International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:2, 2008

Modeling User Behaviour by Planning

Alfredo Milani, and Silvia Suriani

***Abstract*—**A model of user behaviour based automated planning is introduced in this work. The behaviour of users of web interactive systems can be described in term of a planning domain encapsulating the timed actions patterns representing the intended user profile. The user behaviour recognition is then posed as a planning problem where the goal is to parse a given sequence of user logs of the observed activities while reaching a final state.

A general technique for transforming a timed finite state automata description of the behaviour into a numerical parameter planning model is introduced.

Experimental results show that the performance of a planning based behaviour model is effective and scalable for real world applications. A major advantage of the planning based approach is to represent in a single automated reasoning framework problems of plan recognitions, plan synthesis and plan optimisation.

***Keywords*—**User behaviour, Timed Transition Automata, Automated Planning.

I. Introduction

IN most current research works about user behaviour modelling, several approaches focus on formalizing *user session* ([1][6][8]) and *user behaviour* based on developing measures[8], analysing user-action histories[7] or navigation histories [1].

A main drawback of all these models is that they allow to focus and reason on a single aspect of user activity, usually patterns of actions where automata based models [4].

A planning [11,12] based approach to user behaviour modeling would allow to introduce and model more general issues such as goal directed behaviour (behaviours which are characterised by the attainment of specific goals), optimisization directed behaviour (behaviour which minimise/maximise some given cost function), and it also allow to better analyse and support the user activity, i.e. anticipatory and collaborative planning.

One of the major barriers to real world application of planning systems has been the closed world assumption (CWA), i.e. all the state changes affecting the world state can only be due to domain actions. On the other hand virtual environments, such as e-learning platforms and e-commerce environments, seem to be less sensitive to the limits of CWA because the interaction occurring in an artificial environment can be easily and completely modelled.

In the following paragraph we will analise automata based models for user behaviour and we will show how hey can be modeled in the framework of numerical parameters planning model, where more general planning and optimisation problems can be posed. Experimental results both for behaviour recognition and general planning problems are also discussed.

Manuscript received November 9, 2007. This work was partially supported by the under Fondazione Cassa di Risparmio di Perugia E-studium Project Grant.

A. Milani is with the Department of Mathematics and Computer Science, University of Perugia, Perugia, 06100, Italy (phone: +39-075-585.5049; fax: +39-075-585.5023; e-mail: milani@unipg.it).

S. Suriani is with the Department of Mathematics and Computer Science, University of Perugia, Perugia, 06100, Italy (e-mail: suriani@dipmatg.unipg.it).

II. Timed Automata for User Behaviour

The domain of the actions available to a user operating in a structured interface environment (e.g. e-learning platforms, webmail clients, content management platforms) can be easily described by a state transaction diagram extended with time constraints. Each action which can be performed by the user is represented by a state transition label.

A *Timed Transition Automata* (TTA) [1] is a finite state machine which is able to recognise timed words, i.e. a sequence of pairs made by symbols over a given alphabet Σ and time values. The pairs in the sequence can be seen as a sequence of logs records, describing user events or actions annotated with the time in which they occurred.

In a TTA it is possible to constrain a certain action to be executed, i.e. a certain transition to occur, only when some time conditions are met (e.g. submitting an online assignment within a given interval of time). A *domain automata* can then be defined for representing the legal transitions or, equivalently, the legal sequences of actions which can occurs in the system.

A *user behaviour* can be described by a timed automata whose accepted language is a sublanguage of the domain automata. In other words if the sequence of user actions is parsed by the automata the corresponding user behaviour is recognized. The use of TTA for describing user behaviour has been proposed in [4].Please submit your manuscript electronically for review as e-mail attachments.

A. A Sample E-learning Platform

In the following example we describe a sample e-learning platform that can be used for the recognition of user behaviours. The e-learning platform that we consider allows the user to perform 7 main operations or activities: *login, lesson, quiz, assignment, chat, view, logout,* and some additional operations: *main menu* which allows to abandon an activity and go back to the main menu; *submit/abandon* which

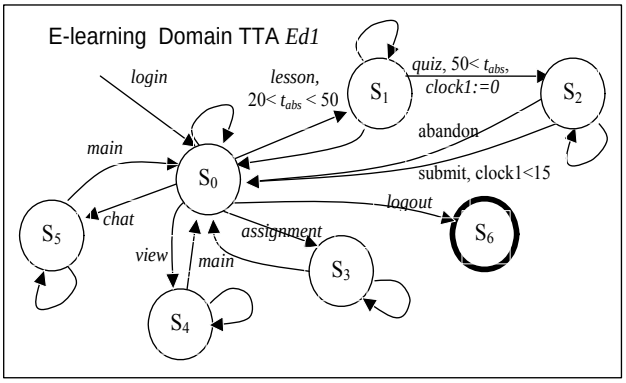

Fig.1 E-learning platform TTA model *Ed1*

respectively allow to submit the answers of a quiz, or abandoning it without answering. The activities are not all available at the same time, but they are subject to time and precedence constraints. For example, the *lesson* activity allows to attend an on-line lesson which begins at a given starting time (absolute time t_{abs}>20) and it last 30 minutes (t_{abs}<50); the *quiz* activity is enabled only if the user has attended the lesson and it is allowed to submit the answer within 15 minutes. The *login/logout* actions allow to access/exit the platform and they have no temporal constraints. The *assignment*, *chat* and *view* activities also have no temporal constraints, but they can only be accessed from the main menu.

This e-learning domain is represented by the following TTA where the activities/operations are represented as arc labels. In each state the dashed loops indicate the idle action, i.e. the action of remaining in the current state.

Two possible user behavior models on the e-learning domain can be represented by others TTAs as follows:

User Model *ub1*. In model *ub1* the user, after entering the e-learning platform (*login* action), can repeat the *assignment* activity many times, but, in order to reach the final state S5, he has to attend the *lesson* until the end for at least 25 minutes (clock2 > 25) and after that he has to *submit* the answer to the quiz.

User Model *ub2*. Behaviour model *ub2*, instead, describes a user which chooses *view* for at least 15 minutes as first activity, and then he/she can alternate *view/chat* without temporal constraints before *logout*.

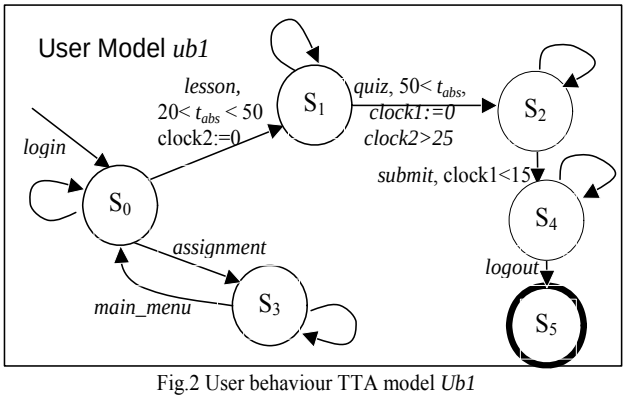

Fig.2 User behaviour TTA model *Ub1*

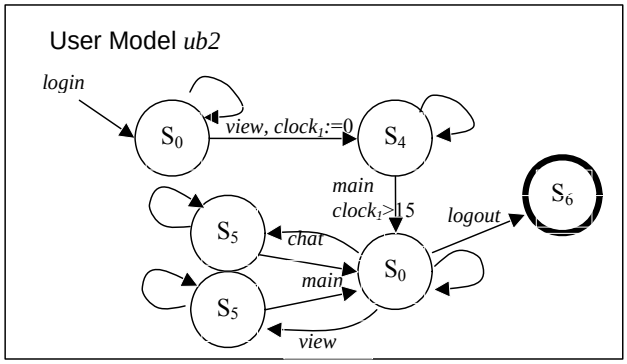

Fig.3 User behaviour TTA model *Ub2*

A time automata recognises a user behaviour, if it parses the timed word associated with a use history, i.e. the timed sequence of user logs.

Let consider, for example, the following user action log sequences, where each log record consists of a pair *time stamps* and *action.*

```
Seq1:[(0,login),(10,assignment),(12,main),(2
2,lesson),(23,main),(24,lesson),(51,quiz),(6
5,submit),(70,logout)]
Seq2 : [(0,login), (3,view), (19,main),
(20,chat) (25,main), (29,chat), (35,main),
(37,view), (40,main), (41,logout)]
Seq3: [(0,logon, (5,view), (30,main),
(32,logout)]
Seq4: [(0,login), (40,lesson), (51,quiz),
(55,submit), (57,logout)]
Seq5 : [(0,login), (5,view), (30,main),
(31,assignment), (41,main) (42,logout)]
```

It is easy to see that sequence *Seq1* is an example of user behaviour which is recognised by TTA *ub1*, sequence *Seq2* and *Seq3* are recognized by *ub2*, while *Seq3* and *Seq5*, although a legal sequence in the sample platform domain *Ed,1* are not accepted behaviours. Seq4 violates the constraint about lesson attendance of *ub1* and it has actions incompatible with *ub2*, while *Seq5* contains activities of type *view* and *assignment* which are either incompatible with *ub1* or with *ub2*.

B. Timed Transition Automata (TTA)

Let us recall more formally some basic concepts related to *Timed Transition Automata*.

Definition (Timed word). Given a finite alphabet Σ, a timed word on ∑, is a finite sequence of pairs $[(a_0,\tau_0) \ldots (a_k,\tau_k)]$ where $a_i \in \Sigma^*$, $\tau_i \in \Re$ for $i \in [0,k]$ with $\tau_i \le \tau_{i+1}$ $i \in [0,k-1]$

Definition (Timed Language). A *timed language* over an alphabet ∑ is a subset of timed words on ∑.

Definition (Time Transition Automata). A *Timed Transition Automata (TTA)* is a tuple
(∑, S, S_0, C, E, F) where

- ∑ is finite alphabet,

- S is a set finite of state,
- $s_0 \in S$ is an initial state,
- C is a finite set of clocks,
- $F \subseteq S$ is a set of final acceptance states,
- $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ defines the transition table for the automata, each transition $e \in E$ is a 5-ple $e=<s,s',a,\Lambda,\delta>$ representing a transition from state s to state s' on input symbol a which can occur at a certain time τ when clock constraint δ is verified by the current values of clocks, the transition also resets to 0 the subset $\Lambda \subseteq C$ of clocks.

Clocks are used to express more easily time constraints such as durations relative to sub patterns in the transition diagram. Clocks are usually initialised to 0 and they are updated as time advances.

Given a set X of clocks, the set of clock constraints $\Phi(X)$ includes all the simple constraints conjunctions and negations defined by $\delta := x \le c \mid c \le x \mid \neg\delta \mid \delta \wedge \delta$ where $x \in X$ is a clock and c is a rational constant.

Definition (Run of Timed Transition Automata). A *run* of a timed transition automata, record a sequence of legal state transitions and the value of all the clocks when state transitions take place, starting from an initial state $s \in S_0$. It is easy to see that a timed word can correspond to a consistent run when a transition occurs at time alphabet symbol a_0 described transition occurring which verified clock constraint.

Definition (Timed Language). The language L(A) accepted by an automata $A=(\Sigma, S, S_0, C, E, F)$ is the set of all timed words which correspond to consistent runs of the automata starting with the state s_0 and ending with a final state $s_f \in F$, i.e. a timed word $w=[(a_i,\tau_i)]$ with $i \in [0,k]$ is also $w \in L(A)$ if exists a run from an s_0 with each transition $<s,s',a_i,\lambda,\delta>$ taking place at time instant τ_i and the final transition being $<s_{f-1},s_f,a_k,\lambda,\delta>$ for a state $s_f \in F$.

III. Numerical Parameters Planning Model

In the following we recall some basic notions about the numerical parameters planning model which is used to implement the TTA recognition process.

The plan synthesis problem consists in finding a sequence of domain actions which, if executed, transform a given initial state in a goals state. Planning systems have been widely used to model domain where one or more deliberative actors can modify the state of the world executing a set of predefined available actions.

The planning model used extends the classical Boolean planning models with the management of numerical resources and goals, moreover effects can depend on numerical continuous parameters of the action instance.

The semantics of the model is based on three finite sets: *B*, *N*, and *P*, which respectively represent *logical fluents, numerical fluents* and *numerical parameters*. Numerical fluents and numerical parameters are defined in bounded real interval domains.

Definition (State). A *state* is a pair of assignments $s=(s_B, s_N)$ where s_B: $B \rightarrow$ {true, false} assigns truth values to logical fluents, and s_N: $N \rightarrow \Re$ assigns real values to numerical fluents. S_B denotes the set of all possible logical assignments and S_N the set of all possible numerical assignments; finally S denotes the set of all possible states.

Definition (Operators). An operator is defined by a triple $o=(X,\pi,\varepsilon)$ where:

- $X \subset P$ are the numerical parameters of o;
- π are the preconditions of o;
- ε are the effects of o.

Preconditions π are conjunctions of *literals* (i.e. b or $\neg b$, where $b \in B$ is a logical fluent) and *numerical constraints* of the form $f_{N \cup X} \otimes 0$, where f is a linear function of numerical fluents/parameters and $\otimes \in \{ <, \le, =, \neq, \ge, > \}$. Effects ε are conjunctions of *literals* and *numerical effects* (i.e. assignments of numerical fluents of the form $u := g_{N \cup X}$ where $u \in N$, g is a linear function of numerical fluents/parameters). Let O denote the set of all operators.

Definition.(Action Instance). An action instance is defined by a pair (o, σ) where $o=(X,\pi,\varepsilon)$ is an operator and σ a parameter assignment σ: $X \rightarrow \Re$. Action instance (o,σ) is said to be executable in a state $s=(s_B,s_N)$ if logical and numerical conditions hold in s and numerical effects are consistent with the domain bounds.

Definition.(Action Execution). If an action instance (o,σ) is executable in a state $s=(s_B,s_N)$, the result of its execution is a state
$s'=(s'_B,s'_N)$, where
for each logical fluent $b \in B$

$s'_B(b) = \text{true}$ if $b \in \varepsilon$
$s'_B(b) = \text{false}$ if $\neg b \in \varepsilon$
$s'_B(b) = s_B(b)$ otherwise

for each numerical fluent $u \in N$

$s'_N(u) = g_{N \cup X}$ if $u := g_{N \cup X} \in \varepsilon$
$s'_N(u) = s_N(u)$ otherwise

s' can be also denoted by $\gamma(s, (o,\sigma))$.

Definition (Numerical Parameterized Planning Problem). A numerical parameterized planning problem is a tuple $\Sigma = (B, N, P, S, O, s_0, G)$ where B, N, P, S, O represent boolean fluents, numerical fluents, numerical parameters, states and operators, and
- $s_0 = (s^0_B, s^0_N)$ is the initial state;
- G is a conjunction of literals and numerical constraints defined over $B \cup N$ representing the goal.

Note that goals are defined over (B,N), i.e. goals cannot contain any parameter symbols.

Definition (Solution Plan). A plan, i.e. a sequence of action instances $((o_0,\sigma_0) \ldots, (o_k,\sigma_k))$, is a solution plan for a planning problem $\Sigma = (B, N, P, S, O, s_0, G)$ if the sequence is executable and the goal G holds in the final state.

The sequence of actions is executable when (o_0,σ_0) is executable in s_0 and each action instance (o_i,σ_i) is executable in $s_i = \gamma(s_{i-1}, (o_{i-1},\sigma_{i-1}))$ for each i = 1,…,k.

The goal G holds in the final state $s_{k+1} = \gamma(s_k, (o_k,\sigma_k))$, if $\forall g \in G$ when g is a literal g=b (g=¬b) then $s^k_B(b)$=true (s^k_B (b)=false), or when g is a numerical constraint $f_{N\cup X}$ then $f_{N\cup X} \otimes 0$ holds in s_{k+1}.

The Numerical Parameter Planning model has been implemented using a technique of mixed integer linear programming (MIP) encodings [13]. The algorithm follows the approach firstly proposed in Blackbox [12] and then developed by many others system [14,15,16]. A planning graph is built with logical fluents and operators ignoring the numerical aspects of the problem, then, the planning graph is encoded as a MIP extended to handle numerical fluents and parameterized actions. A standard MIP solver is then used to solve the planning problem [ILOG CPLEX].

IV. User Behaviour Recognition as a Planning Problem

Since automated planning models encode state transitions, the basic idea of our approach has been to use action to encode TTA state transition. The TTA representing a behaviour can be embedded by an appropriate planning domain, where each planning action corresponds to parse a user action in the TTA model, (i.e. corresponds to a legal TTA transition) and user histories are represented as the initial state of a given planning problem.

The current state of TTA is simulated by asserting/negating appropriate fluents. Each planning action representing a TTA transition $<s,s',a_i,\lambda,\delta>$ is executable only if the current simulated state is "s", and if a log record for "a_i" exists whose time stamp verifies the time constraints δ.

The planner can then be used to verify if the history corresponds to a path from the initial TTA state to a final TTA state.

A. The Planning Domain Problem

Given the TTA (∑, S, S_0, C, E, F) representing a user model, and given a sequence of logs *Log*, it is possible to define a planning domain problem (B, N, P, O, s_0, G) for user model recognition problem, where:

- B={ *curr_state(s_i), final(s_i), success, curr_log(l_i), next(l_i, l_j), log(1_1,a,t,d)*} is the set of logical fluents where s_i and l_i refer to TTA states and *Log*;
- N={ t_{abs} , $t_{\lambda i}$} is the set of numerical fluents,
- P={ t_e } is the set of numerical parameters,
- O={$A_{<s,s',a,\lambda,\delta>}$, $A_{<s,s,a,\lambda,\delta>}$, $Idle_S$, A_f } $\forall s \in S$, $\forall f \in F$, $\forall_{<s,s',a,\lambda,\delta>} \in E$ is the set of the operators
- G= $G_B \cup G_\delta$ with G_B={*success*} is the set of literals defined over B and G_δ ={} is the set of numerical constraints defined over N.

Each log consists of a 4-pla *log(1,a,t,d)* where *l* is a log sequential identifier, *a* is the performed action, *t* is the time stamp of the starting time, and *d* is the time interval between the action and the next one.

B. Fluents and TTA States

Given a TTA (∑, S, S_0, C, E, F) some *logical* and *numerical fluents* are introduced to represent states, logs, current state, current log and logs sequence.

1) Logical Fluents

$\forall$ *s*∈S, *curr_state(s)* logical fluent is defined in order to represent the current state, note that these fluents are used to represent the situation in which the TTA is currently in the state s_i , the domain actions must guarantee that at most one *curr_state(s)* can be true at the same time.

$\forall$ *log(1,a,t,d)*∈*Log* a fluent *log(1,a,t,d)* is introduced.

$\forall$ *log(1,a,t,d)*∈*Log,* a fluent *curr_log(l)*, is a logical fluent defined in order to represent the current log, similarly to *curr_state(s)* only one *curr_log(l)* can be true at the same time . The sequential order of the logs is represented by the fluents *next(l_i,l_j),* where l_i is the successor of l_j in the sequence, a special fluent *curr_log(init)* represents the initial situation when no log are have been parsed yet, conversely a special fluent *curr_log(end)* is used to mark the end of the log sequence; moreover two fluents *next(init,l_1)* and *next(l_k, end)* are also added accordingly.

A set of fluents *final$_{Si}$,* for each final state $s_i \in F$ and a single logical fluent s*uccess* are also used to specify disjunctive goals.

2) Numerical Fluents

A numerical fluent t_{abs} is defined to represent the *absolute time* as it evolves while actions are executed.

A numerical fluent t_c is also introduced for each *clock c*∈*C*.

3) Initial State

The *initial state* of the planning problem represents the initial state of the timed automata and the value of the clocks and of the absolute time are initially set to 0.

curr_state(s_0)= T
curr_state(s_i) = ⊥ $\forall s_i \in S$, i≠0 s_i is false in I
t_{abs} = 0, t_d =0 $\forall d \in \delta$

moreover it is also needed to represent the initial state of the parsing process:

curr_log(init) = T
curr_log(end) = ⊥
curr_log(l_i) = ⊥ $\forall l_i \in Log$
final$_{Si}$ = T $\forall s_i \in F$
success = ⊥

the latter two are needed to indicate which are the final states and the fact that the parsing is not yet successful.

C. TTA Transitions and User Logs

Appropriate actions $A_{<s,s',a,\lambda,\delta>}$, $A_{<s,s,a,\lambda,\delta>}$, *Idle$_{Si}$* and A_{Si} are introduced in the planning domain in order to represent respectively transitions, self-referencing transitions, idle states and the final disjunctive goal.

1) Transitions and Self-referencing Transitions

For each transition *e*∈*E*, e=<s,s',a, Λ ,δ> of the automata corresponding to the system log 1_1 at time *t*, where *s*≠*s'*, a planning operator denoted by $A_{<s,s',a,\lambda,\delta>}$ or equivalently by A_e is introduced as follows

Pre(A_e)={ curr_state(s) ∧ curr_log(1_1)
∧ next(1_1, 1_2)

$\wedge$ *log*(l_1, *a*, *t*, *d*)
$\wedge$ δ $\wedge$ $t_{abs}=t$ }
NumPar(A_e)={}
Eff(A_e)={ $\neg$ *curr_state(s)* $\wedge$ $\neg$ *curr_log*(l_1)
$\wedge$ *curr_state(s')* $\wedge$ *curr_log*(l_2)
$\wedge$ (t_λ:=0, $\forall\lambda \in \Lambda$)
$\wedge$ (t_λ:= t_λ+*d*, $\forall$s.t. $\lambda\in$C and $\lambda\notin \Lambda$)
$\wedge$ (t_{abs} := t_{abs}+*d*)}
where *d* is the duration of the action as defined in the history sequence of user activities.

The time constraints δ are numerical constraints on the numerical fluents corresponding to the clocks and/or the absolute time, the constraint $t_{abs}=t$ enforces the requirement that the transition in TTA take place at the time *t* specified by the log.

Also note that the current state and the current log are updated accordingly to the state transition table and to the logs order, while absolute time is updated with action duration *d* and clocks are either updated or reset to 0.

A special case is when a transition specifies the same starting and target state, i.e. the corresponding node in the automata graph contains a self reference loop. For each transition

$e\in E$ of type $e=<s, s, a, \Lambda, \delta>$ it is introduced an action $A_{<s,s,,a,\Lambda,\delta>}$ whose definition differs from the previous one only in the effects, i.e. the negation of current state and the update to the new state, $\neg$ *curr_state(s)* $\wedge$ *curr_state(s)* , are omitted from the action effects since they would lead to inconsistency.

Note that the execution of an action of type $A_{<s,s',a,\Lambda,\delta>}$ or $A_{<s,s,a,\Lambda,\delta>}$ corresponds to parse a log record as required by the precondition *log(l,a,t,d)*.

Parsing starts from the only one action executable in the initial state, where *curr_state(init)* is true , and it follows the order encoded by the *next* predicates.

2) Idling state

If the TTA model admits idling in a state, i.e. remaining in a state while performing no action, then a special *idle operator* $Idle_{Si}$ is added for each state $s_i\in$S of the TTA in order to model the time flow. The possibility of being idle allows to have gaps in the logs temporal sequence. The idle operators have a quite simple structure since in order to be executed, they do not require either logs to exists, or time/clock constraints to be verified. On the other hand idle operators contain an additional *numerical parameter* t_e which represents the elapsed time

Pre($Idle_{Si}$)={ *cur_state*(s_i) }
NumPar($Idle_{Si}$)={ t_e }
Eff($Idle_{Si}$)={ (t_{abs} := t_{abs}+ t_e)
$\wedge$ ((t_λ:= t_λ+ t_e, $\forall\lambda\in$C) }

Note that the numerical parameter t_e represents the idling interval and it is used to update the absolute time as well as all the clocks. Numerical parameters are values which are chosen by the planner in order to instantiate the action instance.

D. TTA Final States & Planning Goals

The TTA recognizes a timed word when it reaches one of the possible final states after parsing all the logs. In order to model these conditions it is necessary to specify in the planning disjunctive goals like

$$curr\_log(end) \wedge (\bigvee curr\_state(s_i)\ \forall\ s_i \in F) \ ,$$

In other words we want to know, if it is possible that the goal state is either one of the final states *curr_state*(s_i) when the logs are ended, i.e. when *curr_log(end)*.

Since we assume a conjunctive planner we can specify disjunctive goals using a well known technique [9] which requires to introduce a set of dummy actions representing the disjunctive goal.

For each final state, $\forall s_i\in$F, a dummy operator A_{Si} is added to the set of domain operator O such that:

Pre(A_{Si})≡{*curr_state*(s_i), *final*(s_i), *curr_log(end)*},
Eff(A_{Si})≡{ *success* }
where *success* is a logical fluent representing the end of the user behaviour recognition process.

The fluent *success* is true in a state when at least one of the possible action A_{Si} with $s_i\in F$ has been executed, i.e. a final state has been reached (see preconditions *cur_state*(s_i), *final*(s_i)) when parsing the last log (precondition *curr_log(end*)).

V. Experiments

The TTA to planning rules described in the previous paragraph show that the transformation space complexity is linear in the size of the planning domain. On the other hand is not possible to provide a theoretical estimate for plan synthesis time, since it strongly depends on the planner implementation which can employ very efficient strategy expecially for the logical fluents. In order to obtain a general estimate of the effectiveness of the approach we have held systema*tic* experimental tests using PNP (Parametric Numerical Planner), the tests are based on *ub1, ub2* ans *el1* domains.

PNP has been implemented in C language and performs the graph construction phase and the encoding phase, while the solution of the MILP system is performed by using ILOG CPLEX. The test has been executed on Intel Pentium IV 3.00GHz with 1GB of RAM running the operating system Linux.

The tests have been divided into three classes: positive and negative cases for user behaviour recognition, and planning problems in elearning domain. In particular negative cases has been tested for different causes of recognition failure: a) logical failure i.e. action sequences not allowed by the TTA describing the user behaviour and b)numerical failures, action time stamps which violates the numerical time constraint of the TTA. The scalability of the approach has been tested with different users histories, i.e. log sequences of increasing length.

Finally a planning domain corresponding the TTA *el1* has been modelled to show the flexibility and expressivity, since the problem does not require to parse any log, the fluents of

type *log*, *curr_log*,and *next* have been removed from the action descriptions, dummy actions and goals.

TABLE I
POSITIVE RECOGNITION TEST FOR *UB1* AND *UB2* DOMAINS

Logs	*Ub1*			*Ub2*		
	Time	Nodes	Var	Time	Nodes	Var
5	0,03	32	174	0,03	34	165
9	0,04	47	345	0,04	49	364
21	0,1	83	1083	0,09	85	1234
33	0,28	119	2253	0,2	121	2536
41	0,49	143	3273	0,34	145	3644
53	1,09	179	5163	0,63	181	5666
61	1,92	203	6663	1,01	205	7254
73	3,56	239	9273	1,74	241	9996
81	5,03	263	11253	2,26	265	12064
93	6,98	299	14583	3,49	301	15526
101	8,23	322	16711	4,55	325	18074

In figure the execution time for positive cases of *ub1* and *ub2* are shown for increasing log sequence length from 5 to 101 logs per session, the increment steps of size *4* is due to the particular form of legal log sequences for *ub1* and *ub2*, the table also shows the size for plangraph nodes and variables of the linear programming system.

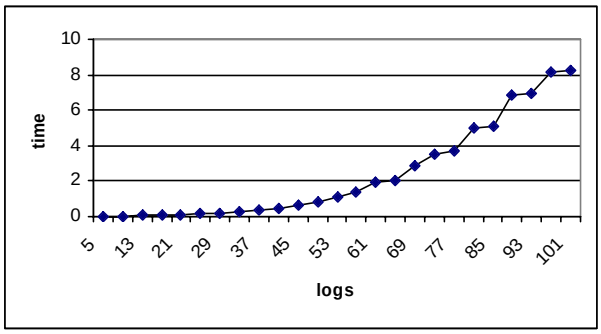

Fig.4 Positive Recognition Tests time for domain *Ub1*

The results obtained are completely satisfactory for the three classes of tests. In particular positive user behaviour recognition is quite efficient to be used in real time applications, since the top sequence size of 101 log records are fairly more than the typical user sessions, which consist of less than ten logs, the time performance for twenty logs is worst case not greater than 0.1 seconds. Negative tests on user behaviour recognition were even more efficient than positive tests, in particular it must be noted that negative test of type a), i.e. where the sequence violates a logical constraint can be detected very efficently in the early plangraph construction phase, and the error detection time is proportional to the length of the correct prefix. Negative tests of type b), i.e. where the timed logs violate the numerical constraints, require the execution of both phases of plangraph construction and LP solving, these tests show an execution time which is slightly minor than the correspondent positive test.

TABLE II
NEGATIVE RECOGNITION TEST
FOR *UB1* AND *UB2* DOMAINS

Logs	*Ub1*		*Ub1*	
	Log	*Num*	*Log*	*Num*
5	0,03	0,06	0,03	0,09
9	0,03	0,06	0,03	0,09
21	0,03	0,07	0,04	0,11
33	0,06	0,12	0,06	0,18
41	0,09	0,19	0,10	0,29
53	0,18	0,38	0,20	0,58
61	0,26	0,54	0,28	0,82
73	0,47	1,00	0,53	1,53
81	0,68	1,43	0,75	2,18
93	1,11	2,30	1,19	3,49
101	1,50	3,09	1,59	4,68

The last class of tests, i.e. general planning problems based on *ed1*, is not plotted since the time results are all extremely fast, always below 0.04 seconds for all the posed problems. It must be noted that the problems which can be defined in this framework, belong to the class of reachability within a given timeline, or optimal reachabilty, i.e. to built a time plan to reach a given state with a possibly optimal cost metric. It would be interesting to investigate in a future extension a task planning approach similar to [10] where task goals and logical goals can be mixed.

It should be noted that the PNP planner used in the experiments is a general purpose one, on the other hand a more efficient search strategy based on forward search can be developed for log sequence regognition. Special purpose planners could also exploit the fact that the recognition plan length correspond to the log sequence length plus one extra dummy action. Moreover incremental strategies can be developed for real time application in order to support online user behaviour recognition, i.e. anticipating the log parsing process before the session ends and before all log records are avaible.

VI. CONCLUSION

A planning approach to user behaviour recognition has been introduced. The available actions occurring in a web platform domain (such as e-learning, webmail and e-commerce platforms), and user behaviours can be easily described by *Timed Transition Automata* (TTA) i.e. state transaction diagrams extended with time constraints. The main idea of the proposed approach is to built a planning domain model to encode the state transitions of TTA representing behaviours, where each planning action corresponds to parse a user action, i.e. corresponds to a legal TTA transition, and user histories are represented as the initial state of a given planning problem. The behaviour recognition problem is then transformed into the planning problem of finding a parsing plan for the sequence of user logs. The formal TTA to plan transformation is proved to be correct, and it is built in the

framework of a numerical parameters planning model, which extends the classical boolean planning models with the management of numerical resources and goals and effects can depend on numerical continuous parameters of the action instance.

One of the relevant advantage in using a planning approach to user task modeling is that user behaviour recognition and user plan optimisation problems can be modeled in a unique framework.

Systematic experiments with PNP, a general purposes parametric numerical planner implementation, show that the approach if effective and scalable for user behaviour detection as well as for goal based plan synthesis and optimisation problems.

Future work will regard the development of special purpose plan search techniques targeted on the logs parsing problem, where forward search techniques seems to be a promising extension. Incremental techniques i.e. plan construction techniques which do not require the whole log sequence available in advance, will be also explored.

Another line of research, which is worth to investigate, consists in estending the proposed model with task constraints [8] in order to directly build a planning based model of the user behaviour without using an intermediate TTA model, and to integrate goals oriented with task oriented behaviour models.

REFERENCES

[1] Alur R., Dill D., "A theory of timed automata", *Theoretical Computer Science*, vol. 126, num. 2, p. 183–235, 1994.

[2] B. Berendt, M. Spiliopoulou, "Analysis of navigation behaviour in web sites integrating multiple information systems", *The VLDB Journal*, 9, Springer-Verlag, 2000, pp. 56–75.

[3] B. Berendt, G. Stumme, A. Hotho, "Usage mining for and on the Semantic Web", In: H. Kargupta, A. Joshi, K. Sivakumar, & Y. Yesha (Eds.), *Data Mining: Next Generation Challenges and Future Directions*, AAAI/MIT Press, Menlo Park, CA, 2004, pp. 461-480.

[4] S. Ceri, F. Daniel, V. Demaldé, F. M. Facca, "An Approach to User-Behavior-Aware Web Applications", ICWE 5 Proceedings, Sydney, Australia, Springer, 2005.

[5] R. Cooley, B. Mobasher, J. Srivastava, "Data preparation for mining world wide web browsing patterns", *Journal of Knowledge and Information Systems*, 1(1), 1999.

[6] F. Masseglia, P. Poncelet, M. Teisseire, A. Marascu, "Web Usage Mining: Extracting Unexpected Periods from Web Logs", *TDM 2 - ICDM'05 Proceedings*, Houston, USA, 2005.

[7] M. Mühlenbrock, "Automatic Action Analysis in an Interactive Learning Environment", *AIED-2005 Proceedings*, Amsterdam, NL, pp. 73-80.

[8] M. Teltzrow, B. Berendt, "Web-Usage-Based Success Metrics for Multi-Channel Businesses", *WebKDD 2003 9th ACM SIGKDD Proceedings*, Washington DC, USA, 2003.

[9] Marco Baioletti, Stefano Marcugini, Alfredo Milani: Encoding Planning Constraints into Partial Order Planners. *KR98 Proceeding, 6^{t} Int. Conf. on Principles of Knowledge Representation and Reasoning*, pp.608-616, Morgan Kauffmann 1998, ISBN 1-55860-554-1.

[10] Marco Baioletti Stefano Marcugini, Alfredo Milani: Task Planning and Partial Order Planning: A Domain Transformation Approach. in *Lecture Notes in Computer Science, Vol.1348*, pp.52-63, Springer-Verlag, Berlin, Germany, 1997, ISBN 3-540-64912-8.

[11] Blum, A., and Frust, M. Fast planning graph analysis. *Artificial Intelligence 90*, 1-2 (1997), 279-298.

[12] Kautz, H., and Selman, B. Planning as satisfiability. In *10th European Confernce on Artificial Intelligence (ECAI)* (1992), B. Neumann, Ed., Wiley & Sons, pp. 360-363.

[13] Kautz, H., and Selman, B. BLACKBOX: A new approach to the application of thorem proving to ploblem solving. In *working notes of the AIPS-98 Workshop on Planning as Combinatorial Search* (1998), pp. 58-60.

[14] Suriani, S. Numerical Parameters in Automated Planning. *PhD Thesis - Department of Mathematics and Computer Science – University of Perugia.*

[15] Wolfaman, S., and Weld, D. The LPSAT engine and its application to resource planning. In *Proc. of IJCAI-99* (!999).

[16] Vossen, T., Ball, M. Lotem, A. and Nau, D. Applying integer programming to AI planning, *Knowledge Engineering Review* 16:85–100, 2001.

[17] Van de Briel, M. and Kambhampati, S. Optiplan: Unifying ip-based and graph-based planning. *Journal of Artificial Intelligence Research 24* (2005), 919-931.