. Some people move more slowly than average because of physical disabilities. Those who must use an interface in which they navigate and step through options are typically slower than those who can directly select options or commands with a mouse or touchscreen. Some people might be using interaction styles that have their own time dependence (such as automatic scanning interfaces) and they cannot generate input at any arbitrary rate. Getting information through speech output or reading braille is inherently slower in many situations than looking at a screen and reading text.

The need for simultaneous input (i.e., FIN/USI simultaneous groupings of inputs) can also be a barrier to accessibility. These interactions are time-dependent and also require a person to manipulate two or more controls at the same time. Some people can only easily manipulate one control at a time because of paralysis or other physical disabilities. Other people can only focus on one task and control at a time and would have difficulty trying to coordinate interactions with multiple controls.

## 4 Comparison to Existing Models

Existing models for the generation of user interfaces cover basic form and data entry interfaces well. However, they have poor coverage of other forms of input. The FIN-USI model was created to address this systematic shortcoming. Table 2 is a comparison of several UI models. The USI model has fewer base input types than other models, which was a conscious decision (Goal 3). Having more input types is both a benefit and a liability: more differentiation potentially allows for finer control and tuning of interfaces to fit particular types of input, but it also increases the number of interactors that must be created in order to support the differentiated input. For people with uncommon disabilities and for those who need novel methods of interaction, it is likely that relatively few interactors will be available or be created. The number of types is not the main difference of note between the USI and other models however.

The set of FIN-USI characteristics are a novel contribution of the model. Most other models allow for multiple selections, but, except for UsiXML and limited HTML input elements, the models do not allow for multiple inputs (i.e., cardinality maximums greater than 1) of arbitrary data. The order of multiple input values (the order dependence characteristic) may be important in some cases. Only the UsiXML abstract user interface model [39] captured this consideration, but only for a specific array datatype. The time-dependence of an input is also an important consideration which may preclude some interactors and indicate others. The other models do not natively support timed input except implicitly with a command, trigger, or button input that could be used to invoke a command or submit values at a point in time. Timed input can potentially be added to

**Table 2.** A comparison of selected interface models.

| Model | Comments | Input elements in model | Cardinality support | Order Dep. support | Time Dep. support |
|---|---|---|---|---|---|
| HTML 5 [30] | A few of the input elements in HTML are redundant and serve the same purpose. | 26 input elements | / | 0 | / |
| WAI-ARIA 1.0 [32] | WAI ARIA specifies non-abstract input roles that can annotate controls. States and properties can be applied to modify their features. | 12 input roles | / | 0 | / |
| XForms 1.1 [31] | XForms has a limited number of core input elements, but may rely on additional extended datatypes, which are not specified in XForms. | 9 core input form control elements | / | 0 | / |
| XML Schema datatypes [34] | While not a standard for UI models, the XML Schema Datatypes are used in other specifications, such as that of URC (ISO/IEC: 24752:2014) | 44 built-in datatypes | / | 0 | 0 |
| UsiXML AUI model [41] | UsiXML has many layers. Applicable abstract UI layer components are considered here. | 13 abstract input interaction units | **X** | / | / |
| PUC [42] | The PUC specification language was designed to support UI generation for appliances. | 8 inputs | 0 | 0 | / |
| SUPPLE [43] | The focus of the SUPPLE project was not on modeling UIs but on generating GUIs from a declarative model. | 7 primitive inputs | / | 0 | / |
| USI | A USI input element is fully specified by a base input (or composites) and its characteristics. | 6 base inputs | **X** | **X** | **X** |

**Key: X** = complete support, / = partial support, 0 = no support for the characteristic.

some interfaces with scripting (e.g., JavaScript in HTML). However, this does not add these time dependencies to the UI model. Without these time dependencies in the UI model, a UI generator would be unable to choose interactors most appropriate to the needs of both the user and the system's functionality.

The two sides of the FIN-USI model also make explicit that users and systems may have different needs. Other models (except for the Common Access Profile) tend to take for granted that the input required by the functionality is already in a user-sensible form. In reality, systems may require input that is not user-understandable in raw form. This model underscores that transformation steps may be required for some input to be understandable to both users and systems.

## 5 Summary of User Testing

A user study with participants who were blind or blindfolded was conducted to compare interfaces that were automatically generated from the FIN-USI model against both physical interfaces and mobile-device-based interfaces that were manually-created by manufacturers. The experimental design and results are detailed in an accompanying paper (Part 2, in this volume), but are summarized here.

Two FIN-USI interface generators were created, both of which created self-voicing interfaces for a mobile PAUI device for people who could not see. Participants tried different interactors for various USI types and chose their favorite interactors. The interface generator then used each participant's favorite interactors to generate personalized interfaces for two products: a copier and a thermostat. As part of a factorial experiment, participants tried a series of tasks on four interfaces: the physical interface of the products, a manufacturer-created interface for a mobile device used in conjunction with a screen reader, and the two self-voicing auto-generated interfaces.

With a study of 24 participants (12 blind and 12 blindfolded), all the measured usability metrics indicated that participants were better with the FIN-USI auto-generated interfaces than either of the manufacturer-created ones—including one (the web-based interface for the copier) that was specifically designed to be used in conjunction with screen readers. Participants were successful more often, and felt that tasks were easier on the auto-generated interfaces. Participants also rated the auto-generated interfaces as significantly more usable and preferred over either of the manufacturer-created ones (including the one specifically designed by the manufacturer for use without sight).

## 6 Conclusion

There has been significant work done in the field of model-based UI generation. However, the complexity and general inadequacy of current UI models have hindered their real-world usage, which would have benefits for people with disabilities. Complex models make both the creation of product-specific abstract UIs and the creation of specialized interface generators more complex. Furthermore, models are inadequate when they cannot cover the range of inputs that devices require. The FIN-USI model is a simpler model of input that builds on existing work by proposing a set of characteristics that can be applied to different input types to support more paradigms of input. A simpler-to-implement, yet broader application model, such as the FIN-USI model, may be more attractive to industry. Furthermore, the USI model might itself be used as a base abstract user interface model for input to a system, or the concepts may be used in the development of new or revised abstract UI modeling languages.

**Acknowledgements.** The contents of this paper are based on work carried out with funding from the National Institute on Disability, Independent Living, and Rehabilitation Research, U.S. Department of Health and Human Services, grant number H133E080022 (RERC on Universal Interface and Information Technology Access). However, the contents do not necessarily represent the policy nor imply endorsement by the funding agencies.

## References


1. Vanderheiden, G.C.: Accessible and usable design of information and communication technologies. In: Stephanidis, C. (ed.) The Universal Access Handbook, pp. 3-1–3-26. CRC Press, Boca Raton (2009). doi:10.1201/9781420064995-c3
2. Vanderheiden, G.: Fundamental principles and priority setting for universal usability. In: Proceedings on the 2000 Conference on Universal Usability, pp. 32–37. ACM, New York (2000). doi:10.1145/355460.355469
3. Caldwell, B., Cooper, M., Reid, L.G., Vanderheiden, G., Chisholm, W., Slatin, J., White, J. (eds.): Web Content Accessibility Guidelines (WCAG) 2.0 (2008). http://www.w3.org/TR/WCAG20/
4. Gibson, B.: Enabling an accessible web 2.0. In: Proceedings of the 2007 International Cross-Disciplinary Conference on Web Accessibility (W4A), pp. 1–6. ACM, New York (2007). doi:10.1145/1243441.1243442
5. Gonzalez, A., Reid, L.G.: Platform-independent accessibility API: accessible document object model. In: Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A), pp. 63–71. ACM, New York (2005). doi:10.1145/1061811.1061824
6. Watson, L., McCathie Nevile, C.: Accessibility APIs: a key to web accessibility. http://www.smashingmagazine.com/2015/03/web-accessibility-with-accessibility-api/
7. Gajos, K.Z., Weld, D.S., Wobbrock, J.O.: Automatically generating personalized user interfaces with Supple. Artif. Intell. **174**, 910–950 (2010). doi:10.1016/j.artint.2010.05.005
8. Meixner, G., Paternò, F., Vanderdonckt, J.: Past, present, and future of model-based user interface development. i-Com **10**, 2–11 (2011). doi:10.1524/icom.2011.0026
9. Myers, B., Hudson, S.E., Pausch, R.: Past, present, and future of user interface software tools. ACM Trans. Comput. Hum. Interact. **7**, 3–28 (2000). doi:10.1145/344949.344959
10. Nichols, J., Chau, D.H., Myers, B.A.: Demonstrating the viability of automatically generated user interfaces. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1283–1292. ACM, New York (2007). doi:10.1145/1240624.1240819
11. Pfaff, G.E. (ed.): User Interface Management systems, Proceedings of IFIP/EG Workshop on UIMS. Springer, Heidelberg (1985)
12. Krasner, G.E., Pope, S.T.: A cookbook for using the model-view controller user interface paradigm in smalltalk-80. J. Object Oriented Program. **1**, 26–49 (1988)
13. Potel, M.: MVP: model-view-presenter, the Taligent programming model for C++ and Java (1996). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.189.782&rep=rep1&type=pdf
14. Smith, J.: Patterns-WPF apps with the Model-View-ViewModel design pattern. MSDN Mag. **24**, 26–49 (2009)
15. Hayes, P.J., Szekely, P.A., Lerner, R.A.: Design alternatives for user interface management systems based on experience with COUSIN. SIGCHI Bull. **16**, 169–175 (1985). doi:10.1145/1165385.317488
16. Calvary, G., Coutaz, J., Thevenin, D., Bouillon, L., Florins, M., Limbourg, Q., Souchon, N., Vanderdonckt, J., Marucci, L., Paternò, F., Santoro, C.: The CAMELEON Reference Framework (2002). http://giove.isti.cnr.it/projects/cameleon/pdf/CAMELEON%20D1.1RefFramework.pdf
17. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interact. Comput. **15**, 289–308 (2003). doi:10.1016/S0953-5438(03)00010-9

18. Peissner, M., Häbe, D., Janssen, D., Sellner, T.: MyUI: generating accessible user interfaces from multimodal design patterns. In: Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 81–90. ACM, New York (2012). doi: 10.1145/2305484.2305500
19. Olsen Jr., D.R.: A programming language basis for user interface. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 171–176. ACM, New York (1989). doi:10.1145/67449.67485
20. Vander Zanden, B., Myers, B.A.: Automatic, look-and-feel independent dialog creation for graphical user interfaces. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 27–34. ACM, New York (1990). doi:10.1145/97243.97248
21. Szekely, P.A., Sukaviriya, P.N., Castells, P., Muthukumarasamy, J., Salcher, E.: Declarative interface models for user interface construction tools: the MASTERMIND approach. In: Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, pp. 120–150. Chapman & Hall Ltd., London (1996)
22. Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Sacré, B., Vanderdonckt, J.: Towards a Systematic building of software architecture: the TRIDENT methodological guide. In: Palanque, P., Bastide, R. (eds.) Design, Specification and Verification of Interactive Systems '95, pp. 262–278. Springer, Vienna (1995)
23. Paternò, F.: Model-Based Design and Evaluation of Interactive Applications. Springer, London (2000)
24. Mori, G., Paternò, F., Santoro, C.: Design and development of multidevice user interfaces through multiple logical descriptions. IEEE Trans. Softw. Eng. **30**, 507–520 (2004). doi: 10.1109/TSE.2004.40
25. Paternò, F., Santoro, C., Spano, L.D.: MARIA: a universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput. Hum. Interact. **16**, 19:1–19:30 (2009). doi:10.1145/1614390.1614394
26. Guerrero Garcìa, J., Vanderdonckt, J., Gonzàlez Calleros, J.M.: FlowiXML: a step towards designing workflow management systems. Int. J. Web Eng. Technol. **4**, 163–182 (2008). doi: 10.1504/IJWET.2008.018096
27. Montero, F., López-Jaquero, V.: IdealXML an interaction design tool. In: Calvary, G., Pribeanu, C., Santucci, G., Vanderdonckt, J. (eds.) Computer-Aided Design of User Interfaces V, pp. 245–252. Springer, Dordrecht (2007)
28. Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R., Pignol, M.: Generating remote control interfaces for complex appliances. In: Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology, pp. 161–170. ACM, New York (2002). doi:10.1145/571985.572008
29. Schwerdtfeger, R.S.: Making the GUI talk. BYTE, 118–128 (1991)
30. Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E.D., O'Connor, E., Pfeiffer, S. (eds.): HTML5 (2014). http://www.w3.org/TR/2014/REC-html5-20141028/
31. Boyer, J.M. (ed.): XForms 1.1 (2009). http://www.w3.org/TR/2009/REC-xforms-20091020/
32. Craig, J., Cooper, M., Pappas, L., Schwerdtfeger, R., Seeman, L. (eds.): Accessible rich internet applications (WAI-ARIA) 1.0 (2014). http://www.w3.org/TR/2014/REC-wai-aria-20140320/
33. Zimmermann, G., Vanderheiden, G.: Use of user interface sockets to create naturally evolving intelligent environments. In: Proceedings of the 11th International Conference on Human-Computer Interaction (HCII 2005). Lawrence Erlbaum Associates, Las Vegas (2005)
34. Biron, P.V., Halhotra, A. (eds.): XML Schema Part 2: Datatypes Second Edition (2004). http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

35. Zimmermann, G., Jordan, J.B., Thakur, P., Gohil, Y.: GenURC: generation platform for personal and context-driven user interfaces. In: Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility, pp. 6:1–6:4. ACM, New York (2013). doi: 10.1145/2461121.2461139
36. Shneiderman, B., Plaisant, C.: Designing the user interface: strategies for effective human-computer interaction. Addison-Wesley, Boston (2010)
37. Foley, J.D., Wallace, V.L., Chan, P.: The human factors of computer graphics interaction techniques. IEEE Comput. Graph. Appl. **4**, 13–48 (1984)
38. Buxton, W.: Lexical and pragmatic considerations of input structures. SIGGRAPH Comput. Graph. **17**, 31–37 (1983)
39. Card, S.K., Mackinlay, J.D., Robertson, G.G.: The design space of input devices. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 117–124. ACM, New York (1990). doi:10.1145/97243.97263
40. Jordan, J.B., Vanderheiden, G.C.: Modality-independent interaction framework for cross-disability accessibility. In: Rau, P.L.P. (ed.) CCD 2013. LNCS, vol. 8023, pp. 218–227. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39143-9_24
41. UCL ed: D1.3: UsiXML Definition (UsiXML Deliverable) v 1.4.3 (2013). https://itea3.org/project/workpackage/document/download/1583/08026-UsiXML-WP-1-D13v3UsiXMLDefinition.pdf
42. Nichols, J., Myers, B.A., Litwack, K., Higgins, M., Hughes, J., Harris, T.K.: Describing appliance user interfaces abstractly with XML. In: Proceedings of the Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages, Gallipoli, Italy, pp. 9–16 (2004)
43. Gajos, K.: SuppleType: Supple API (2005). https://www.cs.washington.edu/ai/supple/docs/edu/washington/cs/supple/rep/SuppleType.html