An integrated framework for temporal aggregation and omission in the Event Calculus

L. Chittaro, A. Montanari, E. Peressi
Dipartimento di Matematica e Informatica, Università di Udine, Via delle Scienze, 206, 33100 Udine, Italy

1 Introduction

Temporal reasoning is a major requirement in many intelligent system applications such as planning and scheduling. These applications need the ability to store information about past, current, and future states of the considered domain [6], as well as the ability to manage temporal information not only to retrieve it as it was stored in the database, but also to automatically derive further data by means of temporal projection [4, 5].

In this paper, we consider two key temporal reasoning mechanisms: aggregation and omission. Temporal aggregation is essentially an abstraction mechanism that allows one to disregard negligible phenomena. Temporal omission exploits domain knowledge to limit temporal projection in the case of incomplete information. We will show that both mechanisms can be easily supported by Kowalski and Sergot’s Event Calculus (EC hereinafter) [3] and provide them with an efficient implementation.

EC database and rules are formulated in a logic programming framework (PROLOG) as a temporal deductive database. From a description of events that occur in the real world and properties they initiate and/or terminate, EC allows one to derive the maximal validity intervals (MVIs hereinafter) over which properties hold. Moreover, EC makes it possible to constrain the initiation and termination of properties to the validity of some given conditions at the time of events’ occurrences [1].

The paper is organized as follows. Section 1 introduces the basic features of EC, and shows how EC deals with temporal aggregation and omission in a uniform way by means of the distinction between weakly and strongly initiated properties. Section 2 describes the behaviour of the system. Section 3 applies the proposed system to a real-world domain.
2 Supporting Aggregation and Omission

Kowalski and Sergot’s EC is a general approach to representing and reasoning about events and their effects in a logic programming framework. It takes the notions of event, property, time-point and time-interval as primitives and defines a model of change in which events happen at time-points and initiate and/or terminate time-intervals over which some property holds. Insertion of events in the database is not required to follow the chronological order of their occurrences. The inference mechanism embeds a notion of default persistence according to which properties are assumed to persist until an event occurs that interrupt them.

Formally, we represent an event occurrence by means of the unit clause \( \text{happens-at}(\text{event}, \text{timePoint}) \). The link between events and properties is defined by means of \( \text{initiates-at} \) and \( \text{terminates-at} \) clauses, containing a (possibly empty) sequence of preconditions (\( \text{holds-at} \) predicate):

\[
\begin{align*}
\text{initiates-at}(\text{event1}, \text{prop1}, T) :&: \text{happens-at}(\text{event1}, T), \\
&: \text{happens-at}(\text{event2}, T), \\
&: \text{holds-at}(\text{p1}, T), \ldots, \text{holds-at}(\text{pM}, T).
\end{align*}
\]

Two alternative interpretations can be given to the \( \text{initiates-at} \) relation. According to a \textbf{weak} interpretation, an event \( e \) initiates a property \( p \) unless \( p \) has been already initiated and not yet terminated, while, according to a \textbf{strong} interpretation, \( e \) initiates \( p \) if (and only if) no events strongly initiating \( p \) occur after \( e \) before \( p \) is terminated. As an example, consider the case of an ordered sequence of events \( e_1, \ldots, e_n \), each one initiating the same property \( p \), followed by an event \( e_{n+1} \) terminating \( p \). According to a weak interpretation of \( \text{initiates-at} \), EC derives the validity of \( p \) between the outermost initiating event \( e_1 \) and \( e_{n+1} \). On the contrary, if a strong interpretation is adopted, then EC derives the validity of \( p \) between the innermost initiating event \( e_n \) and \( e_{n+1} \).

Weak and strong \( \text{initiates-at} \) relations can be used to support temporal aggregation and omission, respectively. For example, consider the problem of monitoring patients who receive a partial mechanical respiratory assistance [2]. A basic requirement of the patient monitoring task is the ability of aggregating similar observed situations [2]. It indeed often happens that data acquired with two consecutive samplings do not cause a transition in the classification of the patient ventilatory state. Temporal aggregation requires that the second data acquisition does not clip the MVI of patient state initiated by the first one. Such a functionality can be easily supported by EC, provided that a weak interpretation of \( \text{initiates-at} \) is assumed.

Omission is needed when dealing with incomplete sequences of events. Consider a switch that can take two different positions: on and off. Its behaviour can be described by means of two properties: on and off, and two events: \( \text{turn-on} \) (resp. \( \text{turn-off} \)), that changes the position from off to
on (resp. from on to off). While two turn_on (resp. turn_off) events cannot occur consecutively in the real world, it may happen that an incomplete sequence consisting of two consecutive turn_on events \( e_1, e_2 \), followed by a turn_off event \( e_3 \), is recorded in the database. In such a case, a strong interpretation of initiates_at allows EC to recognize that a missing turn_off event must have occurred between \( e_1 \) and \( e_2 \). However, since it is not able to temporally locate such an event, it only derives the validity of the property on between \( e_2 \) and \( e_3 \), and considers \( e_1 \) as a pending initiating event.

The terminates_at relation states that an event \( e \) terminates a property \( p \) unless it has already been terminated. This interpretation of terminates_at must be refined to deal with the case of an event \( e \) terminating a property \( p \) that is neither preceded by events initiating \( p \) nor by events terminating \( p \). In such a situation, two alternative choices are possible: (i) if persistency in the past is assumed, then \( p \) holds from the beginning of time, (ii) if persistency in the past is not assumed, then \( p \) does not hold over any time period preceding \( e \) (\( e \) is a pending terminating event).

The following axioms extend the EC model of time and change to deal with weak and strong interpretations of initiates_at (wInitiates_at and sInitiates_at, respectively), and past-persistent and not past-persistent terminates_at relations (pTerminates_at and npTerminates_at, respectively) in an integrated framework:

\[
\text{mholds_for}(P, [\text{Start}, \text{End}]) : - \\
\text{initiates_at}(E_i, P, \text{Start}), \text{terminates_at}(E_t, P, \text{End}), \text{End} \gt \text{Start}, \text{\n+ broken_during}(P, [\text{Start}, \text{End}]).
\]

\[
\text{mholds_for}(P, [\text{Start}, \text{infPlus}]) : - \\
\text{initiates_at}(E_i, P, \text{Start}), \text{\n+ broken_during}(P, [\text{Start}, \text{infPlus}]).
\]

\[
\text{mholds_for}(P, [\text{infMin}, \text{End}]) : - \\
\text{pTerminates_at}(E_t, P, \text{End}), \text{\n+ broken_during}(P, [\text{infMin}, \text{End}]).
\]

\[
\text{broken_during}(P, [\text{Start}, \text{End}]) : - \\
\text{Start} \nleq \text{infMin}, \\
(\text{terminates_at}(E, P, T); \text{sInitiates_at}(E, P, T)), \text{Start} \lt T, \text{End} \gt T.
\]

\[
\text{broken_during}(P, [\text{infMin}, \text{End}]) : - \\
(\text{terminates_at}(E, P, T); \text{initiates_at}(E, P, T)), \text{End} \gt T.
\]

where the predicate \( \text{gi} \) extends the ordinary ordering relationship \( \gt \) to include the cases involving infinite arguments, syntactically denoted by \( \text{infMin} \) and \( \text{infPlus} \) (analogously, we defined the predicates \( \geq \), \( < \) and \( \leq \), respectively), and \( \text{initiates_at} \) and \( \text{terminates_at} \) are defined as follows:

\[
\text{initiates_at}(E, P, T) : - \\
\text{sInitiates_at}(E, P, T); \\
\text{wInitiates_at}(E, P, T).
\]

\[
\text{terminates_at}(E, P, T) : - \\
\text{npTerminates_at}(E, P, T); \\
\text{pTerminates_at}(E, P, T).
\]
Clause (1.1) states that a property \( P \) maximally holds between events \( E_i \) and \( E_t \) if \( E_i \) initiates \( P \) and occurs before \( E_t \) that terminates \( P \), provided there is no known interruption in between. The negation involving the \( \text{broken\_during} \) predicate is indeed interpreted using negation-as-failure. Clauses (1.2) and (1.3) deal with cases of persistence in the future and in the past, respectively. Clause (1.4) states that a given property \( P \) is interrupted between \( E_i \) and \( E_t \) if there is an event \( E \) that happens between them which terminates or strongly initiates \( P \). Finally, clause (1.5) is added to make a MVI for a property \( P \) starting at \( \text{infMin} \) sensitive also to occurrences of weak initiating events for \( P \) inside that MVI.

3 Description of the System

The model described in the previous section has been implemented by means of a suitable extension of the Cached Event Calculus (CEC hereinafter) [1], providing an integrated framework for dealing with weak/strong interpretations of the \( \text{initiates\_at} \) and past-persistent/not past-persistent \( \text{terminates\_at} \) relations.

CEC extends the original implementation of EC with a \textit{MVIs generation and storage} mechanism. CEC caches MVIs of properties for later use in query processing and possibly updates them when a new event is entered in the database. Entering a new event can lead either to clip an existing MVI or to possibly create a new MVI. When a MVI, or a part of it, is retracted (resp. asserted), CEC takes care of propagating that change to properties which depend on the changed one. Propagation of assertion and retractions can recursively activate the process of breaking or creating MVIs.

In the next section, we will describe the process of breaking and creating MVIs. We will focus specifically on the problem of integrating the different interpretations of \( \text{initiates\_at} \) and \( \text{terminates\_at} \).

3.1 Breaking and Adding MVIs

When a new event is entered in the database, CEC handles separately its effects on the initiation and termination of properties. The relevant cases are graphically depicted in Figure 1, and described in the following.

**Strong initiation** If \( E \) strongly initiates a property \( P \) at time \( T \), CEC tests if there exists a MVI \([T_1, T_2]\) for \( P \) (where either \( T_1 \) or \( T_2 \) may be infinite) such that \( T_1 \leq T < T_2 \). If such an interval exists, then \( E \) may possibly break it. On the contrary, if it does not exist, then \( T \) can be the starting point of a new MVI for \( P \). More specifically, the two cases are handled as follows:
1. If the interval exists, we face two possibilities: (i) if $T_1 = T$, there is already an event occurring at $T$ that initiates $P$ and then no changes are needed to interval $[T_1, T_2]$ at the moment, (ii) otherwise, interval $[T_1, T_2]$ is shortened (cases A.1, A.2 and B.2): the new starting point is $T$ and $P$ does not hold anymore in the clipped part $[T_1, T]$.

2. If the interval does not exist, four cases apply: (i) if there is a strongly initiated MVI for $P$ after $T$, no action is taken (cases A.3 and A.4); (ii) if there is a ‘pending’ terminating event for $P$ occurring at time $T_3$ after $T$ with no interruptions in between, then the new MVI becomes $[T, T_3]$ (case B.1), (iii) if there are neither terminating nor strongly initiating events for $P$ occurring after $T$, then the new MVI becomes

---

**Figure 1. Different cases of update.**
Weak initiation If \( E \) weakly initiates a property \( P \) at time \( T \) and there exists a MVI \([T_1, T_2]\) for \( P \) such that \( T_1 \leq T < T_2 \), two cases must be distinguished: if \( T_1 \) is not \( \text{infMin} \), then \( E \) does not break the MVI (cases C.1 and C.2), otherwise the new MVI becomes \([T, T_2]\) (case B.2).

If the MVI does not exist, CEC verifies whether cases B.1 or B.3 apply or there exists a weakly initiated MVI \([T_1, T_2]\) such that \( T < T_1 \), and there are no terminating events between them (cases C.3 and C.4).

Past-persistent and not past-persistent termination The main cases of update for terminating events are symmetrical to the cases of update for strongly initiating events: D.1 corresponds to A.1, D.2 to B.2, E.1 to B.1, and E.2 to B.3. The only special case (F.1) occurs when there is a MVI \([\text{infMin}, T_1]\) for \( P \) with a past-persistent terminating event and a new not past-persistent terminating event for \( P \) occurring at \( T < T_1 \) is recorded. In this case, the effects of the two terminating events taken in isolation are incompatible, and then the whole MVI is retracted.

4 A case study

In the following, we will show a fragment of a CEC temporal knowledge base concerning an health care management domain. The predicates we consider here are devoted to the modeling of some aspects of extraordinary maintenance of machines in a hospital. Unlike preventive maintenance, that is, a periodic verification of the normal functioning of machines, extraordinary maintenance is performed following sudden failures. Some simplifications have been introduced, in particular to remove preconditions. Four different events are modeled by the considered predicates. The first event describes the effect of receiving a failure report (\( \text{flRpt} \) event with two arguments: the machine involved, and a reference to the specific report filed). The machine referred in the report is declared unavailable (the property \text{status} classifies the status of a machine, that can be available, partially available, or unavailable). Notice here that the initiation of the unavailable status for a machine is a \( \text{wInitiates}_{-}\text{at} \), because there can be more than one report filed about the same machine by different people.

\[
\text{wInitiates}_{-}\text{at}(\text{flRpt}(\text{Mchn}, \text{RptNum}), \text{status}(\text{Mchn}, \text{unavailable}), T) :\neg \\
\text{happens}_{-}\text{at}(\text{flRpt}(\text{Mchn}, \text{RptNum}), T).
\]

\[
\text{npTerminates}_{-}\text{at}(\text{flRpt}(\text{Mchn}, \text{RptNum}), \text{status}(\text{Mchn}, \text{available}), T) :\neg \\
\text{happens}_{-}\text{at}(\text{flRpt}(\text{Mchn}, \text{RptNum}), T).
\]

In response to reports of failure, a technician is sent to precisely assess the case. This is modeled by the extraordinary assessment (\( \text{extAst} \)) event.
The first two arguments of this event specify the considered machine and a pointer to the detailed static data of the assessment report. The assessment is a quick evaluation that results mainly in two decisions (indicated by the third and fourth argument of the event). The first decision is about the status of the machine: the technician can declare the machine available (e.g. the failure reports were actually false alarms), partially available, or unavailable. The second decision is about who should repair the machine (internal personnel, external personnel, or no repair needed) and is represented by the required maintenance \((rqrMnt)\) property.

\[
\text{wInitiates\_at}(\text{extAst}(\text{Mchn}, \text{AstNum}, \text{Sts}, \text{MntPrsn}), \text{status}(\text{Mchn}, \text{Sts}), T):- \\
\text{happens\_at}(\text{extAst}(\text{Mchn}, \text{AstNum}, \text{Sts}, \text{MntPrsn}), T).
\]

\[
\text{npTerminates\_at}(\text{extAst}(\text{Mchn}, \text{AstNum}, \text{Sts}, \text{MntPrsn}), \text{status}(\text{Mchn}, \text{OldSts}), T):- \\
\text{happens\_at}(\text{extAst}(\text{Mchn}, \text{AstNum}, \text{Sts}, \text{MntPrsn}), T), \\
\text{Sts} \neq= \text{OldSts}.
\]

\[
\text{sInitiates\_at}(\text{extAst}(\text{Mchn}, \text{AstNum}, \text{Sts}, \text{MntPrsn}), \text{rqrMnt}(\text{Mchn}, \text{MntPrsn}), T):- \\
\text{happens\_at}(\text{extAst}(\text{Mchn}, \text{AstNum}, \text{Sts}, \text{MntPrsn}), T), \\
\text{MntPrsn} \neq= \text{noRepair}.
\]

The start of a maintenance activity is modeled by the start maintenance \((strMnt)\) event. The three arguments of this event specify: the involved machine, the personnel performing the maintenance (internal or external), and a pointer to a detailed report about the maintenance activity. During maintenance, the machine is not just declared unavailable, but a specific property \((inMnt)\) describes which machine is in maintenance and which type of personnel is servicing it. Notice here that the initiation of the unavailable status of the machine is a \(\text{wInitiates\_at}\), because the machine can be already unavailable when the maintenance starts. On the contrary, the initiation of the \(\text{inMnt}\) property is a \(\text{sInitiates\_at}\), because it is not possible that the same machine can be already serviced by the same personnel, when the maintenance starts.

\[
\text{wInitiates\_at}(\text{strMnt}(\text{Mchn}, \text{MntPrsn}, \text{MntNum}), \text{status}(\text{Mchn}, \text{unavailable}), T):- \\
\text{happens\_at}(\text{strMnt}(\text{Mchn}, \text{MntPrsn}, \text{MntNum}), T).
\]

\[
\text{sInitiates\_at}(\text{strMnt}(\text{Mchn}, \text{MntPrsn}, \text{MntNum}), \text{inMnt}(\text{Mchn}, \text{MntPrsn}), T):- \\
\text{happens\_at}(\text{strMnt}(\text{Mchn}, \text{MntPrsn}, \text{MntNum}), T).
\]

\[
\text{npTerminates\_at}(\text{strMnt}(\text{Mchn}, \text{MntPrsn}, \text{MntNum}), \text{status}(\text{Mchn}, \text{OldSts}), T):- \\
\text{happens\_at}(\text{strMnt}(\text{Mchn}, \text{MntPrsn}, \text{MntNum}), T), \\
\text{OldSts} \neq= \text{unavailable}.
\]

\[
\text{pTerminates\_at}(\text{strMnt}(\text{Mchn}, \text{MntPrsn}, \text{MntNum}), \text{rqrMnt}(\text{Mchn}, \text{MntPrsn}), T):- \\
\text{happens\_at}(\text{strMnt}(\text{Mchn}, \text{MntPrsn}, \text{MntNum}), T).
\]

The end of a maintenance activity is modeled by the end maintenance \((endMnt)\) event. The first three arguments of this event are equivalent to
the first three arguments of the \textit{endMnt} event. The fourth argument specifies the status of the machine as a result of the maintenance performed (it is not necessarily available). After the end of maintenance, an extraordinary assessment usually takes place, and can result in a request for further maintenance activity.

\begin{verbatim}
\textbf{wInitiates\_at}(@\textit{endMnt(Mchn,MntPrsn,MntNum,Sts)},\textit{status(Mchn,Sts)},T):-
\textit{happens\_at}(\textit{endMnt(Mchn,MntPrsn,MntNum,Sts)},T).

\textbf{npTerminates\_at}(@\textit{endMnt(Mchn,MntPrsn,MntNum,Sts)},\textit{status(Mchn,unavailable)},T):-
\textit{happens\_at}(\textit{endMnt(Mchn,MntPrsn,MntNum,Sts)},T),
\textit{Sts} \neq \text{unavailable}.

\textbf{npTerminates\_at}(@\textit{endMnt(Mchn,MntPrsn,MntNum,Sts)},\textit{inMnt(Mchn,MntPrsn)},T):-
\textit{happens\_at}(\textit{endMnt(Mchn,MntPrsn,MntNum,Sts)},T).
\end{verbatim}

Acknowledgments
This work has been partially supported by the P.A.O.L.A. Consortium (ASEM, INSIEL, and Università di Udine).

References


