AN EVENT BASED TASK MODEL FOR INTERACTIVE BUSINESS PROCESSES

Parameswaran, Nandan
University of New South Wales, Sydney, Australia

Chakrapani, Pani N.
University of Redlands, Redlands, USA

ABSTRACT

Achieving tasks successfully in dynamic environments needs attributes of task structures that are tolerant to changes in a business environment. In this paper, we have proposed an event based model for interactive tasks where the user interactions are represented in terms of agent intentions in the model. Temporal and other constraints and relations between events are expressed using event models. When constraints are violated, we have proposed strategies for managing deviations. Application of our task model to a simple shopping scenario shows that the model performs favourably in real world.

INTRODUCTION

A task in a business application typically consists of several activities which are represented using actions that agents will execute in order to achieve one or more goals in the task. When all actions are successfully executed, the task is said to have been achieved. Achieving a task in changing environments is often challenging due to changes in states of resources and in the mental states of the agents that were initially committed to execute the task. For example, when an action is executed, there may be other events that unexpectedly follow the execution of the action, and these events may or may not take the resources to a desired state. In such situations, often agents may need to interact with the task structure and make changes in them so that the execution of the task can be continued in the changed situations. In this paper, we propose a task model that is suitable for execution in interactive applications.

In Section 2 on Task Model for Interactive Application, we present a model of task using events, and show how complex tasks can be represented using event abstractions and relations where we define concepts of the relevant objects, relations, and temporal entities, and the associated constraints. In Section 3, we show an application of our model to grocery shopping. Section 4

briefly describes an initial version of an implementation of grocery shopping. In Section 5 we review the related work and Section 6 is conclusion.

EVENT BASED TASK MODEL

A world consists of objects where an object changes its state at time t when an event occurs at t on a timeline T. Each object has its own timeline and events occurring on one object's timeline may affect events occurring on another object's timeline. A task is modelled using events where the events are related to each other by relations that are temporal, spatial, and agent attitudinal. Figure 1 shows a simple task structure consisting of events *ei* and state *si* occurring on a timeline T. An event is denoted as a transition from one state to another state in time. Thus, e1:s0 → s1 is an event where s0 and s1 denote the states of the objects in the world. Events typically occur at a point in a time line. We distinguish four types of events:

(i) α type events these events occur due to the execution of an action ai by an agent;

(ii) λ type events these events, denoted by la, occur due to an external agency;

(iii) μ type events these events, denoted ui, occur due to internal causes in the world resources; and

(iv) ε type event this denotes a null event written simply as ε.

When we do not care to distinguish between events, we use ei to denote any of them. In Figure 1, event e1 occurs at time t1 causing the state s1 which lasts until time t2 at which point another event e2 occurs causing state s2. At time t3, the diagram shows a choice of four events: (i) la:s2 → s3, (ii) a1: s2 → s4, (iii) E: s2 → s2, (iv) a2: s2 → s5 || a3: s2 → s6, where || signifies the fact that the actions a2 and a3 occur parallelly. At t3, any one of the four events can occur. The state s6 is the result of executing the action a3 at time t3, and at time t4, one of events e3 and e4 will occur resulting in s7 or s8. Events generally do not occur in isolation and often they are related. In Figure 1, events e1 and e4 are related by the relation r1, denoted as r1(e1,e4).

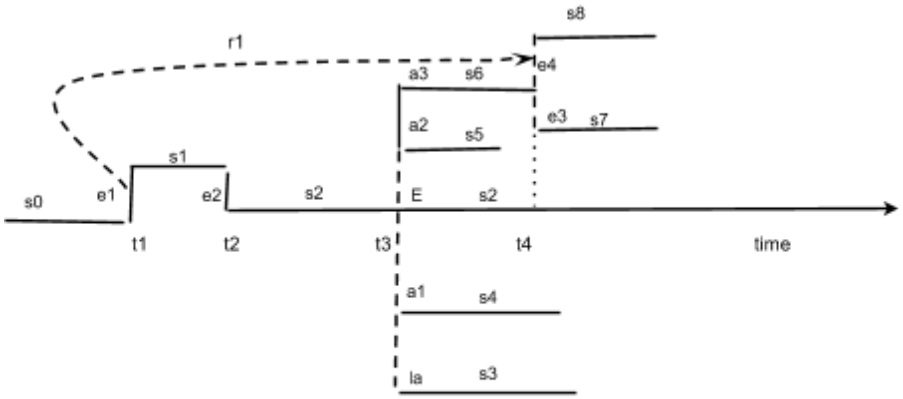

Figure 1. A simple task diagram.

Abstractions over events and states

Event diagrams can be simplified using abstractions over events and states. An event as well as a state can be an abstraction over events and states. Figure 2(a) shows a detailed event diagram and Figure 2(b) an abstract version of it. The event ea is an abstract event defined over events e1,e2,e3, and e4 and states s1,s2,and s3. Similarly, the state sa is an abstract state defined over the events e5,e6,e7,e8,e9,and e10, and the states s4,s5,s6,s7,s8,s9,and s10. Thus, we have an abstract event ea causing the state sa from the state s0, that is, ea:s0 → sa. Note the the relation r1(e1,e2) has been hidden in ea since r1 was viewed as a lower level level entity.

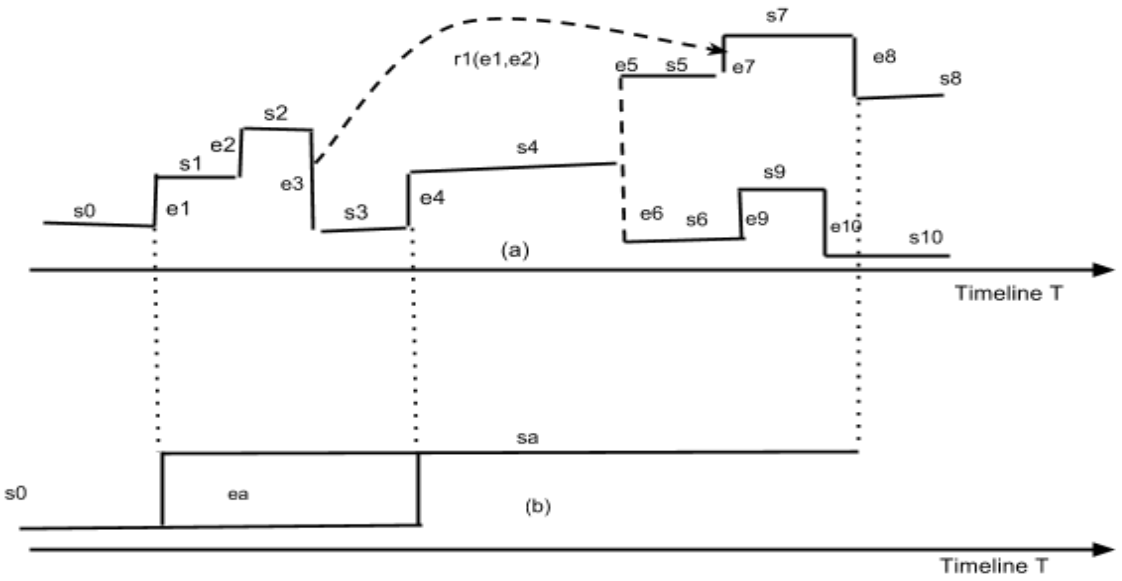

Figure 2. The event ea is an abstract event defined over events e1,e2,e3, and e4 and states s1,s2,and s3. Similarly, the state sa is an abstract state defined over the events e5,e6,e7,e8,e9,and e10, and the states s4,s5,s6, s7,s8,s9,and s10. Thus, ea: s0 → sa.

Abstraction over parallelly occurring events and states

Figure 3 shows an abstraction over parallel events. The event e12 is an event abstraction over the events e1 and e2, and the state s12 is a state abstraction over the states s1 and s2.

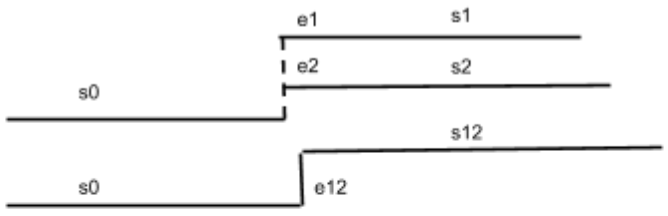

Figure 3. Abstraction over events and state. Event e12 is an abstraction over e1 and e2, and the state s12 is an abstraction over s1 and s2.

Relations

The following table contains some of the important relations that are useful in interactive task model.

Relation	**Description** (Events may occur in different timelines or on the same timeline.)
e1 enables e2	/* Successful occurrence of e1 at time t1 enables the occurrence of e2 at t2. If e1 does not occur (successfully), e2 will not occur. */ $\forall e1\ \forall t2 \forall e2\ \exists\ t1$ [t2 > t1 & Occurs(e2,t2) → Occurs(e1,t1)] Note that there may be more than one event enabling an event.
e1 commits e2	/* Occurrence of e1 at t1 will make the agent A commit to the occurrence of e2 at a later point t2 in time. */ $\forall e1\ \forall t1 \forall e2\ \exists\ t2$ [occurrence(e1,t1) & t1 < t2 → commit (A,e2,t2)]
e1 disables e2	/*The occurrence of e1 at t1 disables the occurrence of e2 at time in future. */ $\forall e1\ \forall t2 \forall e2\ \forall\ t1$ [Occurs(e1,t1) → ¬ Occurs(e2,t2)] where the symbol ¬ stands for negation.
e1 causes e2	/*Event e1 occurring on a timeline T1 causes an event e2 to occur on another timeline T2 */ $\forall e1\ \forall t1 \forall e2\ \exists\ t2$ [occurs(e1,t1) & t2 > t1 → occurs(e2,t2)]
e1 syncs e2	/* On a timeline T1, if an event e1 occurs at time t1, then no other events occur on T1 until event e2 occurs at time t2 on timeline T2. */ $\forall e1\ \forall t1 \forall e2\ \ \forall t2\ \forall e3 \forall t3$ [occurs(e1,t1) & occurs(e2,t1) & occurs(e3,t3) & t2 ≥ t1 → t3 > t2] /* That is, e3 has to wait for the occurrence of e2. */

Temporal constraints

Constraints are relations on resources,time, and space which are required to be satisfied at specified time points or time intervals during a process. While an exhaustive set of constraints is highly dependent on a given domain, below we list a set of temporal constraints that are relevant in business domains.

Deadline(s, t): State s must be achieved before time t.

Duration(s, t): The duration of a state s should not exceed t units of time.

Options(s, n): The number of options at the end of state s should be at least n. Larger the value

Criticality: There are states that are designated as critical states in an application. From this, we can define critical events as follows.

- If si is a critical state, then an event ei such that ei: sj → si is a critical event.

- If ei is a critical event, then any event ej that enables ei is a critical event.

The notion of criticality C(e,s) of an event e denotes how critical e is to finally cause a state s that is considered as critical. When multiple events e1,...,en enable an event e, the criticality of each event ei depends on how collectively they affect e. They may be classified as ANDenable or ORenable. We define degree of criticality d where 0 <= d <= 1 as follows:

  - ANDenable: In this case, degree d = 1 for each enabling event. For example, if e1 and e2 are events where both have to occur, then C(e1,s)=C(e2,s)=1.
  - ORenable: In this case, degree d = 1/n where n = number of enabling events. For example, if e1 or e2 are the enabling events then, C(e1,s)=C(e2,s)=1/2.

User Interactions

We model user interactions using the notion of intentions of agents and strategies for managing them in the interactive tasks. Intention lets an agent select parts of an event plan and execute them. An intention is modelled as a commitment to a set of sequences of (executable)events (actions) in the task model. With regard to Figure 1, we can define I1 and I2 as intentions where I1 = Commitment({ <a2>}) and I2 = Commitment({<a2,a3>}). An intention can be simple or compound. I(a2) for example is a simple intention where the agent adopts the intention I to execute the action a2, executes a2, and finally drops I as it is no longer needed. Similarly, after adopting intention I1 above, the agent adopts a subintention I(a2) to execute a2.

After executing a2, the intention I(a2) is dropped and finally I1 is dropped as well. (It may be pointed out that the agent can drop its attention at any time after adopting an intention.) While any random sequences of actions can be involved in defining an intention, it is useful to have some patterns in the sequences of events in real world business processes. We propose the following patterns.

(a) Simple Intention(SI) This intention is defined for a single action. For example I(a1) is a simple intention where a1 is an action.

(b) Compound Intention(COI) Often adopting an intention may result in adopting several subintentions. For example, in I2 =Commit({ <a2,a3>}) above, adopting I2 involves adopting subintention I(a2), and adopting subintention I(a3). Following are more complex forms of compound intentions.

(c) Conditional Intention(CI): In this, the agent initially commits to two sequences

of events. The agent then at the time of execution t, chooses one of them as its sub-intention

if some condition C is true at t. The agent holds the Conditional Intention until it finishes execution of the selected sequence of events and drops the subintention.

(d) Repetitive Intention (RI): The agent adopts the intention RI, denoted as RI(w,c), when a sequence w of events has to be executed several times until some condition c is satisfied. When the agent commits to RI(w,c), if c is true, then the agent first adapts a subintention I(w) to execute the events in w, executes w, drops RI(w,c), and then adopts a new intention RI'(w,c). No intention is adopted when c is false.

Intention supports

An intention may be supported by states of world objects and other intentions. Often intentions occur in nested forms. In such cases, inner level intentions are supported by outer level intentions. When supports are removed, the supported intention is dropped, and all its inner level intentions are also dropped.

Dropping an Intention: Agents must be careful about dropping intentions, since dropping intentions randomly can be harmful. Typically in interactive applications, we consider dropping intentions in the following cases:

- Simple Intention: Dropping simple intention amounts to aborting the execution of an action. Note that this may produce serious consequences as the world resources may be thrown into unforeseen states recovering from which can be problematic.
- Conditional Intention: Dropping this intention skips the execution of the entire conditional sequence.
- Repetitive Intention: Dropping this intention at any time results in skipping the remaining execution of the loop.

Constraint violation

The occurrence of an event may cause certain constraints to be violated. Typically, a constraint violation results in an error state of the resources(instead of a valid state). While the details of how to handle violations are specific to the application, we present below

the generic techniques for recovering from any constraint violation.

- **Return to closest normal state**: Perform operations on the resource to change the error state to a valid state as quickly as possible.
- **Return to closest reference state**: In this, some of the the normal states are identified as reference states. When the agent finds the resources in an error state, it attempts to perform actions so as to navigate the resource states to the closest reference state.

- **Change of subgoal** When the above two strategies fail, the agent examines the next subgoal G1(of a supergoal G) and replaces it by a new subgoal G1'(which is a valid subgoal of the supergoal G), obtains a new event plan, and continues execution.

APPLICATION TO SHOPPING

In a typical shopping activity, we categorize the events as follows:

- α type events: pick item from shelf, place item back in shelf, make payment, and finish shopping.
- λ type events : sale, nonavailability of items, phonemessages and lack of funds in account.
- μ type events: melting of frozen items, leaking cans, cooling of hot food, etc.
- ε type events: This refers to null events which is said to occur when events of other types do not occur and the agent chooses not to do any action. (in such cases the current state continues to exist since there are no events.)

Task Model

The shopping task is modelled using events as follows.

- Start with a consumption plan; then derive a buying plan expressed as an interactive task model.

- Every item will potentially have a consumption link using general and domain specific relations. That is, every item has a temporal behaviour. The absence of relations will make an item an orphan and orphaned items are to be minimized in a task plan. Typically, a buying event is linked to an interval of time, which may be specified at increasing levels of precision such as years, months and weeks, and hours. We use a calendar with increasing levels of (optional) refinements to refer to intervals over time in the format, for example, <year1,year2> which refers to an interval between year1 and year2. Refinement is specified by nesting. To refer to

an abstract point in time, we write year: [month: [day: [hour:[min:[sec1,sec2,...,secn], ..],..],...],..],..]. To refer to an interval, we use the tuple notation <timePoint1,timePoint2>. Thus, 2013:[Oct:[23:[<8:30,9:30>, <18:00,19:30>], <22,28>, 30], <Nov,Dec>] denotes a time point in the year 2013 using two time intervals <8:30,9:30> and <18:00,19:30> on 23 Oct 2013, an interval of seven days from Oct 22 to Oct 28, and an interval of two months from November to December. Note that years such as 2013, though are abstract time points, are actually intervals at the lower levels of abstraction. An event when occurring over an abstract interval will occur at any point in that interval. Thus, buyCarrot at 2013:[Oct:[23]] will occur at any time on 23 Oct 2013, and buyRice at 2013:[Oct] will occur at any time in the month Oct, 2013. buyOil in 2013:[Oct[<23,25>]] will occur at any time in the interval from 23 Oct 2013 to 25 Oct 2013. We can denote an event occurring over an interval using a similar notation. The notation e:[e1:[e10,e11],e2,e3:[e31,e31,e33: [e331,e332],e34], e4] specifies an event at four levels of abstraction. An abstract event may occur at a single abstract time instant, where the components of the abstract event are mapped on to time points in the given abstract timeline.

Every item that is bought must be related to a consumption behaviour. At any given level of abstraction, consumptions may be instantaneous or may occur over an interval of time. Each item may induce a consumption behaviour that is specific to itself.

We are now ready to specify consumption for a few examples.

- Milk is consumed regularly morning and evening. We express this as: milkconsumeIn (w1) & (w1 = 2013:[Oct:[21,22,23:[<0830,0930>,<1800,1900>],24,25,26,27]])
- Consumeyoghurt (weeks12) & weeks12 = [21Oct2013, 22Oct-2013, 24Oct2013, 25Oct2013, 26Oct2013, 28Oct2013, 29Oct2013, 1Nov2013, 2Nov2013, 4Nov2013].

Constraints

Constraints in the shopping domain specify permitted resource states. Following are typical constraints in this domain.

Object constraints This type of constraints specify the relations between states of objects that must be satisfied.

a. If an item x is bought, then the item y has to be bought, too.
b. If n is the number of items of type X bought, then n0 <= n <= nmax.
c. Budget < nmax & overallweightofallitems < wmax.

Temporal constraints This type of constraints specify the relations between temporal quantities.

d. Shopping must be finished within 2 hours.
e. Item x must be bought on all Saturdays in a month.
f. Buy item x at the end of a shopping interval.
g. Buy item x not more than once in a month.

SHOPPING ASSISTANT

The architecture of a preliminary version of a shopping agent implemented on an Android mobile device is shown in Figure 4 below [Cheng 2013]. The shopping plan implicitly uses an interactive task model implemented using a voice driven user interface. The dialogs (represented as dialog trees) between the user and the VUI exploits the flexibilities built into the shopping plan.

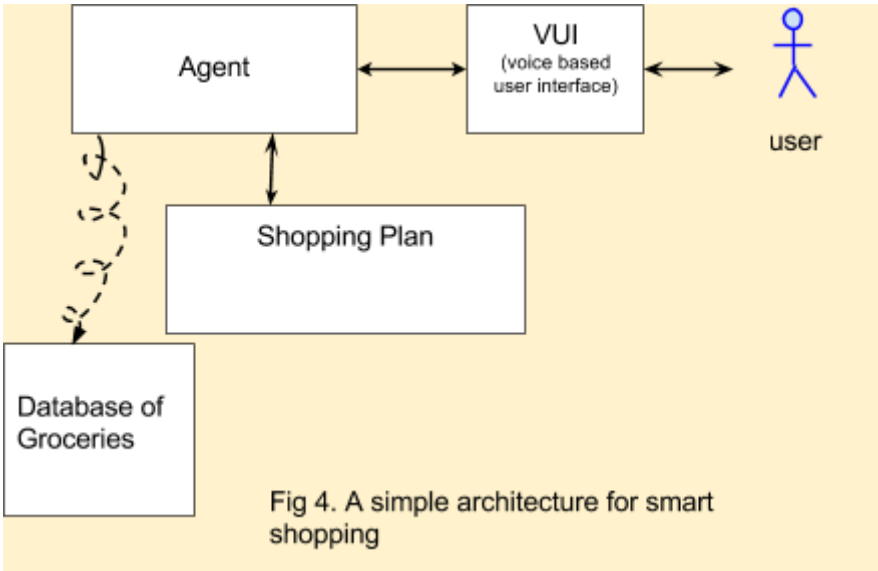

Fig 4. A simple architecture for smart shopping

RELATED WORK

Events and states as foundational concepts to language understanding were first proposed in [Parsons 1990]. Actions and events occurring over intervals of time was initially proposed by Allen [Allen et al. 1994] along with thirteen temporal relations

between temporal intervals which were popularly used in many applications ever since. The use of event calculus in reasoning with common sense knowledge is well known [Mueller 2010]. Work on interactive task models is very minimal so far despite its importance in business process applications. [Iqbal and Bailey 2007] discusses the need for breakpoints in interactive tasks. Constraints is a well studied concept and has found applications in many real world tasks [Ashamalla et al. 2012]. Presently, use of event ontologies have found their way in simplifying quick development of event based applications[Event 2007] .

CONCLUSION

In this paper, we have presented a model for interactive task representation. We have argued that the task model is naturally represented using events and task model essentially is an event model incorporating actions and unexpected events that occur in a real world. We have also shown how abstractions can be defined over events and time, and how abstract events can be scheduled over a time interval. Some preliminary details of an initial implementation of our model in the shopping domain implemented on an Android mobile device has also been presented. The performance results from the implementation will be reported elsewhere later. The ultimate goal of this research is to extend this approach to model stories and subsequently apply the model to real world multiagent interactive tasks.

ACKNOWLEDGEMENT

The implementation referred to in this paper on an Android phone was carried out by Harry Shing Yan Cheng[Cheng 2013] .

REFERENCES

[Allen et al. 1994] Actions and Events in Interval Temporal Logic JAMES F. ALLEN and GEORGE FERGUSON, J. Logic Computation, 5, pp531579, 1994.

[Ashamalla et al. 2012] Ashamalla, A., Beydoun, G., Low, G. & Yan, J., Towards modelling real time constraints, ICSOFT 2012: 7th International Conference on Software Paradigm Trends, pp. 158164, 2012.

[Cheng 2013] Cheng H.S.Y., An Assistant on a Mobile Phone for Grocery Shopping, Honours Thesis, SCHOOL OF COMPUTER SCIENCE AND

ENGINEERING, THE UNIVERSITY OF NEW SOUTH WALES, Sydney 2013.

[Event 2007] EVENT The Event Ontology, http://lov.okfn.org/dataset/lov/details/vocabulary_event.html.

[Iqbaln and Bailey, 2007] S Iqbal and B Bailey, Understanding and developing models for detecting and differentiating breakpoints during interactive tasks, Proceeding CHI '07 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Pages 697706, ACM New York, NY, USA 2007.

[Mueller 2010] Erik T. Mueller, Commonsense Reasoning, Morgan Kaufmann, 2010. [Parsons 1990] Events in the Semantics of English, T Parsons, MIT Press, Cambridge, Massachusetts, 1990