Modelling Human-Computer Interaction from an Agential Perspective

0203669

Abstract

This paper argues for an agent orientated approach to modelling human computer interaction, through the use of widgets. These are small fragments of a script that are used to help a human interact with a computer model. An example implementation, the Agent Orientated Widget Toolkit, is presented along with its notation. This is then compared to an object orientated approach to writing a 'user Interface' toolkit.

1 Introduction

The arena of human-computer interaction is the subject of much study. The basis of the field is firmly rooted in psychology, ergonomics and graphic design. This is often problematic for a classical, computer science, approach to programming. Consider the following example.

How does Java's Swing toolkit support the elementary principles of usability?

Swing is modelled on Object Orientated Programming Principles. In this each widget is represented by one object. Interaction between objects comes in the form of visual placement and of event handling. In the event handling mechanism a thread runs in the background, receiving messages from the system about keyboard and mouse input, sends event messages to event handlers that are registered with each object which execute a java method.

The elementary principles of usability are briefly presented below: (J, 1994)

1. Mapping - the relationship between controls are their effects in the world should be obvious and understandable.

2. Observability - the state of the program and the appropriate functions in a given state should be observable by the user.

3. Feedback - Information should be returned to the user about what actions have been done and what is has accomplished.

4. Forgiveness - users will make mistakes, so the system should should allow account for this through state reversal and warning messages.

5. Progressive Disclosure - increased functionality should be modelled and availible only as the user becomes more experienced.

6. Consistency - designing interfaces to have similar operations and similar elements for achieving similar tasks.

The astute observer will have already recognised the point: that Swing's toolkit offerings and HCI theory are almost a complete disjunct of each other. This is not to say that you cannot build a good user interface in Swing, there are examples. It is to say that swing does not offer any support for building a good user interface - it simply presents you with tools in order to accomplish that goal. It fails to offer a model that allows one to unify their understanding of HCI with a programming model. Simultaneously the traditional HCI principles have no place within a Computer Science universe: it is impossible to unify the HCI model with that of classical programming because their domain ontologies are completely disjunct. In other words there are no unifying concepts that exist within both.

It is also important to note that the elementary principles are all situated and also emphasise the role of the user's experience in regards to the computational system. These are ideas more in line with the paradigm of the Radical Empirical Modelling agenda than traditional Computer Science.

This paper presents an Agent orientated perspective on the HCI agenda, which attempts to resolve, at least, some of these conflicts. The compatibility of these concepts is enabled by the agent orientated nature of the empirical modelling research programme, and its ability to deconstruct cognition into set of modes of observation that tie agents together using dependancy.

2 An Agent orientated Model

W.M.Beynon and A.A.Pasko (1994) presents a specification for the modelling and visualising the manipulation of complex mathematical entities. The specification of this paper includes the notion of specifying the user's cognitive interaction with the model, and thus enables the paper to precisely model the nature of the interaction between a computer and a human. A generalisation of this interaction involves several steps. The user views a visualisation of their model on screen. The user's mental state is updated to accomodate what they have seen. The user's interaction with the model depends upon the state of their mind, and at this point the user may change something in the model, or his curiosity may be satisfied. The model's internal agency and dependancy updates what the user see.

This process can be viewed as analogous to the traditional view of HCI, where a user's mental model of what they are seeing is updated and their responses depend upon this model. The beauty of the above conception of Pasko's is that it allows to be explicit, in EM terms, about the HCI. The state of the model depends upon the user's agency. The state of the user agent depends upon the model. Therefore any attempt, in the empirical modelling paradigm, to perform HCI can be conceived as creating a dependance between the model's observables and the user, or vice-versa through a visual means.

It is also important to note at this point that focusing upon the user's agency does not preclude the development of N-Agent systems (where $N > 1$). Here the user can be viewed as an agent in the system, as any other. Whilst the other agents are tied together through computed observables, as in the Tkeden system, the dependancies on and from the user are visualised and interpreted in a more sophisticated manner. Conceptually, however, they remain identical.

3 Visualing Dependancy

Given the above theorising about the nature of the HCI, we can derive two categories of interaction to visualise. Those in which the state of the user agent is dependant upon that of the system, and those in which the state of a system observable is dependant upon the user agent. The entities used to visualise the dependancies (Widgets) in the former category shall be referred to as dependant widgets, and the latter category interactive widgets.

For example, a progress bar would be a dependant widget, since it simply visualises an observable describing the progress of some action. A textbox would be an example of an interactive widget, since here the user agent writes the text in the box. The example of a textbox also identifies another characteristic of interactive widgets - they frequently also act as dependant widgets, but visualising their own observable. This is because, whilst computational agents can always be sure of the value of an observable (assuming no memory corruption occurs), in order to meet HCI principles we must make the state of the model observable to the user agent.

4 An introduction to the Toolkit

Due to the nature and scope of this project, only a small selection of widgets have been written. Due to the openness of Empirical Modelling at no stage in time could the toolkit be considered complete, however, even given a standard conception of completeness the widget toolkit could not be considered complete. The implementation of the widget toolkit has progressed to the stage, however, that it can be considered a complete proof of the concepts involved in its design, and of the modelling of user interaction. Currently five widgets have been implemented, along with an accompanying notation.

4.1 Widgets

4.1.1 Progress Bar

The progress bar is the only dependant widget that has been implemented within the toolkit. It attempts to visualise a numerical observable on a scale. Despite its name, the semantics of this widget should be considered interpretive. It can be used to demonstrate more than just progress along some spectrum. For example a model of a boiler may utilise this widget in order to visualise the pressure levels inside the boiler.

4.1.2 Push Button

The push button offers an interaction with a boolean observable. Its visual response is to change upon being pushed. The key difference between the way in which this visualises a boolean observable and the Radio Option (see below) is here the nature of the interactive visualisation is a change in the viewing upon the change in state of the observable. The Radio Option, however, offers different visuals dependant upon the state of its observable.

There is a subtle difference here that is important to understanding the visual semantic relation. A difference that the author only understood after modelling

this push button, and the radio option. The modeller may wish to understand differing types of things for observables with different meanings, but the same type. It is, therefore, important to offer differing ways of visualising an observable of a certain type. Whilst the push button may be considered a more imperative interaction and the radio button a more dependant one, it is important that we recognise that such a distinction is only analogous, or metaphorical at best. It this distinction is not real, only the experience is real, and the existence of both these widgets adds richness to this experience.

4.1.3 Discrete Slider

The discrete slider offers interaction with an integer observable. When the user agent clicks within the box, the horizontal position of the click determines the value of this observable. This allows an analogue 'feel' to the interaction with the numerical observable. This kind of widget could be useful in models that require the setting of some observable, which affect the underlying rules in the model - for example setting the gravity in a physics simulation.

4.1.4 Radio Option

The radio button offers an alternative visualisation of a boolean observable to the Push Button (see above).

4.1.5 Textbox

The textbox allows the input of character strings. Whilst there is a textbox within the Scout notation, this textbox can be specifically placed using normal scout coordinates.

4.2 Notation

The key notion within the AOWT notation is that of observation. This is the keyword used to refer to the idea of visualising a dependancy. If a widget observes an observable, then it attempts to visualise a dependancy for the user agent upon that observable. For example the following: displays a progress bar,

Progress { p , pp } observes val

that visualises the observable val. Terms in braces are definitive relationships to properties of the widget. Here, p is the point representing the top left hand corner of the widget, pp the bottom right - this ordering and meaning is a common convention. Below are the remaining Statements that the AOWT consists of.

1. val observes DSlider { p , pp } - Interactive widgets observe variables, this creates creates a discrete slider using the same point notion, observing the variable val.

2. val observes TextBox { p , pp } - The semantics of the variables are identical to the DSlider except that val is a string, a textbox is created.

3. val observes Button { t , tt , p , pp } - Creates a button widget. Here t is the text when the widget is up, tt the text when the widget is down.

4. val observes Option { t , p , pp } - t is the label of the option. val a boolean observable It is on true when the Option's radio is black, false when it is lightblue.

4.3 Implementation Details

A broad overview of the implementation is described here, in addition to some interesting specifics that shed light on what it is to write definitive scripts. Each widget is supplied as a function that generates a string of eden commands to execute, creating the widget. These functions are passed up the parse tree in the notation from the matching rules using the '$v' variable provided by the AOP. Once at the top of the parse tree, the function can be coalesced with its arguments and executed. The implementation makes use of EDEN, the AOP, SCOUT & DONALD in this manner.

5 Comparison

Here a comparison between the AOWT and Swing is presented. Since this comparison is being made across paradigm lines it is necessary to compare in terms of a broader and more flexible system of understanding - that of philosophy. This has already been used as a system of comparison between traditional programming paradigms (imperative and declarative), software engineering methodologies and database models. There have also been published attempts to use philosophy to gain deeper understanding into some of the conceptions within Computer Science by academics such as Cantwell Smith. After a comparison of the ontology and epistemology of the AOWT & Swing, a more standard comparison suggesting the advantages of each is offered.

5.1 Ontology

Ontology is the underlying conception within metaphysics - that of the state of existence of an entity, or within some domain. An ontological comparison within this context compares the nature of existence of the widgets in the Empirically Modelled AOWT with those in Object Orientated Swing Toolkit.

It is generally accepted (Descartes, 1641) that cognition is a process that exists, within the empirically paradigm cognition is inconceivable without the notions of agency and dependancy. Therefore these must exist. This is furthered by the explicit nature of agency and dependancy within definitive scripts. It is also clear that the model itself exists notionally, since it is referrant of an entity within what some may term the neumenal world. (Kant, 1787) The model also exists within phenomenal world of the modeller, and if he/she has done his/her correctly then it should have the same nature.

The clearest different when comparing with a standard object orientated toolkit, such as swing is that there exists a user interface within the domain ontology of swing. This acts as a barrier through which all user interaction must pass. In other words, within empirical modelling, the model of the machine is the model within the mind of the user, whilst within a normal 'User Interface' there exists a 'mental model' (J, 1994) through which user conceptions of the system are formed. It is important to note that the AOWT is not a user interface, and it is in this sense that an empirical model can reach beyond the bounds that the Turing Machine imposes upon traditional computation.

The objects within the Swing toolkit are of particular interest. Some programming theorists define an object as a collection of methods and variables. Some define an object as a collection of methods and variables about something. It is the existence of this referrent which one knows exists that determines whether one can assert anything about the ontological status of a programming object or not.

The important difference between the object orientated approach and agent orientated approach, therefore, is that whilst the object orientated approach emphasizes the existance of a user interface, that it models with objects of questionable ontological status, the agent orientated approach affirms the existence of its model, breaking the barriers between the modeller's mind that of the modelling environment.

5.2 Epistemology

Epistemology is the study of the what it means to know something and how knowledge is accumulated. Within the context of this comparison the epistemology compares the extent to which the widgets can be said to accumulate knowledge, depend upon knowledge, and the nature of their knowledge.

Since it is sometimes difficult to state whether objects exist, it is even more difficult to state the nature of their knowledge. It is especially important to recognise that since the user interface widgets that are provided by swing are not references to neumenal objects they can know nothing. The user and the program both hold their knowledge seperately - the user in a mental model of its interaction with the program and the program in an explicit and formal manner - through the storage of binary encoded sequences within a computer's memory. The user interface merely allows the exchange of some of this knowledge.

The modelling of user interaction definitively allows the knowledge of user to me embedded within that of the model, through the existance of the observables. It is also true that any agents within the model can be understand to have knowledge: their knowledge is simply their mode of observation upon the knowledge of the user. It is through the experience that the dependancy based interaction gives that allows all the agents (including the modeller) to gain knowledge. It is also important to understand the manner in which this knowledge is gained: the dependancies that the AOWT provide are situated within the model. The knowledge is gained within the model: whether the modeller chooses to generalise from this experiement and investigation is up to them, it is not a necessary condition of the creation of the model.

It is important to recognise that these distinctions are subtle, but definite. In terms of their implementation details, both Swing and the AOWT have a thread that updates variables, and executes machine code in order to facilitate changes with their respective systems of user interaction. The important thing is that as a model of user interaction swing forces one to program against this raw implementation, not focussing the interaction around dependancy to and from a user agent. It is this distinction that underlies the epistemic differences, because in order for it to be knowledge, and not just data an agent has to be performing the 'knowing', or there must be some theoretical framework in which this knowledge can exist. Swing has neither of these. In reality there is a user agent in swing, but because swing doesn't try to to model this

user agent, it doesn't exist within the swing domain ontology, and cannot be there to underly the epistemology.

Therefore we can say that whilst a traditional user interface provides a bridge through which knowledge may pass, the AOWT empirically generates knowledge, both for the modeller and the agents that the modeller has created.

5.3 Pros/Cons

Since the comparison provided between Swing & the AOWT is one of binary opposition, I have presented the pros of each of the toolkits.

5.3.1 Agent Orientated Pros

1. Simple - integrating the widgets follows the standard agency and dependancy patterns within Empirical Modelling, and does not require addition design patterns such as observers and event listeners. It also does not require any semantic fudges, such as anonymous inner classes.

2. Provides a Model - User interaction clearly requires some paradigm in order to operate within, in order to support the process of designing good user interfaces. The agency/dependancy style in which the AOWT is implemented provides this model, whilst not forcing the modeller into some enforced formalism.

3. Unification with HCI - the ability to situate the computation with an agent orientated perspective, and give the modeller a paradigm in which to construct their interaction with the user enables one to unify the perspectives of HCI and programming within a modelling world.

5.3.2 Swing Pros

1. Performance - Java offers substantial performance advantages over the scripted, dependancy maintaining approach of Tkeden. These advantages can be furthered for fully compiled languages running with compiled widget toolkits (eg C with GTK).

2. Sophistication - Swing offers layout management, different font rendering, the option to have multiple windows open - none of which the AOWT currently posses.

3. Development Tools - Swing has sophisticated interface builders that allow one to graphically manipulate and build its user interfaces. The AOWT lacks these.

4. More and More varied Widgets - Since the AOWT is primarily at proof of concept stage it lacks a lot of the widgets that swing has and also lacks a lot of the variety that swing widgets possess.

6 Conclusion

An important observation to make when comparing in terms of the pros of each toolkit over each other is that all of the Swing pros are practical in consideration. In other words: they are problems for the AOWT that are soluble, given the necessary human resources. It is in this light that the AOWT should be seen: it is not a solution to the problems of HCI, but it should be a step in the right direction. The AOWT & Empirical Modelling provides a paradigm in which good application of HCI principles can occur. This is enabled by situating the modeller in a perspective where the interaction of the user agent with the system is a part of the system - not an interface to the system.

It is also important to understand the semantic relation that has been visually developed here. The rich and interactive nature of the widgets means something to the modeller. This is why there are two ways of visualising and interaction with a boolean observable. Each of the two interactions means something different. The semantics here are not those of formal logics, given to a proof by some modality. It is a meaning that is to be interpreted by the user agent, as part of their interaction with then model. The ability to view the interaction between the user agent and the model definitively is enlightening, accordingly. Since the view of semantics has become one of interpretation, the existence of the AOWT and its associated notation are not simply syntactic sugar: they mean something that Swing could not.

References

R. Descartes. Meditations of first philosophy. *Various*, 1:0, 1641.

P. J. Fry. Empirical modelling. *Journal of the Future*, 3000:1–42, 2005.

Preece J. Human-computer interaction. *Addison-Wesley Books*, 1:0, 1994.

I. Kant. Critique of pure reason. *Various*, 1:0, 1787.

V.D.Adzhiev W.M.Beynon and A.A.Pasko. Interactive geometric modelling based on r-functions. *Proc CSG'94: Set-Theoretic Solid Modelling*, 1: 253–272, 1994.