The Difference a Day Makes – Recognizing Important Events in Daily Context Logs

Michael Wessel[1], Marko Luther[2], and Matthias Wagner[1]

[1] Racer Systems GmbH & Co. KG, Blumenau 50, 22089 Hamburg, Germany
[2] DoCoMo Euro-Labs, Landsbergerstr. 312, 80687 Munich, Germany

Abstract. We study the extension of context ontologies towards enhanced qualitative spatio-temporal representations and reasoning. Our goal is to model and extract events that are important to the user from her context log, i.e. the history of context data collected over a longer period. We present a case study based on actual context ontologies and context data from the ContextWatcher mobile application. The presented work has been fully implemented in the DL-based reasoning engine RACERPRO.

1 Introduction

In this paper we present the representation and recognition of significant events within the context data that a mobile user collects over a longer period such as a couple of days. As contextual data sources we assume data collected by ContextWatcher [1]. ContextWatcher[3] is a mobile application that facilitates easy gathering and sharing of personal context from an underlying network of context providers. These context providers include the user's location in terms of the present location, location traces as well as frequently visited places, all kinds of user-tagged objects and activities, and location-specific information extracted from public sources, such as local weather information. In ContextWatcher, the value of personal context information is multiplied by sharing context with others through networks of defined social relationships.

ContextWatcher is implemented as a self-contained mobile client but can also connect to third party applications. A currently very popular application is the automatic compilation of gathered context into personal daily Web logs, for instance, to show pictures taken on the phone in a certain context, display visits to selected places or to disclose social encounters. Such contextual blogs have been a strong motivation for the work presented in this paper: to enhance the blog readability, to make sharing of posts easier and to simply make blogs more attractive, enhanced concepts to model and recognize important events are needed.

As most data delivered by our context providers is of quantitative nature in the first place, abstraction methods and context ontologies haven been introduced to deal with context at a higher level [2]. At the level of these context ontologies, complex conceptual dependencies between context elements are introduced to enrich contextual descriptions and to implement classification-based reasoning about the user's situation. Qualitative context descriptions were firstly introduced in ContextWatcher to describe user places as conceptual abstractions from exact locations. Examples include place descriptions like "Office", "Home" ore "Business Place". As the supporting context

[3] `http://www.contextwatcher.com`

ontologies evolved, more qualitative concepts that connect to these place descriptions were added. In the current version of ContextWatcher, the linkage between exact physical user locations and qualitative places is implemented through clustering methods which are applied to user traces.

In this paper we exploit extensions of the existing ContextWatcher ontologies towards enhanced qualitative spatial representation and reasoning. We study the implementation of a complex event recognition and management system with RACERPRO [4] as our DL-based reasoning component of choice. The rest of this paper is organized as follows. We first define required terminology and sketch the overall architecture of the proposed framework. Then we describe our RACERPRO event model. We illustrate the potentials of the modeling with an example scenario. Finally we conclude. In the following we assume some basic knowledge of Description Logics (DLs) [3] and related semantic technologies (e.g., W3C standards such as OWL[5] and semantic query languages such as nRQL [4]).

2 Context Awareness and Event Recognition

The most prominent definition of *context* was coined by A. K. Dey et al.: *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

Such a piece of relevant "‘information"’ is also called a *context element*. As mentioned in the introduction, context elements are provided by *context providers.* In this case study, we are primarily considering the context elements *location* and *time*. The collection of all context elements is called the current context or current *situation*.

Situational reasoning [2] uses background knowledge specified in OWL ontologies to infer additional context elements from the asserted ones. One can claim that context providers merely provide *raw context data,* and that this context data can only be tranformed into context elements (being characterized as "information") by means of *interpretation.* This interpretation is performed with the help of logical reasoning.

Currently, the approach taken by ContextWatcher is to map the context data into *context assertions* in a *context or situation ABox* in RACERPRO. The *ABox realization service* (which is a standard DL inference service) is then used by ContextWatcher to derive the entailed, logically implied ABox context assertions. Each agent is represented as an individual in the ABox, describing the agent's current context. The ABox also includes social as well as spatial relationships.

The mapping function from *context data* to *context assertions* is currently defined procedurally. For example, the location of an agent is provided by a GPS device. So-called *location clusters* are acquired from GPS agent traces which are analyzed offline by statistical learning / clustering methods to find so-called location clusters. An acquired location cluster can then be annotated by the user with an OWL class or DL concept, e.g. *home* or *office*. Membership in these clusters is from now on recognized automatically by ContextWatcher, and appropriate *qualitative* location assertions are put into the context ABox. This mapping function (which not only takes care of location) is called the *Situation Description Generator* in the following. In many cases,

[5] `http://www.w3.org/TR/owl-ref/`

quantitative context data is mapped to *qualitative* context assertions such that OWL or DL reasoning can be exploited, which primarily works on a qualitative, symbolic level. However, by exploiting the *expressive concrete domain reasoning facilities* of RACERPRO we will also show how reasoning on quantitative (time) context data can be performed and exploited.

Context ABoxes in ContextWatcher so far can be described as static descriptions of "snapshots" in space-time. We claim that the recognition of *dynamic space-time histories*, so called events, can provide valuable additional context elements for ContextWatcher – in fact, certain situations can only be recognized if the *situational changes* are considered rather than the (static) situations themselves. For example, the situation *leaving home* is characterized by a *certain change in the agent's situation:* First the agent is *inside* its home cluster, then, in the next situation, he no longer is. In case the event takes place in the early morning hours of a working day, it is reasonable to assume that the agent is leaving his home for work. In case he should be too late, an SMS could be send automatically to his boss, apologizing in advance for being late. Moreover, as mentioned in the introduction, event structures which have been recognized in daily context logs can be used for the automated generation of diary-like blogs. Thus, a *dynamic* context ABox and DL-based event model is needed in which *notions of time and change play a major role.*

3 DL-based Event Recognition – A Case Study with RACERPRO

Our RACERPRO event recognition model includes *three basic building blocks:* a model of time, a model for situations, and an event model. In the following, a situation is called a *state* to make the resemblance with temporal modal logics or AI planning formalisms [5,6] more explicit.

Time Points and Intervals The basic temporal building blocks are *time points* and *intervals*. Let us start with time points. A time point is any ABox individual which has a real valued filler of the `time` attribute in the concrete domain of $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$, which is the DL implemented by RACERPRO:

```
(define-concrete-domain-attribute time :type real)
(define-concept point-in-time (a time))
```

Two individual time points `p1` and `p2` can be modeled in the ABox as follows:

```
(instance p1 (= time 6.5))
(instance p2 (= time 8.0))
```

Certain day times can be modeled as defined concepts:

```
(define-concept early-morning-time (and (<= 6.0 time) (< time 7.0)))
```

Note that `p1` is an instance of `early-morning-time` then. We also want to be able to reason about the *relative locations* of time points to one another, e.g., we want to know whether `p1` is before of after `p2`. A mapping to qualitative relationships such as `before-point-in-time` and `after-point-in-time` is thus needed. In RACERPRO we can use *defined* NRQL *queries* or NRQL *ABox rules* to establish such a mapping:

```
(defquery before-point-in-time (?s1 ?s2)
          (and (?s1 point-in-time) (?s2 point-in-time)
               (?s1 ?s2 (constraint time time <))))
```

A `defquery` form can be understood as a simple macro which can be used in nRQL queries such as `(retrieve (?x ?y) (?x ?y before-point-in-time))`, which then returns `(((?x p1) (?y p2)))`. But in order to make ABox reasoning aware of the qualitative relationship holding between `p1` and `p2`, we must add an `before-point-in-time` *role assertion* to the ABox. This can be done with an ABox rule:

```
(firerule (?x ?y before-point-in-time)
    ((related ?x ?x before-point-in-time-role)))
```

This rule fires and adds a `(related p1 p2 before-point-in-time-role)` assertion to the ABox. Due to the added role assertion, `p1` can now for example be recognized as an instance of the concept

```
(define-concept has-successor-point
        (and point-in-time (some before-point-in-time-role point-in-time)))
```

Having modeled time points, we can continue defining *intervals* which have a start and end time point; moreover, the intervals duration shall be greater than zero:

```
(define-concrete-domain-attribute start-time :type real)
(define-concrete-domain-attribute   end-time :type real)
(define-concept interval (and (a start-time) (a end-time)
                              (< start-time end-time)))
```

Given this definition of interval, it is even possible to classify / recognize events as *short or long intervals*, again by means of the expressive concrete domain reasoning offered by RacerPro:

```
(define-concept short-interval
                (and interval (< end-time (+ start-time 1.0))))
```

A short interval is thus an interval that lasts at most one hour; note that `(< end-time (+ start-time 1.0))` is satisfied iff $end\_time - start\_time < 1$; this equation cannot be expressed in a more direct way in RacerPro.

The *point in interval* relationship is an important qualitative relationship. It can be modeled as a defined query as follows:

```
(defquery point-in-time-inside-interval (?s ?e)
          (and (?s point-in-time) (?e interval)
               (?e ?s (constraint start-time time <=))
               (?s ?e (constraint time end-time <=))))
```

It is now reasonable to define *certain special day times* as *interval individuals*, e.g., like `morning-hours`. The rationale is that these intervals can be used in queries such as *What happened during the morning hours?*:

```
(instance early-morning-interval
          (and interval (= start-time 6.0) (= end-time 7.0)))
```

Moreover, the famous *Allen temporal relationships* [7] provide well-known qualitative temporal relational vocabulary for intervals (meets, overlaps, during, ...). Like the point in interval relation, the Allen relations can be defined as queries. If required, corresponding ABox rules can again add *Allen role assertions* to the ABox so that further reasoning processes are aware of the qualitative temporal relationships holding between the intervals. The `meets` relationship between intervals looks as follows:

```
(defquery meets (?e1 ?e2)
                (and (?e1 interval) (?e2 interval)
                     (?e1 ?e2 (constraint end-time start-time =))))
```

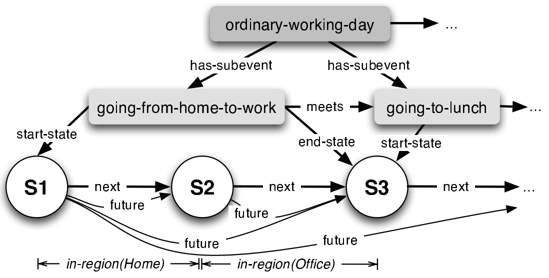

Fig. 1. States (Circles), Simple Events (Light Gray), and Complex Events (Gray)

From Time Points to States and Histories A *state* is a user/agent-specific description of the user's current situations as well as of its relevant (spatial, social, ...) environment at a given time point. States will be generated by the Situation Description Generator. Every time point that *has some agent* associated with it is called *a state of that agent*:

```
(define-concept state (and point-in-time (some has-agent agent)))
```

An arbitrary amount of additional context information can be "attached" to a state individual; for example, information regarding the current location for which we are using the `in-region` role. Regions can be cluster regions annotated with location concepts, but also annotated map regions, points of interest etc:

```
(define-primitive-role in-region :domain state :range region)
```

A *sequence of states of an agent* is called a *history.* Like in a modal temporal logic based on a discrete linear model of time, we are are introducing a functional role `next` to reference the successor state. The inverse of next is called `previous`; `next` has a transitive super-role called `future` which can thus be used to access all future states from the current state. Obviously, `past` is the inverse of `future`.

Since the state individuals of the agents are generated by the Situation Description Generator, the generator can as well create the required `(related s1 s2 next)` role assertions to produce the time thread. However, since the quantitative time information is available, the required *next role assertions* can also be created with an ABox rule:

```
(firerule (and (?s1 state) (?s1 ?a has-agent)
               (?s2 state) (?s2 ?a has-agent)
               (?s1 ?s2 before-point-in-time)
               (neg (project-to (?s1 ?s2 ?a)
                        (and (?s1 ?s before-point-in-time)
                             (?s ?s2 before-point-in-time)
                             (?s ?a has-agent)))))
    ((related ?s1 ?s2 next)))
```

The variables `?s1` and `?s2` will be bound to states of the same agent `?a`. Moreover, `?s1` precedes `?s2` in time. We also have to verify that `?s2` is the *direct* successor of `?s1`. This means that there is no state `?s` *in between* `?s1` and `?s2` of that same agent. This is verified with the expression `(neg (project-to ...))`. If satisfying `?s1`, `?s2` bindings are found, the rule adds a `(related ?s1 ?s2 next)` assertion to the ABox.

From Histories to Events Now we have an ABox containing all the histories of the agents. Events shall now be recognized on agent histories. An example history of an agent on which events have been recognized is shown in Fig. 1.

An event *is a time interval* having a start state s_1 and an end state s_2. An event either describes a *constancy holding between* s_1 *and* s_2, e.g., like *staying at home*, or a *certain change that happened between* s_1 *and* s_2, e.g. *going from home to work.* The former events are called *homogeneous events*. Such an event has the property that the described constancy does not only hold for the whole event, but also for all its subevents. Moreover, such events shall often be of *maximum length*, i.e., there shall be no proper subintervals which also satisfy the event property. In contrast, the events describing changes are often called *Gestalt events*, if they may *not* have subevents for which the property also holds. Thus, such events shall be of *minimum length*. We will show how these requirements (whose modeling would require an *until* modality in temporal modal logics) can be formalized in nRQL.

We distinguish *generic or non-thematic events* and *thematic events*. A thematic event requires background knowledge (e.g., social reasoning) in order to be recognized. For example, the event *staying in a region* is a (homogeneous) generic event, whereas the *staying at home* event is a thematic event. An additional discriminator is given by the distinction of *simple vs. complex events*, as illustrated in Fig. 2. Simple events have no subevents, whereas complex events have. The required relationships of the subevents to one another are specified with the help of Allen relations.

Events describing constancies can only be recognized if states are also automatically generated even if the situation description has *not changed*, but simply time has gone by. Thus, if a significant change of the value of the time attribute is considered as a relevant change in the situation description, then a new state will be generated automatically, so events describing constancies can be recognized from the similarity (non change) of attributes between situations. However, this also reveals the question of *how frequent* new states shall be constructed. We do not answer this question here.

Given the structure of the history ABoxes, the next important question to ask is: How to recognize the events? As a first idea, we can try to identify events with their *start states* and then exploit the temporal structure spawned by the *next* and *future* relationships. Thus, a *leaving home event* could be recognized with the following concept definition:

```
(define-concept leaving-home-event
     (and state (some in-region home) (some next (all in-region (not home)))))
```

However, due to the Open World Semantics [3, pp. 68] employed by DLs, we see that `(all in-region (not home))` can only be proven if appropriate closure assertions are added on the `next` successor's `in-region` role. Moreover, it does not seem to be adequate to identify events which have a certain duration and are thus conceptually intervals with their start states which are conceptually time points. Also, there is no way to access or refer to the duration of such an event, since role quantification on `next` and `future` can only *see* the require future states, but cannot *fix* them. Thus, *variables* are needed. Moreover, while / until operators known from temporal modal logics would be needed in order to express that an event has maximum or minimum length. Also, a concept such as `home` depends on the agent and thus cannot be used if more than one agent individual is present. Thus, we have to verify that the region is indeed the home *of the agent*.

We are thus defining events with the help of rules again. Events are instances of an event concept and reference their start and end states with the roles `start-state` and `end-state`. In case of a complex event, the subevents are aggregated using the `has-subevent` role. These events thus satisfy

```
(define-concept event
                (and interval (some has-agent agent)
                     (some start-state state) (some end-state state)))
```

Event rules have to construct *new individuals*. So-called *DL-safe rules* are rules whose variables only range over ABox individuals, i.e., all variables are *distinguished*. This is always the case in nRQL. However, since nRQL allows the creation of new individuals with rules we need to be careful, since rules may be applied to freshly created individuals as well. In order to avoid termination problems, nRQL does not offer an automatic rule application strategy; instead, API functions function are supplied to first identify the applicable rules, and then to fire (all or some of) them. This is called a single *rule application cycle*. In principle it is unclear how many cycles will be needed. Thus, the application runs a loop. To ensure termination, we make the antecedences of the rules *non-monotonic* such that a rule can only be fired once for a certain set of input individuals. The *general pattern / idiom for simple event rules* thus looks as follows:

```
(prepare-abox-rule
     (and (?s1 state) (?s2 state)
          (?s1 ?a has-agent) (?s2 ?a has-agent)
          (?s1 ?s2 next) // or future or past or previous
          ... // some more conditions on the states
          // ensure that rule can only be fired "once":
          (neg (project-to (?s1 ?s2)
                   (and (?e some-simple-event)
                        (?e ?s1 start-state) (?e ?s2 end-state)))))
   ((instance (new-ind new-simple-event ?s1 ?s2) some-simple-event)
    (related  (new-ind new-simple-event ?s1 ?s2) ?s1 start-state)
    (related  (new-ind new-simple-event ?s1 ?s2) ?s2 end-state)))
```

If the antecedence of the rule identified appropriate start and end states `?s1` and `?s2` in the same history (belonging to the same agent `?a`), and such an event has not already been constructed, then a new event instance referencing `?s1, ?s2` is created. The `new-ind` operator is used to construct a new ABox individual; if `?s1` is bound to `s1` and `?s2` to `s2`, then the expression `(new-ind new-simple-event ?s1 ?s2)` creates a new individual `new-simple-event-s1-s2`.

Using this pattern, we can define homogeneous and gestalt *generic simple events.* For example, we have the following spatial events: *leaving a region, entering a region, staying in a region,* and the *in no region event.* By means of *blutooth devices and buddy lists*, it can also be recognized if a buddy is close by. We thus also have the *meeting buddy, leaving buddy, staying in company of a buddy* as well as the *being alone event.*

It is obvious that the *leaving and entering a region events* are easy to model with ABox rules as follows: the general event rule pattern is used, but additional constraints on the states are imposed, for example, `(?s1 ?r in-region)` and `(neg (?s2 ?r in-region))` in case of the leaving a region event, and vice versa for the entering a region event. Maximum duration and homogeneity of events are harder to enforce. Let us consider the `staying-in-region` event. Assume that `?r` is the region in which the agent is currently staying. To enforce maximum duration of the interval to the left, we require that `?s1` does not have a `previous` state which is also contained in `?r`, and similar for `?s2` and `next`. Homogeneity can be expressed as well – between `?s1` and `?s2` there shall be no states `?s3` in which `(?s3 ?r in-region)` does *not* hold. This gives us the additional conjuncts:

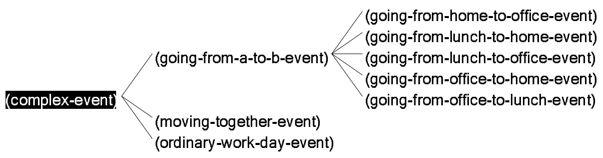
(complex-event)
(going-from-a-to-b-event)
(going-from-home-to-office-event)
(going-from-lunch-to-home-event)
(going-from-lunch-to-office-event)
(going-from-office-to-home-event)
(going-from-office-to-lunch-event)
(moving-together-event)
(ordinary-work-day-event)

Fig. 2. Taxonomy of Complex Thematic Events

```
(and (neg (project-to (?s3 ?r)
              (and (?s3 ?s4 next) (?s4 ?r in-region))))
     (neg (project-to (?s2 ?r)
              (and (?s12 ?s2 next) (?s12 ?r in-region))))
     (neg (project-to (?s2 ?s3 ?r)
              (and (?s2 ?sx future) (?sx ?s3 future)
                   (neg (?sx ?r in-region))))))
```

The analog social event, *staying in company of a buddy,* is even more complicated, since here one has to relate states of histories of two different agents in order to detect the constancy; note that the `in-buddy-proximity` relation holds between *states* of agents.

Having recognized the simple generic events, we can specialize these to *thematic events,* for example, a `leaving-home-event` is a special `leaving-region-event`. In some cases, simple concept definitions are sufficient for recognition, but in other cases, rules are needed again.

Complex and Very High Level Events We then continue and define *complex event* that consist of several subevents. As with the simple events, we distinguish *generic and thematic complex events.* An important generic complex event is the *going from A to B event*. This event is neither maximal nor homogeneous; instead, it is well known that going from A to B eventually means that one fist has to go from A to C, and then from C to B. Such recursive event rules can become very complex.

So, what are reasonable complex *thematic* events in our case study scenario? Given the *typical working day scenario,* we primarily consider further specializations of the *going from A to B event* which takes the thematic types of the origin and destination regions into account. For example, a *going from office to lunch event* is recognized if the destination region is a restaurant, and if the source region is the work office of the agent. Moreover, such an event has to overlap the *lunch time* individual interval. The introduced complex thematic events are visualized in Fig. 2.

Finally, we can define *very high level complex events*. An *ordinary working day event* is assumed to consist of the following consecutive *sequence of events:* `going-from-home-to-office-event, working-event, going-from-office-to-lunch-event, lunch-event, going-from-lunch-to-office-event, working-event, going-from-office-to-home-event`. If such a sequence of events `?e1` to `?e7` is found, all belonging to the same agent, such that `(?e[n] ?e[n+1] meets)` holds for all `n` from 1 to 6, then a complex event of type `ordinary-working-day-event` is constructed, and the seven subevents are connected to it using the `has-subevent` role.

4 A Complex Example

A complex history ABox is visualized in Fig. 3. The histories of the three agents `A1, A2, A3` are shown. Circles denote states, and containment of a state in a region (the

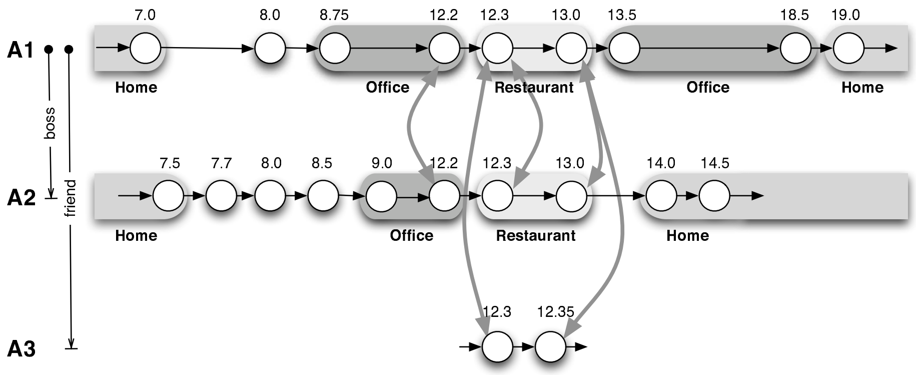

Fig. 3. History ABox for Agents `A1, A2, A3`

`in-region` relationship) is depicted with the help of the state enclosing gray shaded boxes (visualizing the regions). The `home` regions of `A1` and `A2` are different, but the `restaurant` and `office` boxes visualize the same region. The bold gray arrows visualize the `in-buddy-proximity` relationship. The values of the `time` attributes are shown as well (in decimal coding).

The scenario modeled in the example ABox goes as follows: Agent `A2` is the boss of `A1`, and `A3` is a friend of `A1`. `A1` leaves its home at 7.0 and enters his office at 8.75; in the meantime he was *on the road*. He stayed in the office (presumably working) until 12.20 (we are omitting the concrete times in the remaining description). He is then in buddy proximity with his boss. Both are leaving the office together and are entering the restaurant, where they are having lunch. In the restaurant, `A1` meets a friend (`A3`) for a couple of minutes. After staying a while in the restaurant, `A1` and `A2` are leaving the restaurant together. `A1` goes back to office and stays there until he leaves the office in the evening, heading towards home. In contrast, `A2` goes home after lunch.

This history ABox induces a complex event structure. After the rules no more apply, the following complex thematic events have been constructed for `A1`. The following list is the result of a nRQL query; for each binding `?x` to a complex thematic event we are also including its start and end time as well as its *most specific types*. For the events, a `<event-name><start-state><end-state>` naming schema is used, and `<state-number><agent>` for the states:

```
(((?x ordinary-working-day-s1a1-s9a1) (7.0) (19.0)
     (ordinary-working-day-event long-interval))

 ((?x going-from-a-to-b-s1a1-s3a1) (7.0) (8.75)
     (going-from-home-to-office-event region-event short-interval))

 ((?x going-from-a-to-b-s4a1-s5a1) (12.2) (12.3)
     (going-from-office-to-lunch-event moving-together-event region-event))

 ((?x going-from-a-to-b-s6a1-s7a1) (13.0) (13.5)
     (going-from-lunch-to-office-event region-event))

 ((?x going-from-a-to-b-s8a1-s9a1) (18.5) (19.0)
     (going-from-office-to-home-event region-event)))
```

Thus, as expected, `A1` has experienced an arbitrary working day. However, this is neither the case for `A2` nor for `A3`. Note that some more events have been recognized, but they are not “complex”, e.g., the events **`staying-in-region-of-office-of-a1-s3a1-s4a1`** from

8.75 to 12.2 of type `(working-event long-interval)`, `in-company-a3-s5a1-s6a1` from 12.3 to 13.0 of type `(in-company-event meeting-friend-event))` (which is a "social event" due to the bluetooth proximity with friend A3), and `staying-in-region-of-restaurant-1-s5a1-s6a1` from 12.3 to 13.0 of type `(lunch-with-boss-event)`.

5 Conclusion & Future Work

We have proposed a practical and working event model methodology in the RACERPRO DL system. The long term research goal of this work is to enhance the spatial, temporal and dynamic awareness of the ContextWatcher application framework. The principle feasibility of the approach has been demonstrated with a case study. A drawback of the proposed model is the slightly non-declarative semantics shown by some rules, especially those that create new individuals. Recently it has been shown that *abduction* – which is a non-deductive inference process – has the potential to deliver hypotheses and can thus also be used to hypothesize the assertions which we have constructed simply by means of rules [8,9]. How to apply this abduction framework is future work.

It should be stressed that the proposed model only works with RACERPRO, since current W3C Semantic Web standards (OWL, SPARQL, SWRL etc.) do not offer the required expressivity for the formulation of rules (e.g., negation as failure, closed domain universal quantification, creation of new individuals, concrete domain reasoning). It is clear that corresponding concept constructors easily lead to undecidability. But *pragmatic solutions* have to be developed for practical applications, as we have demonstrated.

References

1. Koolwaaij, J., Tarlano, A., Luther, M., et al.: ContextWatcher – Sharing Context Information in Everyday Life. In: Proc. of the IASTED Int. Conference on Web Technologies, Applications, and Services (WTAS'06). (2006)
2. Luther, M., Fukazawa, Y., Wagner, M., Kurakake, S.: Situational Reasoning for Task-oriented Mobile Service Recommendation. The Knowledge Engineering Review **Special Issue on Contexts and Ontologies: Theory, Practice and Applications** (2007) To Appear.
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook – Theory, Implemenation and Applications. Cambridge University Press (2003)
4. Wessel, M., Möller, R.: A High Performance Semantic Web Query Answering Engine. In: Proc. Int. Workshop on Description Logics (DL '05). (2005)
5. McCarthy, J., Hayes, P.J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Meltzer, B., Michie, D., eds.: Machine Intelligence 4. Edinburgh University Press (1969) 463–502
6. Kowalski, R., Sergot, M.: A Logic-based Calculus of Events. New Gen. Computing **4** (1986)
7. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. Communications of the ACM **26** (1983) 832–843
8. Espinosa, S., Kaya, A., Melzer, S., Möller, R., Wessel, M.: Towards a Foundation for Knowledge Management: Multimedia Interpretation as Abduction. In: Proc. Int. Workshop on Description Logics (DL '07). (2007)
9. Möller, R., Neumann, B.: Ontology-based Reasoning Techniques for Multi-Media Interpreation and Retrieval. In Hobbson, P., Kompatsiaris, Y., eds.: Semantic Multi-Media and Ontolgies: Theory and Applications. (2007) To appear.