Application of agent architecture to modelling human-computer interactions

Elizabeth Pollitzer and Jacinto A. Da'vila

Department of Computing
Imperial College
180 Queens Gate
London SW7 2BZ
eep@doc.ic.ac.uk
j.davila@doc.ic.ac.uk

Abstract

We outline an approach to the analysis of user interactions with a computer system based on a logic programming agent architecture which unifies planning and reactive behaviours. The agent model deploys definitions for the reduction of goals to subgoals, integrity constraints for condition-action rule behaviour, and uses Abductive Event Calculus to represent and reason about actions and change. This computational environment is particularly well matched to the requirements of engineering models of human performance deployed in human-computer interaction design. These models aim to produce a priori quantitative predictions of execution time, learning time and errors, at an earlier stage in the development than prototyping and user testing. The most mature of the engineering models is GOMS, which stands for: **g**oals (what the user wants to accomplish), **o**perators (permissible actions on an interface), **m**ethods (sequences of actions needed to accomplish a goal), and **s**election rules (to chose a most appropriate method). The GOMS-style analysis can be easily accommodated within the agent model, and, furthermore, extended in scope to cover some critical aspects of usability problems, for example the role of feedback, as well as details of interface constructs structure.

1. Introduction

The GOMS model was introduced in the early 1980's as a technique for interface analysis [1]. The model uses the concepts of **g**oals, **o**perators, **m**ethods and **s**election rules to represent the procedural knowledge a user needs in order to accomplish a task with the help of a given computer system. Briefly, goals is what the user wants to accomplish, and for which the computer system is used; operators are the basic actions that the system allows user to take; methods are sequences of operators that achieve some subgoal; and selection rules are used to decide which method is most appropriate in a given situation. A number of extensions have been proposed to GOMS since its introduction. These include NGOMSL (Natural GOMS Language) [2], a structured natural language used to represent the user's methods and selection rules that easily translates into computer-based representations. Another significant extension is CPM-GOMS [3], which adds **c**ognitive, **p**erceptual and **m**otor operators and critical-path method schedule chart (PERT chart) to enable modelling parallel activities.

The GOMS model is currently the most mature of the engineering models of human performance deployed for the analysis of human-computer interactions. Its main goal is to produce a priori quantitative predictions of human performance (e.g. execution time, learning time, errors) that can be used to assess the effectiveness of an interface design at the earliest stages in the computer system development process. The GOMS models have been successfully applied to the analysis of perceptual-motor interactions (e.g. the effects of layout on keystroking or mouse pointing); the complexity and

efficiency of interface procedures; and the relationship between interface design and procedural aspects of task performance [4,5]. The development of appropriate computer-based tools can significantly further the use of engineering approaches to the design of interactive systems [6,7].
The agent architecture described in this paper, deploys definitions for the reduction of goals to subgoals, integrity constraints as condition-action rules for reactive behaviour, and uses the Abductive Event Calculus to represent and reason about actions and change. These computational techniques are particularly well matched to the requirements of modelling user interactions with an interface. One advantage is that simulation of the outcomes of the interaction process can be obtained early in a design process. A further advantage is that the scope of design analysis can be extended to include sources of important usability problems, such as lack of timely and accurate feedback.
The reminder of the paper is structured as follows. In Section 2 we describe the key problems in modelling user interactions, in Section 3 we introduce the major components of the agent architecture, and show how the requirements of modelling user interactions can be accommodated in this environment. In Section 4 we discuss the issues raised for computer-based tools. Section 5 concludes the paper.

2. Modelling human-computer interaction

It is common to separate the interaction process into the input side and the output side. At the input side, we distinguish three levels at which the interactions take place. Firstly, at the application level are user goals and tasks, for example wanting to 'edit a text' with the help of a given word processing software. Second, goals at application level are performed by executing one or more specific interface actions, which require selecting from the available interface constructs and those that help accomplish the task, for example, selecting the "Cut" command from the "Edit" pull down menu. Third, the chosen interface actions have to be executed with the help of available interaction techniques, which specify the actual physical actions possible, such as "click-left-mouse-button". At the output side, the user's interactions involve first perceiving the output, then interpreting its meaning and finally evaluating it with respect to the goal. The main objective of interface design analysis is to establish how effectively the behaviour of interface constructs supports the accomplishment of user goals.

2.1 GOMS representation

The GOMS analysis of user interactions is primarily concerned with defining goal-subgoals structure, operator sequences and task execution time. In general, expert behaviour is assumed, which means that the user has complete familiarity with the task and the relevant methods, and can decide which method to use in what circumstance. An example of a NGOMSL decomposition of a goal of moving a text using the cut-and-paste method is shown in Figure 1. The example is adapted from [5], where a more extensive explanation can be found. The example demonstrates the basic constraints of GOMS style analyses. In particular, we observe that:

a) Only the durations of physical actions are specified (which need to be obtained empirically)

b) The outcomes of the actions are not specified, or for how long do their effects persist at the interface. For example, it is not stated for how long the mouse cursor stays in selected position.

c) Certain mental actions, e.g. retrieving command names from memory, are included in the methods (since they are not directly observable no time duration are provided). Their success can only be assumed.

d) The order of actions in a sequence is fixed, with a (sub)goal being implicit in the sequence of actions that defines procedures for it (this is part of the expert behaviour assumption).
e) The definition of the goal/subgoal structure assumes timely availability of certain interface design features.
f) No pauses or errors are allowed, which means that each action in a sequence is assumed to immediately follow a preceding one. Given some starting point, each action is expected to occur at some specific later time. Actions cannot be reversed.
g) The position of the cursor is a crucial constraint on the execution of certain interface-level actions (this reflects the fact that the interaction takes place in the context of a WIMPS style interface design (**W**indows, **I**cons, **M**ice, **P**ull-down menus and **S**election)). It is implicit that the cursor persist in a position until some other selection action is performed.

NGOMSL Statements		**Execution**	**Operator Times**
Method for goal:	Move text	1	
Step 1.	Accomplish goal: Cut text	1	
Step 2.	Accomplish goal: Paste text	1	
Step 3.	Return with goal accomplished	1	
Method for goal:	Cut text	1	
Step 1.	Accomplish goal: Select text	1	
Step 2.	Remember that the goal is Cut, and accomplish goal: Issue a command	1	
Step3.	Return with goal accomplished	1	
Method for goal:	Paste text	1	
Step 1	Accomplish goal: Select insertion point	1	
Step 2	Remember that command is Paste, and	1	
Step 3	accomplish goal: Issue a command	1	
Method for goal:	Select text	1	
Step 1	Determine start position of text	1	1.20
Step 2	Move cursor to start position	1	1.10
Step 3	Click mouse button	1	0.20
Step 4	Determine end position of text (already known)	1	0.00
Step 5	Move cursor to end position	1	1.10
Step 6	Shift-click mouse button	1	0.48
Step 7	Verify that correct text was selected	1	1.20
Step 8	Return with goal accomplished	1	
Method for goal:	Issue a command	1	
Step 1	Recall command name and retrieve from memory the menu name for it and retain the menu name	1	
Step 2	Recall the menu name, and move cursor to it on menu bar	1	1.10
Step 3	Press mouse button down	1	0.10
Step 4	Recall command name, and move cursor to it	1	1.10
Step 5	Recall command name, and verify that it is selected	1	1.20
Step 6	Release mouse button	1	0.10
Step 7	Forget menu name, forget command name and return with goal accomplished	1	

Figure 1. An example of NGOMSL methods for moving text. Adapted from [5].

One useful way in which the GOMS analysis can be extended is to allow for the possibility of user errors occurring. Clearly, novice users are expected to make mistakes, however, evidence is that experts make errors too [8], and there is significant body of examples implicating human error as a cause of computer system malfunction and failure [9].

2.2. Usability problems and human error

When interacting with a computer system a user may experience problems at either the input or the output side of the interaction process. The possible causes of human errors have been suggested in [10]. They include the possibility that

a) an action was omitted/ taken too early/taken too late;
b) an action had too little effect/too much effect
c) an action took too long/not long enough
d) right action was taken on wrong object/wrong action on right object
e) input/output information was not obtained/transmitted
f) wrong input/output information was obtained/transmitted
g) input/output check was omitted/done on wrong object/mistimed/wrong check

There may, of course be many possible contributing sources of human error, for example fatigue, stressful working environment, lack of training, that have little to do with interface design itself. There is a suggestive evidence that interface design may include on an average around 40 usability problems [11], though these may vary from 9 to 140 in number. A usual way of eliminating usability problems is to conduct heuristic evaluation and user testing. The difficulty with heuristic evaluation is that the method requires usability specialists, who may not be readily available, and there is evidence that more than one such specialist is needed [12,13] to gain agreement on the problems present. A recent study [14] compared the explanatory power of 101 usability heuristics, composed of the various sets of heuristics that have been proposed in the usability literature for interface evaluation. The heuristics were tested on a database of 249 usability problems. This study identified seven design factors that appear to form an essential part in promoting effective interaction:

1) visibility of system status, e.g. providing timely and accurate feedback
2) match between system and real world, e.g. concepts and terminology must be familiar to user
3) user control and freedom, e.g. ability to reverse actions
4) consistency and standards, e.g. expressing same thing in the same way
5) prevent errors and help users recognise, diagnose and recover from errors,
6) recognition rather than recall, e.g. making the repertoire of available actions salient
7) flexibility and efficiency of use, e.g. provide shortcuts and accelerators for frequent actions

We explore how the problems identified in [10] and [14] can be accommodated as part of modelling and analysis of user interactions with an interface. As context, we use interactions with a WIMPS interface .

2.3 WIMPS representation of user actions

The useful property of the WIMPS interface style is that the principal concepts underlying most interactions involve one or more combinations of four basic components, free of implementation detail:

a) "select this" (e.g. point and click to select an object)
b) "open this" (e.g. point and double click on a selected object)

c) "put this there" (e.g. select an object and drag-drop it to a new position)
d) "do this to that" (e.g. identify an operation - "this" - and apply it to a selected item - "that")

An example is given in Figure 2, showing how step by step instructions in a manual (which include implementation detail) can be represented using the set of more general action descriptions listed above, with some additions (underlined). The example is adapted from [15].

00. Task: Set up limits on prices	
01 Click on **Setup** on the menu bar, then **Limit Minders** in the pull-down menu (this displays a dialogue box)	Select and Open **Setup** Select and Open **Limit Minders**
02. Type a *RIC* code (**do not** press *RETURN*)	Select and Type a *RIC* code
03. Select BID from the Field box. The current value for the RIC is shown in a small display window	Select BID from the Filed box
04. Click on the down arrow to change the displayed value by a few points. Try to choose a value that the RIC might realistically fall to	Select **down arrow**, Select *Change*,
05. Click on **Lower**	Select button **Lower**
07. Click on >>**Add**>>. The values you have chosen are added to the list of current limits at the bottom of the dialogue box. If you wish, set an upper limit in the same way (click on **Upper**, then click on the upward arrow to raise the value to the upper limit, then click on >>**Add**>>).	Select Add and Apply *Choices* to *RIC*
08. Click **Exi**t to remove the dialogue box.	Select **Exit** button and Apply Exit to dialogue box
09. If the value of the instrument moves outside either limit, a warning will appear in the alerts box. For example **ANBS.L** **351 BID** **Break 351 Upper Limit**	Select and Open **alert box**

Figure 2. An example of an interaction process which is part of the Reuter system used to support trading in financial instruments. The example was adapted from [15].

The interactions between a user and a WIMPS style interface can be perceived as a directed path through the space of available interface constructs (e.g. menu items, buttons) and task-related objects (user files and their contents). Traditionally, GOMS models have focused on the time taken by an expert user to traverse this path. The scope of GOMS analysis can be significantly extended in the agent-based approach, for example to consider relationships between user actions and design features that are intended to deal with feedback and error . Certain important points to note about the WIMPS style is that the positions of the mouse pointer have to match the positions of the interface constructs that enable execution of the intended task-related actions. The pointer 'persists' in the selected position until a new selection is made. Another key point is that each menu and a window has own constructs associated with it (e.g. menu command names, scroll buttons), which are made visible and ready for a user action when the menu or window has been selected and opened. These features of interaction can be easily represented within the agent architecture.

3. Applying a logic-based agent architecture to modelling user interactions

A logic programming agent architecture that unifies rationality and reactivity has been proposed recently in [16]. The proof procedure uses definitions for "rational" reduction of goals to subgoals and integrity constraints for reactive, condition-action rule behaviour. This agent model has now been implemented [17]. The resulting computational environment, called GLORIA, which supports the use of the event calculus for the representation of actions and change, matches the requirements of GOMS-style user interaction models, and helps to extend interaction modelling to include consideration of user error and of certain important usability problem relating to the role of feedback. In the following sections we demonstrate how the agent's use of event calculus and integrity constraints can be deployed to model user interaction with an interface.

3.1 Using Event Calculus to represent user actions

The Event Calculus was first defined in [18] as a formalism for representing and reasoning about the occurrence of events, the properties that events initiate and terminate, and the maximal time periods for which those properties hold [19]. Further developments focused on a simplified variant [20,21,22,23,24,25,26] which employed time points instead of time periods. The key intuition of the simplified event calculus is that a property (in the world) holds if an event has happened to initiate it and, after the event, nothing has happened to terminate it. The occurrence of an event can initiate or terminate fluents, depending on the action associated with the event. A fluent is a property that can change as time passes. Domain dependent axioms are needed to define the *initiates*, *terminates* and *isfluent* predicates.

3.2 Representing user actions in GLORIA

In the agent model [17] the principal intuitions behind event calculus are informally represented as

a fluent P holds at time T

if someone does an action of type A between T' and T1
and an action of type A ending at T1 initiates the fluent P
and T1 < T
and fluent P is not clipped between T1 and T

fluent P is clipped between T1 and T2

if someone does an action of type A between T' and T
and an action of type A ending at T terminates fluent P
and T1 < T and T ≤ T2

The basic WIMPS interface actions: "select this", "open this", "put this there", "do this to that" can have the following fluents associated with them. The action *select (Item, Source)* has the effect that a particular interface construct becomes *selected (Item, Source)* for further action. The action *open(Item, Source)* has the effect that the selected construct is *opened(Item, Content),* which, if the construct is a window or a menu, means making visible and available for action a further set of interface constructs or task objects. An opposite effect can be obtained with a *close* action. The action *put (Item, Position)* has the effect that an object is placed *at(Item, Position)*, the new location. The basic action *launch(Operation, Item)* (representing "do this to that") has the effect that the intended operation selected by the user was *executed(Operation, Item)*, by the system.

In GLORIA, formal description of actions and properties and the relationships between action-types and property-types is part of the agent's background theory. A background theory consists of two sub-theories. The domain-independent axioms, notably the base case of *done* and the definition of *hold* stating how actions and properties interact. Informally, *done(A,To,Tf)* reads "An action of type A is started at To and completed at Tf". The predicate *hold(P, T)* reads *fluent P holds at T.* The other component of the background theory is a set of domain dependent axioms, describing the particular properties, actions and inter-relationships that characterise a domain of application (including the definitions of the predicates *initiates, terminates and isfluent*). A more detailed explanation of background theory can be found in [17].

We can now demonstrate how the *initiates* and *terminates* predicates can be defined in GLORIA for a WIMPS interface.

```
initiates(select(X, Y), T, currentselected(X)) <---  precond(select(X, Y), T)

precond(select(X,Y), T)  <---   holds(pointerpos(X1), T)
                                holds(objectpos(X,Y), T)
                                and X1 = X
```

The intended meaning is that for the *select* action to succeed, a fluent must hold whereby the position of mouse pointer has to match the position of the selected object.

```
initiates(open(X, Y), T, opened(X, Z))   <---   precond(open(X, Y), T)

 precond(open(X, Y), T)  <---  holds(currentselected(X, Y), T)
```

The intended meaning is that an object cannot be opened unless it was selected first.

```
initiates(launch(X, Y), T, P)  <---   precond(launch(X, Y), T)
                                      and   postcond(launch(X, Y), P)

precond(launch(X, Y), T)  <---  holds(currentselected(X, Y1), T)
                                and   holds(currentselected(Y, Z), T)


precond(launch(X, Y), T)   <---  holds(opened(Y, Z), T
                                 and   holds(currentselected(X, Z1), T)
```

The intended meaning of the first *precond* definition is that both the object and the operation that will be performed on it must be selected first (as for example when the 'Cut' command in the 'Edit' menu is used to remove a repeated word in a body of text being edited). The second definition deals with a more global use of functions (as for example, when the 'Close' command in the 'File' menu is used to close an active dialogue box).
The *postcond* predicate with the fluents that must be initiated by the user action that launches an operation will have to be defined for each function.

initiates(put(X,Y), T, at(X,Y)) <--- precond(put(X, Y), T)

precond(put(X, Y), T) <--- holds(currentselected(X, Z), T)
and holds(opened(Y,Z1), T)

The intended meaning is that before an object can be placed in a new location, that location has to be available and the object must be selected. (An example is moving a file from a floppy disk to a folder on a hard disk, using the drag and drop method)

The same approach is taken to define the *terminates* predicates.

3.3 Simulating an interaction process in GLORIA

A simulation of the interaction process between a user and an interface design can be achieved in GLORIA by specifying a plan of user actions on the interface design associated with a given user task. The provision of feedback for each action can also be taken care of. One solution may be to implement feedback as integrity constraints. Another solution may be to have two interacting agents, one simulating the behaviour of the user and the other simulating the behaviour of the interface. At the present state of its implementation, GLORIA is better suited to the first solution. However, the second solution is achievable [16,17].
We show a partial plan of actions needed for a successful accomplishment of the task given in Figure 2. We indicate what would be the corresponding actions of the interface agent, which take place immediately after the completion of an action by the user agent. The interface agent has its own set of basic actions, which include "highlight' and "display", for example, which provide feedback information). We assume that with the exception of the times associated with the occurrence of actions, all other variables are universally quantified.

exists t3 do(user, select(Setup, Menubar), t1, t2) ---> 01
do(interface, highlight(Setup), t2, t3)

exists t5 do(user, open(Setup, Menubar), t3, t4) ---> 02
do(interface, display(Content, Setup), t4, t5)

exists t7 do(user, open(LimitMinders, Content), t5, t6) ---> 03
do(interface, display(Content, Limitdialogbox, t6,t7)

exists t9 do(user, type(RICcode), t7, t8) ---> 04
do(interface, display(RICcode, RICbox), t8, t9)

exists t11 do(user, select(BID, Fieldbox), t9, t10) ---> 05
do(interface, highlight(BID), t10, t11)
and do(interface, display(CurrentBIDprice, Currentvalbox, t10, t11)

exists t13 do(user, select(Lower, Limitsbuttons), t11, t12) ---> 06
do(interface, highlight(Lower), t12, t13)

continue.....

exists t24 do(user, apply(Exitbutton, Limitswindow), t22, t23) --->
do(interface, close(Content, Limitwindow), t23, t24)

3.4 Non-deterministic effects and 'safety' of interaction outcomes

One way to enable the modelling of user errors is to allow for non-deterministic effects in the model of the interaction process. A very simple way of providing for non-deterministic effects in abductive logic programming was suggested in [25]. The technique involves introducing an undefined predicate, for example *goodluck(E),* representing the piece of information missing. The effect of a non-deterministic action then depends on the actual truth value of this predicate, which is not fixed by the program (though it may follow from additional constraints). However, since the analysis of interface and interaction design should predict which design features are likely to contribute to usability problems and may lead to user error, this technique is insufficient. We would like to use the results of [10] and [14] to find out what sort of non-deterministic effects can occur during an interaction process. For example, the case that an "action was omitted" by interface agent may be modelled by adding additional conditions, for instance "nil":

exists t9 do(user, type(RICcode), t7, t8) ---> 04
do(interface, display(RICcode, RICbox), t8, t9)
or nil

In GLORIA the process of abduction is treated as deduction on the "only if" halves of the definitions of the predicates involved, realised through the "iff proof procedure" and declaration of *do* atoms as *abducibles*. The use of integrity constraints in GLORIA can be used to supports the modelling of interactions with the kind of user errors as suggested in [10], for example, by declaring some actions as 'safe' (following the formal approaches to reasoning about the behaviour of computer programs). Informally, a safety property stipulates that 'bad things' do not happen during execution of a sequence of actions. The opposite, a liveness property stipulates that "good things" do happen [27,28]. One example of safety property is that the mouse pointer must not disappear from view. Another is that the intended order of actions is not spoilt.

2.2 GLORIA and other Computer-based tools for interaction modelling

Several efforts have been made in recent years to develop computer-based tools for GOMS models. These include USAGE (**U**ser **I**nterface **D**esign Environment for **s**emi-**A**utomated **G**OMS **E**valuation) [6] and GLEAN (**G**OMS **L**anguage **E**valuation **A**nalysis) [7]. In the USAGE system, an application is constructed using the User Interface Design Environment (UIDE) by specifying a model of the interface and then "plugging in" the application-specific code. The interface model consists of application actions, interface actions, and interaction techniques, but not ouput information that the user perceives (and reacts to) at the interface. The GLEAN system is a computer-based tool that generates quantitative predictions from a supplied GOMS model and a set of benchmark tasks. It simulates the interaction between a simulated user who interacts

with a simulated device (the interface) to execute a set of specific benchmark tasks, following the method shown in Figure 1.
We are at the initial stages of applying GLORIA for interaction modelling, exploring its full potential in this regard. It is clear that significant enhancements to the GOMS-based analyses can be obtained. For example, in addition to the critical performance factor of time, it is possible to directly relate user actions to the design features that are intended to deal with feedback and error. GLORIA could be used at the earliest stages of the system design process, since paper-based interface designs can be relatively easily translated into plans of interaction and the results of the simulation used to check against the results of tasks analysis. There are further benefits that can be derived from this approach, namely that the results can be also used to design help materials, e.g. manuals, and for training.

6. Conclusions

The objective of this paper was to introduce a logic programming agent-based approach to the modelling and analysis of interaction and interface design. The architecture of the agent unifies planning and reactive behaviours and is well suited to the goal/subgoal decomposition and reaction to feedback behaviours that are part of interactions between a user and a computer system. The costs of interface design have been estimated at 80 per cent of the total development cost of an interactive human-computer system, and the persistence of usability problems makes a strong case for the development computer-based tools that can ease the design process and make it more effective. We have discussed the use of GOMS-based approaches to this problem, pointing out their strengths and limitations. We have introduced the agent GLORIA and have shown how it can be used to enhance and automate the analysis of interactions. This research is in initial stages. Our next step is to test GLORIA on increasingly complex interaction situations. The objective is to refine the representation formalism in particular with respect to the analysis of the role of feedback and user errors.

References

1. Card, S, Moran, T., Newell, A., "The psychology of human-computer interaction", Hillsdale, NJ: Erlbaum, 1983.
2. Kieras, D.E., "Towards a practical GOMS model methodology for user-interface design". In M. Helander (ed) Handbook of human-computer interaction, Amsterdam, North Holland Elsevier, 1988, pp 135-158.
3. John, B.E., "Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information". In Human Factors in Computer Systems, CHI'90, ACM, New York, pp 107-115.
4. John, B.E. and Kieras, D.E., "Using GOMS for User Interface Design and Evaluation: Which Technique ?", ACM Transactions on Computer-Human Interactions, Vol.3, No.4, pp 287-319, 1996.
5. John, B.E. and Kieras, D.E., "The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast". ACM Transactions on Computer-Human Interactions, Vol.3, No.4, pp 321-361, 1996.
6. Byrne, M.D., Wood, S.D, Sukaviriya P.N., Foley, J.D., Kieras, D.E., "Automating Interface Evaluation, Human Factors in Computing Systems, CHI'94, Boston, Massachusetts, April 24-28, 1994
7. Kieras, D.E., Wood, S.D., Abotel, K., and Hornof, A., GLEAN: A computer-based tool for rapid GOMS model usability evaluation of user interface designs. In Proceedings. ACM, New York, 1995.

8. Kitajima, M., and Polson, P.G., A comprehension-based model of exploration, Tech. Rept. 96-02, Institute of Cognitive Science, University of Colorado, 1996.
9. Neumann, P.R., Computer related risks. Addison Wesley Publishing Company, 1995
10. Rassmussen, J., Pederson, O.M., Mancini, G., Carnino, A., Griffon, M. and Gagnolet, P., Classification System for reporting events involving human malfunction. Rept. No. Riso-M-2240. Roskilde, Denmark: Riso, National Laboratory, 1981
11. Landauer, T., The trouble with computers. Usefulness, Usability and Productivity, The MIT Press, Cambridge, Massachusetts, 1995
12. Jeffries, R.J., "Usability problem reports: Helping evaluators communicate effectively with developers". In Nielsen, J., and Mack, R., (Eds) Usability Inspection Methods, John Wiley & Sons, New York, NY, 1994, 271-292.
13. Jeffries, R. J.Miller, J.R., Wharton C., and Uyeda, K.M., "User Interface Evaluation in the Real World: A Comparison of Four Techniques. Proc. ACM CHI'91 Conf., New Orleans, LA, April 28-3May, pp 119-124.
14. Nielsen, J., "Enhancing the Explanatory Power of Usability Heuristics", Human Factors if Computing, CH'94, Boston, Massachusetts, April 24-28, 1994, pp 153-158
15. Reuter Terminal, User Guide Version 3, Reuters, 1995, p20
16. Kowalski, R. and Sadri, F., An agent architecture that unifies rationality with reactivity. In Proc. of Logic and databases, Italy, 1996.
17. Da'vila, J., Agents in Logic Programming, PhD Thesis, Imperial College, April 1997
18. Kowalski, R and Sergot, M., " A Logic-based Calculus of Events", New Generation Computing, Vol.4, No.1, February 1986, pp.67-95.
19. Sadri, F., and Kowalski, R., "Variants of the Event Calculus", Proceedings of the International Conference on Logic Programming., MIT Press, 1995.
20. Eshgi, K., "Abductive planning with event calculus", Proceedings of the International Conference on Logic Programming., MIT Press, 1988.
21. Kowalski, R., "Database Updates in the Event Calculus", Journal of Logic Programming, Vol12, No.162, 1992, pp. 121-146.
22. Sadri, F., Three recent approaches to temporal reasoning, in "Temporal logics and their applications", Galton, A. [ed], Academic Press, 1987, 121-168.
23. Shanahan M.P., "Prediction is deduction but explanation is abduction", Proceedings of International Joint Conference on Artificial Intelligence, 1989.
24. Denecker M., Missiaen, L., Bruynooghe M., "Temporal reasoning with abductive event calculus", Proceedings of ECAI, 1992.
25. Sripada, SM., "Efficient Implementations of the Event Calculus for Temporal Database Applications", In,...MIT 1995, pp99-113
26. Van Belleghem, K., Denecker, M., De Schreye, D, "Combining Situation Calculus and Event Calculus, In, ...MIT 1995, pp83-97
27. Alpern, B., and Schneider, F.B., "Recognising Safety and Liveness. Distributed Computing, Vol 2, 1987, pp 117-126
28. Zave, P. and Jackson, M., Four Dark Corners of Requirement Engineering, ACM Transactions on Software Engineering and Methodology, Vol 6, No.1, January 1997, pp 1-25