Usability Properties in Dialog Models

Martijn van Welie, Gerrit C. van der Veer, Anton Eliëns

Vrije Universiteit, Department of Computer Science
De Boelelaan 1081a, 1081 HV Amsterdam, Holland
+31 20 4447788, {martijn,gerrit,eliens}@cs.vu.nl

Abstract. Usability has gained a lot of attention in the design community and it is one of main goals of every design project. Evaluating usability is usually done with end-users after a prototype has been built and there are not many techniques available that allow usability evaluation during the early design phases. Current dialog modeling techniques generally do not deal with usability aspects, as they are often functional based models, dealing only with states and state changes. This paper investigates how usability aspects can be incorporated into dialog models so that usability can be evaluated during the design process without doing usage tests. A set of measurable properties is given which together could give an indication about the usability of the design, This way, some usability aspects can be covered early in the design process without the need for an executable prototype or end-users.

Keywords. Usability, Verification, Validation, Dialog Models.

1 Introduction

In every design, the dialog between the user and the systems needs to be designed. The dialog is concerned with the structural aspects of the communication between the user and the system. At the dialog level, we are not dealing with presentational aspects such as layout and color usage. Many methods and techniques to model the dialog have been proposed but most dialog models can hardly deal with usability aspects. In [1][8]a set of properties for dialog models is given but the focus is only on aspects such as state reachability and interaction-paths. Although some of those properties can be very useful to evaluate, they only cover a very small aspect of usability. Most dialog models are state-based and do not contain information for evaluating other usability properties.

Usability is concerned with many aspects that cannot all be covered by state based models. For instance, the usability of a design strongly depends on the cognitive and motor abilities of the users together with the tasks and the goals they have to achieve. User centered design focuses on the user by involving the user heavily into the iterative design process, thereby ensuring usability. An iterative design process can be very useful but is essentially a trial-and-error kind of process and does not address the underlying problem of how to ensure usability *during* the design process itself. Formal dialog modeling could be very useful for evaluating usability early in the design process without the need of some kind of prototype or end-users. This would allow a more objective evaluation that is not highly dependent on the expertise of a usability expert. However, if a formal dialog model is used with the purpose of early usability evaluation, the dialog model and the properties to be checked *must* give a valid indication of usability. Consequently, there needs to be relation between a framework for usability and the dialog model. Obviously, not all aspects of usability can possibly be evaluated using a dialog model, so it is important to understand how much early usability evaluation is possible using dialog models.

The next sections will first sketch a framework for looking at usability and the relationship with dialog models. Then we will investigate what aspects of usability can be quantified and which cannot, in relation to a dialog model. In the last sections a new dialog modeling technique is presented that allows some usability properties, other than concerning states, to be checked. A small example is given to show what is possible.

1.1 Related Work

Usability evaluation can be done at several stages of design. Approaches such as EMA[2] and ERGOVAL[10] can be applied once a prototype exists and evaluation is done by means of real usage test. Consequently, such approaches cannot be used early in the design process when prototypes do not exist yet. Several techniques exist that model the dialog part of a design and those are used very early in the process. However, most of those techniques only allow a small set of usability properties to be evaluated. GOMS[14] is a technique that focuses on the performance by evaluating the structure of the dialog. It is based on counting the number of steps needed to perform a task and the time required for each task. The properties defined by Abowd[1] are derivable from a state based dialog model and concern state reachability. In our opinion, this is not directly related to usability but more to the correctness of the specification. ETAG[24] addresses consistency by using grammars although the focus is not on determining the level of consistency. In the following sections we also will try to define additional properties to those mentioned in the related work so that dialog models can offer more effective possibilities for early usability evaluation.

2 Usability during Design

User centered design mainly uses iterative design/prototyping as the main driving force. In itself this is a very poor method and is characterized by the "trial and error" principle. It prohibits the preservation of "design knowledge" and the same mistakes could be made repeatedly. In real-life design projects, there are many factors that influence the design process. Time and money are often constraints that prevent iteration in the design process, making the evaluation of usability during the early design stages even more important. When a project is running out of time, the last activities in the design process such as user testing are likely to be skipped. In practice, the skills and experiences of the designers and design guidelines are the main sources for design knowledge. Guidelines preserve design knowledge but skill and experience are easily lost. In [3] it is suggested that *design patterns* may capture some of this design knowledge similar to patterns in Software Engineering. The idea of capturing knowledge of good designs may be attractive but there is no agreement on what *good* design is. A good design can be defined as a design that maximizes the goals of the design within a set of given constraints. Maximizing usability may not always be the main design goal and in practice, money and marketing strategies may dominate the constraints which may lead to less attention to the usability aspects. In each design project there will be conflicting requirements, a well-known fact in requirements engineering. Therefore, it is important that the design which *is* produced and that can not be evaluated as much as desired, is still as good as possible. This can only be achieved if design knowledge is structurally incorporated into design methodology. It is a challenge for formal methods to adequately formalize this knowledge in a way that makes it easy for designers to use and facilitates early usability evaluation.

3 A View on Usability

Before usability aspects are incorporated into dialog modeling, it needs to be clear what usability is and how usability is related to the dialog part of a system. Several authors have given definitions or categorizations of usability alongside guidelines, heuristics, principles and criteria [4,8,21,22]. All the different definitions and principles make usability a confusing concept when actually designing a new system. Usually authors spent a lot of effort trying to find out what is the "best" set of principles or to define a "complete set of heuristics". Although these "aids" are useful, it remains unclear how they are related and they lack a theoretical underpinning. In [26] we introduced a model that gives more structure to the concept of usability. It is a theoretical view on usability that integrates several well-known definitions of usability and puts them into a structure. Although it is a theoretical model, it can also be of practical value when designers have to deal with actual usability problems. We will briefly summarize the structure of the model since the model helps to understand the possibilities and limitations of usability evaluation.

Figure 1 shows the layered model of usability that helps understanding the various aids. On the highest level, the ISO definition of usability is given split up in three aspects: efficiency, effectiveness and satisfaction. This level is a rather abstract way of looking at usability and is not directly applicable in practice. However, it does give three pillars for looking at usability that are based on a well-formed theory[4]. The next level contains a number of *usage indicators* which are indicators of the usability level that can actually be observed in practice when users are at work. Each of these indicators contributes to the abstract aspects of the higher level. For instance, a low error-rate contributes to a better effectiveness and good performance speed indicates good efficiency.

One level lower is the level of *means*. Means cannot be observed in user tests and are not goals by themselves whereas indicators are observable goals. The means are used in "heuristics" for improving one or more of the usage indicators and are consequently not *goals* by themselves. For instance, consistency may have a positive effect on learnability and warnings may reduce errors. On the other hand, there may be good reasons for not complying completely with a platform style.

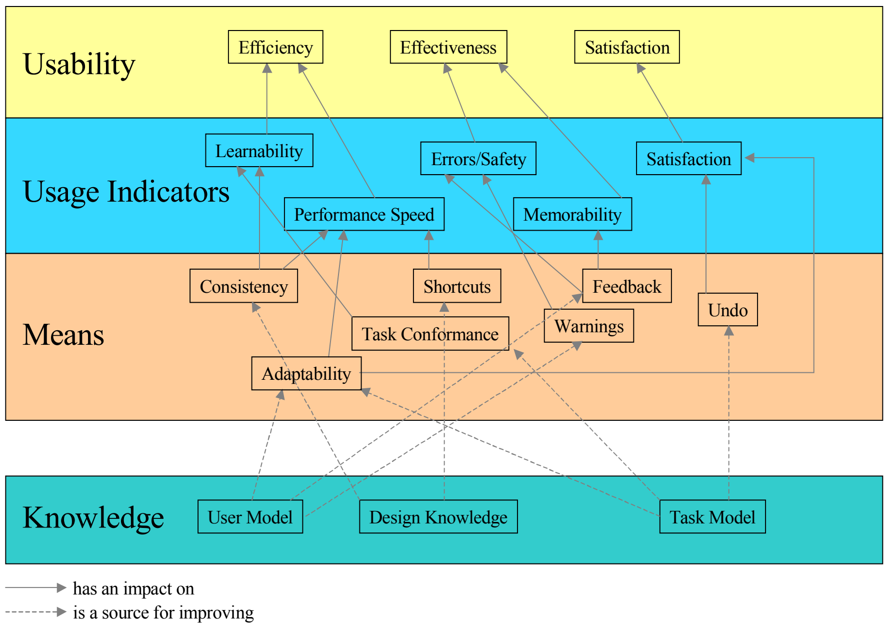

Figure 1 A layered view of usability

Each means can have a positive or negative effect on some of the indicators. The means need to be "used with care" and a designer should take care not to apply them automatically. The best usability results from an optimal use of the means where each means is at a certain "level", somewhere between "none" and "completely/ everywhere/all the time". It is up to the designer to find those optimal levels for each means. In order to do that the designer has to use the three knowledge domains (humans, design, and task) to determine the appropriate levels. For example, when design knowledge is applied by using *guidelines,* it is clear that the guidelines should embody the knowledge of *how* changes in use of the means affect the usage indicators.

The means and usage indicators of **Figure 1** are examples and stem from literature[8,17,22]. All of the guidelines and heuristics can give suggestions for other useful means and usage indicators. More research is needed to determine which means are most effective for improving usability. Knowing the impact of a means on usage indicators could then lead to more effective usage of the means.

4 Dialog Modeling and Verification

If a dialog model is to be verified on usability, the *usage indicators* in principle cannot be used since their evaluation is not possible yet, as there is no prototype. However, model based approaches *can* generate a prototype from a collection of models that among other things describe the dialog. In addition, evaluating usability with prototypes is a form of model validation and not model verification. A simulation based on the dialog model can give data about *some* of the usage indicators but there is not much foundation for conclusions; it remains unclear how aspects such as learnability or memorizability can be measured during simulations. The tool

GLEAN[13] is an example of such a simulator but its power remains restricted to the capabilities of GOMS, hence a strong focus on performance times.

The *means* are therefore more suited for usability evaluating of dialog models. The means are not directly in the right "form" because they are often in an intangible form and need to be translated into usability properties. For instance, if consistency helps improving (among others) learnability, then a property could be that "confirmation tasks" are always handled in the same way. Several of such properties can be given for each means. The properties should be measurable, either relatively or absolutely. In addition, such properties should be restricted to properties that are related to the dialog level and not the presentation level. So, "consistent use of colors for buttons" is not an appropriate usability property for usability evaluation of dialog models. Abowd[1] presents some properties for dialog models but they are mainly concerned with state reachability and they consequently do not have a strong relationship with usability.

Although the usability properties should be measurable, their values typically only have a meaning in the context of that design. Some properties may also have a general meaning like a property, "Percentage of Undoable functions" should in any design be larger than, say 50%. The values of all checked properties should be discussed within the group of stakeholders of the design. To determine what a "good" value for a property is, designers need to look at the three knowledge domains that are the basis for the *means*. For instance, a task model is a good source of knowledge in order to determine task conformance and knowledge of the cognitive processes of humans can be used to evaluate memory load or the need for feedback. As can be seen from **Figure 1** each of the knowledge domains discussed in section 3 is important for usability. Case specific attributes such as giving appropriate warnings and task conformance cannot be checked without a formal task model. Other human aspects concerning efficiency are easier to check and can even be measured using current state based models. Effectiveness and satisfaction are harder to measure than efficiency and more information concerning application functionality or the specific case is needed.

5 Dialog Models and Validation

Formally verifying a dialog model alone takes a design out of its context. The design is verified without looking at the tasks users need to do, conventions and styles that are posing constraints, or specific aspects of the intended user group. Validation of a dialog includes the context of the design and opens the possibility to look at other important usability aspects, especially concerning the effectiveness and satisfaction of a design.

5.1 Validating against User Models

Checking a dialog against a user model means looking at the dialog aspects where cognitive and motor skills and limitations are involved. Some aspects of user modeling are easier to deal with than other aspects. Considerable research into user modeling has been done to capture relevant aspects of user behavior when interacting with s system. PUMs[27] is a technique that uses both a user model and a system model to evaluate usability. PUMs does not clearly define any general usability properties and the actual formal proof of properties remains difficult[5].

5.2 Validating against Design Models

Design models only sparsely exist. Source for design knowledge can be found in guidelines, style definitions, standards and design heuristics. For formal evaluation purposes these sources are usually not formal enough and can only be used by humans, as they require a lot of interpretation. However, some attempts have been made to formalize design knowledge and "connect" it to dialog models, see EXPOSE[18] and DIADES-II[7]. The knowledge is usually a mix of dialog and presentation aspects with a strong focus on presentation aspects. Such tools can be very useful in assisting designers very early in the design process. Some tools such as ERGOVAL[10] do an automatic evaluation of a prototype using ergonomic rules. The disadvantages are that a limited amount of information can be extracted out of an executable and the necessity of an executable.

5.3 Validating against Task Models

Verifying a dialog model against a task model is not straightforward. First of all because of the diversity of task models it is not guaranteed that they model the same thing. An important issue in discussions about task models is the question what exactly they describe. Task models for model based systems and other methods like GOMS[14] and ConcurTaskTrees[19] are *prescriptive* task models. Other task models such as TKS[12] and GTA[25] are *descriptive* and focus on modeling the user's task knowledge. A consequence of this distinction is that the meaning of task and object is different. In a system's task model the objects are all part of the system which is not necessarily true for a user task model. Also with a system's task model usually the focus is on *one* user interacting with the system instead of taking into account other users and stake holders, other roles and the environment in which a user may interact with a system.

It is clear that a task model's most obvious contribution is in checking task conformance although this may not be easy to do. However, for other usability means the task model can also be a valuable source. For instance, when determining when and how often warnings should be given by the system the task model should have information about critical tasks and the frequency of those tasks. In order to make a dialog more usable for both novices advanced users, information about the tasks and the different types of users is needed from the task model. Unfortunately, not all task-modeling methods describe all those aspects of the task world[25].

Other approaches such as EMA[2] and USINE[15] use a combination of a task model or interface models and actual usage logs. The actual usage logs are analyzed against the task model. The outcome of the analysis is an annotated user log that still needs to be interpreted by an expert. The properties that are found are mainly related to deviations of user actions compared to the *prescriptive* task model. Such an evaluation is still highly subjective and the method does not provide direct clues on causes of usability problems nor on possible improvements. That has to be done by the expert. A requirement for these approaches is a prototype where logging code has been added. The models used are models that contain both dialog *and* presentational aspects. When analyzing the results it is difficult to assess whether causes of the deviations are related to the presentation aspects, to the dialog, or to the task model.

6 Towards Usability Properties for Dialog Models

In the remainder of the paper, we will focus on automatic usability evaluation based on models that describe the dialog solely and *without* doing any user testing. This will allow very early evaluation by constantly evaluating properties of dialog models that are explicitly related to usability. It would not require any software prototype and designers could make a clear distinction between dialog and presentation aspects. Before we can define any usability properties for dialog models, we need to determine what reasonable usability measures for dialog models are. As said before, usability measures need to be derived from the *means* of **Figure 1** for improving usability. A usability measure says something about the usage of a means by evaluating properties that express the means. In order to define such usability properties we have to look closer at the possible usability measures. There are several restrictions that need to be made. First of all, the usability measures need to concern the dialog only and secondly the assumptions about the behavior of the dialog components need to be restricted in order to keep evaluation as simple as possible. Some approaches such as [11] give a formal definition of the behavior of the components of the dialog. Such a definition would need to describe the complete windowing toolkit that is used and the additional project specific controls that were needed. Using that formal description, more claims can be made than without this knowledge. The connection of components to the system is however important because the basic application functionality is very relevant for usability on the dialog level. A formal description of the application functionality is not considered at the moment and only some properties of functions that are directly relevant for usability evaluation are included.

Since we are interested in expanding the use of dialog modeling for usability evaluation, we need to look further than measures that deal with interaction paths and measures such as reachability of states. In order to define properties for dialog models, it needs to be known which basic concepts should be present in a dialog model.

6.1 Some usability measures for the Dialog

This section will give a list of possible measures. Each measure is informally explained and discussed in the context of a dialog model. The measures are mostly taken from literature on usability, guidelines and modeling techniques together with some new measures. The aim was to come up with a comprehensive list of dialog related usability measures.

Interface feedback

Different functions require several kinds of feedback in response to user actions. Actually, the term feedback is not very good because it implies that all the interactions are initiated by a user, which is not true. A better term would be *interface actions* indicating that the interface acts just as well as a user, sometimes in reaction to user actions but also on its own initiative. Especially in a windowing environment, the system often initiates interaction. Consider an email program that pops up a window notifying the user a new email has arrived in which case the system needs to give feedback because of events that have occurred. For some cases of interaction, it could be stated what kind of feedback is needed.

- A calculation task requires feedback that the system is busy and the user needs to be informed of its progress.
- Functions that cannot be undone should warn the user that the result cannot be undone.
- The goal of the function is to display information.
- A function changes the system state with respect to possible next user actions.
- The system needs input so it has to indicate what user actions are needed.
- The system is in a complex activity with time characteristics that relate to temporal aspects of user input enabling/disabling

In relation to a dialog model, it is clear that it has to be identifiable when feedback or system actions occur, preferably in relation to the function type or the task type.

Forgiving the user

A user should not be punished for unintended actions, for instance when it is not directly clear to the user what a function does. Basically this means that the consequences can be undone either directly using an undo function or indirectly by going through a sequence of actions. Undo could mean that the user can go back to a previous *interface state* but also that the user goes back to the previous *application state*. Being able to go back to the previous interface state does not say too much about "forgiving the user" because the application state may not have been undone. Offering the possibility to undo complete actions may strongly impact learnability and error rate as well as the satisfaction of the user. In terms of the dialog model, it has to be known if a function is (or will be) undoable and if the previous interface state can be reached. In addition, it needs to be known if the function has *side-effects* that are not undone if the function is undone.

Consistency and platform conformability

Being consistent within one application and being consistent with a platform style also helps improves learnability and the error rate. It makes a system more predictable for the user. Consistency could be found in many aspects of a dialog:

- A message window always contains an OK button.
- Every confirmation window always contains an OK and CANCEL button
- Functions on the same object are selected in the same way.

Each platform has its own style definitions, the Apple, Windows, and Motif standards explicitly define their styles. Some aspects of a style are shared between styles and others are style specific. Concerning the dialog one could verify several properties such as:

- The number of levels of submenus. In the Apple guidelines it is stated that there should not be more than one level of submenu’s
- Closing a document window should ask for saving when needed.

- Menu entries that require additional input before execution should have (…) added to the text.

In order to find out if similar tasks are handled the same way, it has to be possible to identify small structured sequences in the dialog model. For other aspects the interactors would need to be of a certain widget type, allowing platform rules to be checked.

Total number of enabled visual functions

There is a clear difference between the number of functions that are enabled at one point and the functions that have an access path of length one. In a complex application, the number of active functions can be quite large while the number of directly accessible functions should not be very high, perhaps maximally 50. For example, imagine Word with all the toolbars active. This obviously violates this rule and is bad for usability. Reducing the number of directly accessible functions reduces the cognitive load of the user. When the user has to compare things the number 7 plus or minus 2 is a good guideline [6] but for localizing tasks where the user knows what to look for this number can be much higher, assuming the search space is in some way structured and the user knows or understands the structure[23]. The question remains if the concept of having many toolbars is bad per se or that users are just not prevented from abusing them. In a WIMP interface the fact that the button is visible is the only guarantee that the button can be selected so visibility is closely related to enabling or disabling of functions.

Interaction Path Length

Each function has a path of interactions before it is selected. The path can be a number of keypresses or mouse movement and mouseclicks. Methods such as GOMS have already tried to make prediction about the user performance based on the path length. The path length is a good indicator for the speed of performance (efficiency) and is also of interest in determining how usable the system can be for novices and experts. If shortcuts are added more paths of different length are available.

Modalness of windows

Modal windows are windows that get all device inputs so that no other windows get input. The modal window is then the active window and the user cannot change focus to another window that belongs to the same application. When a task has parallel subtasks, none of the subtasks is allowed to be modal since that would violate the parallelism. When a window is modal only the contained widgets are enabled and the rest is disabled. Modal *non-movable* windows should probably always be avoided since they even prevent the user from looking at possibly relevant information elsewhere on the screen. Modal windows are often the result of platform limitations on multitasking capabilities or because of limitations in the toolkit that is being used. They force the user into a certain state that does not allow them to do anything else and should therefore be used with caution. Although non-modal dialog windows increase the cognitive load of the user, it depends on the task whether this is desired or not. In a dialog model modalness can be detected when one component or function disables all components that are not part of it.

Preventing errors

In certain tasks it may be good to warn the user for the consequences. Typically, these are tasks that cannot be undone by the application such as tasks or actions towards the network or any other external process. For instance, opening a valve in an oil refinery is not simply undone by closing the value and the operator has to use other means to deal with the resulting situation. When the task can be undone, no warning is necessary in general. When dealing with critical task a warning is also good to make the user more aware of his decision. In such a task undoing the function may take to much time and lead to hazardous situations. When a warning is needed depends on the task of the user, the type of user and the system function.

Task type classifications and interface feedback

Both tasks done by the user and tasks done by the system can be classified. Paterno gives a classification of task types in his TLIM[19] method. For instance a calculating task may involve progress feedback. Similar a formfilling task may involve help on the meaning of the form fields. Application tasks types include report, compare, give information, locate, group of data, control, store/retrieve etc. User task types could be to select, edit, control (confirm), ask for help, navigate etc.

Adaptability of function access

A simplistic view on an application would regard an application as a collection of functions that can be used. Interaction styles make these functions available. However, it may often be wise to provide more than one access path to the function so that for instance advanced users can bypass a menu using a shortcut. Even better would be to have functionality to change the contents of toolbars so that the user can adapt his access paths.

6.2 Extracting the concepts

A dialog model that allows usability to be evaluated needs to be based on a set of concepts that enable the usability to be evaluated. Basically every formal method has a set of concepts that it uses and those concepts should be exactly what is needed in terms of the purpose of the modeling technique. The usability measures that were described show that more concepts than bare states and state changes are needed. The following aspects seem important for usability evaluating of a dialog model:

- Task/Function Typing
- Interaction paths
- Feedback
- Enabling and disabling of interaction objects
- Visibility of interaction objects
- The type of an interaction object
- Function undo-ability and side-effects

Once a dialog model can describe these aspects, we can define usability properties concerning those concepts.

7 DIMUSE

In this section a first sketch of a new dialog modeling method is given that allows some usability properties to be evaluated. DIMUSE (DIalog Modeling for USability Evaluation) is based the concept of interactors[9] and is extended to be able to deal with usability evaluation. The purpose of this dialog modeling technique is to allow usability evaluation so that design choices can be evaluated by their impact on usability properties. The concepts and relationships of DIMUSE were based on the list of possible usability measures that were described in the previous section.

7.1 Basic Concepts

The basic concepts of DIMUSE are interactors, functions, events, and actions. The dialog consists of a structure of interactors on which the user can perform actions. In the end, the actions lead to the execution of a function. In return a function can also perform actions and thereby give feedback to the user. Both functions and interactors are typed. An event is given by the system but is not directly triggered by an action of the user. Essential for the evaluation is that the actions are specified in sufficient detail to determine the usability properties. In addition some extra information about the functions is also needed such as the possibility to undo this function. Sometimes the possibility to undo a function is a design choice but sometimes it may be a given constraint; sending an email just cannot be undone. In other cases it may be undoable but the side effects are not, consider closing/opening a valve in an oil refinery; closing the valve does not undo the fact that fluid has passed through.

7.2 An Example

In order to give an idea of what could be evaluated using DIMUSE a small example will be given. **Figure 2** shows a fraction of a dialog model that describes an email program similar to most commonly found email programs. There is a list of all emails and an email can be viewed by clicking on a list item, which causes a new window to appear: the email viewer. The email viewer has several buttons which lead to the selection of a function.

Interactors can handle several *actions* such as activation and deactivation. One of the parameters of actions is the input device action that triggered the activation and deactivation. In other cases it can be a condition of the application state or interface state. Both the user and the application can be the initiator of interface changes. User initiated changes are actions starting in interactors and application initiated changes start in events or functions.

Figure 2 is a sketch of how a graphical representation could look like. At this point, we are still in the process of determining the exact definitions of the concepts and attributes that are needed to enable usability evaluation. After that, a formal definition of the modeling language and the usability properties can be given.

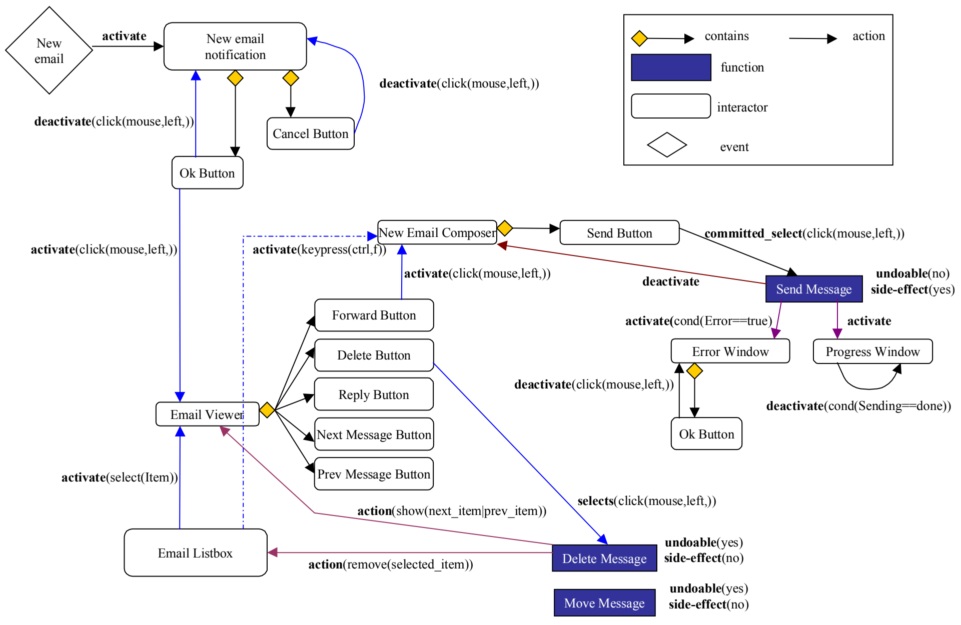

Figure 2 Part of the dialog of an email progam

Using **Figure 2** we can define how the properties of section 6.1 can be detected. For instance, interaction path can be determined by counting the number of action arrows from one interactor to a function. By inspecting activations of interactors, following actions selecting a function we can give meaning to feedback. Re-use of interactors in multiple locations of the dialog together with the containment structure of interactors, can be used to determine consistency properties. Because the dialog model also contains basic information about functions, it is also possible to detect if warnings are given for functions that have side effects.

7.3 Static Property Evaluation

One way of evaluating this dialog model is by determining static properties. The model is taken "as-is" and the value of a property is checked. Checking static properties says something about the "value" of the *means* as described in section 3. It is therefore not valid to make assumptions about the usage indicators, and consequently usability itself. However, such properties can be useful to *compare* design variants. For most means, it is intuitively clear that their value should be within certain ranges. For instance, most designers would agree that it is better to have the possibility to undo "some" actions than no such possibility. Properties can also concern structures within the dialog model. For instance, TAG[20] addresses consistency by using feature grammars; tasks that have a similar structure only differ in features and the user only has to learn the structure once to use all of the similar tasks. From guidelines and experiences from designers, it should be possible to give some "on average" values for most properties. The properties from [1] and other evaluations of state models only evaluate static properties. Typically these are properties about state reachability which is often supported by some model checking tools. The difficulty here is that in order to give a *valid* indication of the usability of the design, state-reachability does not give much justification. Using the example some other properties can be checked as well, see Table 1. The properties that are given are of general nature, i.e. they could be applied to any dialog model. There may also be non-general properties that can be useful for usability evaluation but those need to be determined per project.

Maximum number of items in a menu
Maximum number of interactors visible to the user
Percentage of undo-able functions
Percentage of warnings given for not undo-able functions
The number of undo-able function with side-effects
The number of functions without any feedback
The number of different paths to one function
The minimum path length to a function
The number of functions reachable without using the mouse
Number of possible deadlocks
The number of unreachable functions

Table 1 Examples of static usability properties

It is clear that it is possible to evaluate more usability properties than only those concerning state reachability. The examples given are still rather simple but they give an indication of what is possible. None of the examples has taken advantage of interactor typing or function typing. Especially using interactor typing, some of the design guidelines concerning for instance menu structuring could be defined as properties.

7.4 Dynamic Property Evaluation

Although it seems to be problematic to make predictions about the usability of a dialog, it is possible to get some data about more dynamic properties related to usage indicators. One way is by making some basic assumptions about users' behavior. Essentially this is what GOMS[14] was based on; the assumption that tasks take an average amount of time depending on the type of task. Using that assumption, a prediction about the speed of performance could be made. Again the absolute value was always under discussion but at least designs could be compared using this assumption and the most favorable could be chosen. Other assumptions could concern the differences in behavior of novices and expert users. A novice uses less shortcuts than an expert and for each a ratio could be defined between shortcut use and menu usage. Using that assumption one can calculate the expected number of interaction steps needed to perform and action and consequently the performance gain. Going even further one could make assumptions about the chances that a user selects the wrong menu or button. Consequently, one could calculate how often one would select a function that cannot be undone. For those kind of functions extra warnings could be useful.

Estimated path-length for an novice/advanced user
Estimated time to complete a task
Estimated number of errors that are not undo-able
Estimated memory load

Table 2 Examples of dynamic properties

However, assumptions underlying dynamic properties must be approached with caution. Such assumptions must be validated empirically before simulations are possible. Ideally, one would like to have a *simulator* that can be used to evaluate dialog models on a large set of properties both static and dynamic. However at this point it is too early to construct such a simulator and as long as there is not enough information about human behavior to make valid assumptions for a simulator, there cannot be a valid simulator.

8 Discussion

The proposed model is essentially an extension to the existing dialog models. It allows somewhat more verification than plain state based models although it still questionable how good the properties are as an indicator of usability. Usability has many aspects and if we *only* look at the dialog level we certainly do not cover all aspects of usability. At the presentation level there are also a lot of details that can be checked and even some tools have been produced[16]. Especially a style definition can help in checking the platform style conformance which is an important aspect of usability when it comes to aspects such as predictability and learning time. Besides the fact that presentation contributes a lot to usability, dialog and presentation are not completely independent. Consider a dialog describing is a menubar with 8 menu's in a window but the font size is large and it is used on a handheld PC with a resolution of 320x200 pixels so that only the first 5 menus are visible. In that case, there is no problem that may be detected on the dialog level or on the presentation level but together there is a problem.

For all of the static properties it holds, that their absolute value does not give a valid usability indication. Interpretation would become easier if we would know more about their values because of lessons learnt in practice. Such knowledge could be captured in heuristics that would help interpret static properties. Another possibility is to relate the static property to other models such as a user or a task model. How to do this is unclear and especially when relating to for instance *descriptive* task models.

Modeling the dialog remains a very time consuming and nontrivial task but if the resulting dialog model can be used for usability evaluation, the usability of the final outcome is definitely improved and the development time could be shortened, making it worth to engage in the modeling activity.

9 Conclusions

This paper has discussed the possibilities for early usability evaluation of dialog models without the need for prototypes. Using a framework of usability, we have argued that it is possible to do early usability evaluation by determining usability properties in dialog models. Although these properties can be objectively determined, there is still little ground for valid conclusions since their values need to be interpreted in their context. Consequently, the usability properties are more valuable when comparing alternative dialog models than for determining the absolute level of usability. In order to determine the usability properties, a suitable dialog modeling technique is needed since none of the existing techniques fully allow the properties to be determined. We have outlined such a new modeling technique which we will develop further in the near future.

References

1. Abowd, G. D., Wan, H. M., and Monk, A. F. (1995), *A Formal Technique for Automated Dialogue Development,* DIS '95, Ann Arbor MI.

2. Balbo, S. (1994), *EMA: Automatic Analysis Mechanism for the Ergonomic Evaluation of User Interfaces*, CSIRO Technical report 96/44.

3. Bayle, E. (1998), *Putting it All Together: Towards a Pattern Language for Interaction Design*, SIGCHI Bulletin, vol 30, no. 1, pp.17-24.

4. Bevan, N. (1994), *Guidance on Usability*, ISO 9241-11 Ergonomic Requirements for Office Work With VDTs..

5. Butterworth, R., Blandford, A., and Duke, D. (1998), *The Role of Formal Proof in Modeling Interactive Behavior,* DSV-IS, Abingdon, UK, Springer-Verlag.

6. Card, S.K., Moran, T.P. and Newell, A. (1983), *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Ass, Hillsdale.

7. Dilli, I. and Hoffmann, H. J. (1994), *DIADES-II, a multi-agent user interface design approach with an integrated assesment component,* CHI'94 HCI Bibliography, SIG on Tools for Working with Guidelines.

8. Dix, A., Abowd, G., Beale, R. and Finlay, J. (1998), *Human-Computer Interaction*, Prentice Hall Europe, 1998

9. Duke, D., Faconti, F., Harrison, M. D., and Paternó, F. (1994), *Unifying Views of Interactors,* Proceedings of the Workshop on Advanced Visual Interfaces, Bari, ACM Press.

10. Farenc, C., Palanque, P., and Vanderdonckt, J. (1995), *User Interface Evaluation: is it still usable ?,* Proceedings of 6th International Conference on Human-Computer Interaction HCI International'95, Yokohama, Elsevier Science, Amsterdam.

11. Hussey, Andrew and Carrington, David (1998), *Which Widgets? Deriving Implementations from User-Interface Specifications,* DSV-IS, Abingdon, UK, Springer Verlag.

12. Johnson, P., Johnson, H., Waddington, R. and Shouls, A. (1988), *Task-Related Knowledge Structures: Analysis, Modeling and Application*, in: Jones, D. M. and Winder, R., People and Computers IV pp. 35-62, University Press, Cambridge.

13. Kieras, D. E., Wood, S. D., Abotel, K., and Hornof, A. (1995), *GLEAN: A Computer-Based Tool for Rapid GOMS Model Usability Evaluation of User Interface Designs,* Proceedings of UIST '95, Pittsburgh, PA, ACM Press.

14. Kieras, D. and Polson, P.G. (1985), *An approach to the formal analysis of user complexity*, International Journal of Man-Machine Studies, vol 22, no. 365-394.

15. Lecerof, A. and Paterno, F. (1998), *Automatic Support for Usability Evaluation*, IEEE Transactions on Software Engineering, vol 24, no. 10, pp.863-888.

16. Mahajan, R. and Shneiderman, B. (1995), *A Familiy of User Interface Consistency Checking Tools*, CS-TR-3472.

17. Nielsen, J. (1993), *Usability Engineering*, Academic Press, London, 1993

18. P. Gorny (1995), *EXPOSE, HCI-Counseling for User Interface Design,* Human Computer Interaction - Interact '95, Lillehammer, Norway, Chapman & Hall.

19. Paterno, F. D., Mancini, C., and Meniconi, S. (1997), *ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models,* Proceedings of Interact '97, Sydney, Chapman & Hall.

20. Payne, S.J. and Green, T.R.G. (1989), *Task-Action Grammar: the model and its developments*, in: Diaper, D., Task Analysis for Human-Computer Interaction , Ellis Horwood, Cambridge MA.

21. Scapin, D.L. and Bastien, J.M.C. (1997), *Ergonomic criteria for evaluating the ergonomic quality of interactive systems*, Behaviour & Information Technology, vol 16, no. 4/5, pp.220-231.

22. Shneiderman, B. (1998), *Designing the User Interface*, Addison-Wesley Publishing Company, USA, 1998

23. Smith and Mosier (1986), *Guidelines for Designing User Interface Software*, MITRE, 1986

24. Tauber, M. J. (1990), *ETAG: Extended Task Action Grammar - a language for the description of the user's task language,* Proceedings of INTERACT '90, Amsterdam, Elsevier, Amsterdam.

25. van Welie, M., van der Veer, G. C., and Eliëns, A. (1998), *An Ontology for Task World Models,* Proceedings of DSV-IS98, Abingdon UK, Springer-Verlag, Wien.

26. van Welie, M., van der Veer, G. C., and Eliëns, A. (1999), *Breaking down Usability,* Proceedings of Interact '99, Edinburgh, Scotland.

27. Young, R. M., Green, T. R. G., and Simon, T. (1989), *Programmable user models for predictive evaluation of interface designs,* CHI '89 Conference Proceedings: Human factors in Computings Systems, ACM Press.