What Happened to Bob?

Semantic Data Mining of Context Histories

Michael Wessel[1], Marko Luther[2] and Ralf Möller[3]

[1] Racer Systems GmbH & Co. KG, Blumenau 50, 22089 Hamburg, Germany
[2] DOCOMO Euro-Labs, Landsbergerstr. 312, 80687 Munich, Germany
[3] Hamburg University of Technology (TUHH), Institute for Software Systems (STS), Harburger Schloßstr. 20, 21079 Hamburg, Germany

1 Introduction

Modern Social Web applications like Facebook and Twitter allow people to socialize over a distance by sharing notes, photos and other personal information with their buddies or the public. By automating the data gathering via the mobile phone, IYOUIT,[4] a community service in the field of context-awareness, is realizing a sophisticated life logging platform [1]. The mobile part of IYOUIT makes use of the fast data connection and the multitude of sensors (e.g., WLAN, GPS, accelerometer,...) available on modern handsets to transmit recorded context elements to components in the network for further processing. These components combine, abstract, interpret, store and redistribute context streams by making use of external data sources (e.g., mapping GPS coordinates to locations or weather informations) and internal computations (e.g., detecting important places of a person by applying clustering algorithms to location traces). Eventually, context data is transformed into status updates, like *listening to music* or *just arrived home*, which are distributed via the mobile application or any of the connected Web 2.0 services like Facebook. In addition, context is used to automatically tag media data, e.g. photos on Flickr, to ease the search for specific items like *photos taken in Italy*.

To increase the level of abstraction and to derive additional information that is not given explicitly, IYOUIT uses knowledge formalized as OWL DL ontologies to classify qualitative context w.r.t. situation concepts [2]. For example, a *business meeting* may be derived based on the people in proximity, their social relations and the actual place. While the derived (static) situations allow for the generation of more meaningful status updates, the navigation in context histories requires a more complex event model based on situation changes.

In a previous work, we exploited simple ontology-based event recognition with DLs [3]. The work reported here improves this initial approach by providing additional pragmatic solutions for DL-based event recognition, solving some of the scalability issues. Besides formalizing the event recognition problem, we reflect on a set of modeling alternatives and give some practical guidances. Although the method has been implemented using RACERPRO, the identified problems generalize to all DL and OWL systems.

We are assuming familiarity with basic DL notions, syntax, semantics, standard DL reasoning services, as well as with DL and Semantic Web query languages [4]. Those notions will be used without formally introducing them.

[4] `www.iyouit.eu`

2 Situational Reasoning

Previous work introduced the notion *situation* [2] as a vector of context attribute-value pairs describing the circumstances of a person: $(CA_1 : CV_1, \ldots, CA_n : CV_n)$, where CA stands for context attribute, and CV for context value concept. A situation was represented as an ABox individual sit and an assertion such as

$$sit : (\exists CA_1.CV_1) \sqcap \cdots \sqcap (\exists CA_n.CV_n)$$

There is one OWL ontology for each CA, structuring the possible CVs in a taxonomy, e.g., for $CA = at\_place$, we might have $CV \in \{home, office, restaurant, \ldots\}$ (see [5] for information on how location concepts are detected and realized). CAs are not necessarily functional (e.g., the $near\_by\ CA$).

In order to *recognize a situation*, defined concepts are exploited, e.g., a *business meeting* could be detected with defined concepts such as $business\_meeting \doteq\equiv \geq_3 near\_by.colleague \sqcap \exists at\_place.office$. In the following, such concepts will be called *recognizer concepts*. Whereas for simple situation descriptions as the ones just given also the *concept classification service* would be sufficient, in general a "more explicit" representation of the situation as an ABox of the form

$\{sit : situation, (sit, val_1) : CA_1, val_1 : CV_1, \ldots, (sit, val_n) : CA_n, val_n : CV_n\}$

is preferred, since this allows us to exploit an additional, and, concerning the *relational expressivity*, much more powerful recognition mechanism, namely *queries*. For example, a standard *grounded conjunctive query* [6–9] can detect those business meetings in which at least three people are involved which mutually dislike each other:

$$ans(x) \leftarrow business\_meeting(x), near\_by(x, y), near\_by(y, z), near\_by(z, x), \\ hates(x, y), hates(y, z), hates(z, x).$$

In the following, such a query (rule) is called a *recognizer query (rule)*. It is well-known that it is not possible to recognize such situations by means of concepts in standard DLs or OWL, due to a lack of relational expressivity. Such recognizer queries (rules) can be written in SPARQL, SWRL, SWRQL, ... We are using NRQL [9].

In a NRQL *rule*, the ans head predicate is replaced by a so-called generalized ABox which is an ABox whose assertions may reference variables from the body of the rule. Using first-order notation for such generalized ABoxes, the rule conclusion $hateful\_business\_meeting(x)$ would add the concept assertion $sit : hateful\_business\_meeting$ given the binding $x = sit$ satisfies the query / rule body. Similar things can be done in SWRL or SPARQL (using `construct`). But unlike SWRL or SPARQL, NRQL also allows to introduce *new ABox individuals*. This feature will be exploited in the following. Note that NRQL goes beyond the expressivity of SPARQL and SWRQL by implementing *epistemic first-order queries, not only grounded conjunctive queries*. Due to its unique combination of *negation as failure (NAF)* and the *projection operator*, a universal closed domain quantifier is available.

Within the IYOUIT framework, we call the component which creates per day an ABox representing the accumulated context data the *Day Description Generator (DDG)*, and the generated ABox the *Day Description ABox (DDA)*. The DDG is an external program outside of the DL system.

Whereas in many cases, the value concept is directly asserted via $val_i : CV$, sometimes ABox realization is used to also recognize the CV's. To faciliate this, the DDA not only contains situation descriptions, but also user profile data, e.g., the social net-

work of a user, as well as other mostly static information, e.g. home country, typical office hours, etc. With the help of queries, this additional data can be exploited. These modeling ideas are described in more detail in [2], and [10] uses the corresponding OWL ontologies for benchmarking purposes.

3 Event Recognition with Description Logics

In [3], the focus was shifted from recognizing static situations which are conceptually instantaneous "snapshots in space-time" to so-called *events* which also take the dynamic and temporal aspects of situational changes into account. Events have a temporal duration. Recognized structures can be exploited for the creation of static diary-like day summaries (Blogs), but also for querying and data mining purposes). As such, the famous *Allen temporal relations* [11] play an important role, not only for query formulation and natural language generation for Blogs, but also as *the* vocabulary defining *event recognizers*. Events are also *context-providing*. By enabling IYOUIT to recognize *situational changes* rather than just static *situations*, a deeper level of context awareness can be achieved. For example, an *ordinary office day* event could be defined as a *sequence of consecutive events*, which stand in the Allen relation "meets": $\langle$at home, moving, in office, moving, in restaurant, moving, in office, moving, at home$\rangle$, together with some additional restrictions regarding certain fixed day time intervals, such as "in restaurant *during* noon", etc. Thus, the DDA also contains these day time intervals. An ordinary office day is therefore a complex event aggregating a series of subevents and should thus be recognizable from the types of its subevents and from the temporal relations holding between them.

The challenging questions, then, are: *a) how to represent such events and b), how to define "event recognizers" for them (either as defined concepts or as recognition queries / rules).* A first case study was perfomed in [3]. Both answers to these questions have to take into account the side conditions that existing standard DL or OWL reasoners (such as RACERPRO) shall be used in order to get a working system with good performance *today*.

Regarding a), the least "ontological commitment" we can make is saying that we want to represent actual events in the ABox which satisfy the following axiom:

$$event \doteq \exists start\_state.state \sqcap \exists end\_state.state$$

We simply state that an event is an "aggregate" which has a start state and an end state, and that $state$s are basically the $situation$s just described, but augmented with some *temporal information* which allows to determine the temporal order between states. We assume a linear discrete time model, e.g. $(\mathbb{N}, <)$. There are various options for the representation of the temporal relations between the states; these options will be discussed in the next section. It is also assumed $start\_state < end\_state$ holds, for all events. In addition, *complex events* are composed of (one or several) subevents:

$$complex\_event \doteq event \sqcap \exists has\_part.event.$$

Non-complex events are called *simple events*. Two important subclasses are $change\_events$ and $const\_events$: $change\_event \dot{\sqsubseteq} event$, $const\_event \dot{\sqsubseteq} event$. The former *witnesses a change of some* CA from its $start\_state$ to its end_state. The latter *witnesses the constancy (non-change) of some* CA, e.g., if $CA = at\_place$ changes from $CV = home$ to $CV = office$, then this change will be reflected by a

$leaving\_home\_entering\_office$ event. The end_state should be the immediate successor w.r.t. the temporal order $<$. Hence, a basic *change event* should have a *minimal temporal duration.* In contrast, a *constancy event* should have *maximal duration,* and must also be *homogeneous* [12], i.e., there is no state in between the start and end state for which the $CA \times CV$ attribute value pair does *not* hold. An example is the $staying\_in\_the\_office$ event, which is a constant (and thus maximal and homogeneous) event. These properties are very important, since otherwise the definition of more complex events based on the basis of subevents becomes infeasible if one never knows in how many "segments" a subevent is splitted.

By exploiting the epistemic first-order properties of NRQL, *maximal and homogeneous events* for some $some\_condition$ are recognized by the following query (rule); we assume that $future$ holds between s_1 and s_2 iff s_1 preceeds s_2 on the time line, i.e. $s_1 < s_2$, and $next$ between s_1 and s_2 iff s_2 is the direct successor of s_1 w.r.t. $<$:

$$\begin{aligned} ans(s_1, s_2) \leftarrow\ & state(s_1), state(s_2), future(s_1, s_2), \\ & some\_condition(s_1), some\_condition(s_2), \\ & \backslash\pi(s_1)\ (state(s_0), next(s_0, s_1), some\_condition(s_0)), \\ & \backslash\pi(s_2)\ (state(s_3), next(s_2, s_3), some\_condition(s_3)), \\ & \backslash\pi(s_1, s_2)\ (\ state(s_3), future(s_1, s_3), future(s_3, s_2), \\ & \qquad \backslash some\_condition(s_3)) \end{aligned}$$

The "$\backslash$" is the NAF operator, and "π" is the projection operator. Note that $\backslash\pi \ldots$ implements a first-order epistemic closed-domain quantifier. The first $\backslash\pi \ldots$ ensure maximality of the interval to the left, the second occurence maximality of the interval to the right, and the third occurence verifies homogeneity, i.e., there is no state s_3 in between s_1 and s_2 for which we *cannot* prove $some\_condition$; hence, $\backslash some\_condition(s_3)$ shall not hold. Note that $\neg some\_condition(s_3)$ is a too strong requirement, since absence of $some\_condition$ on s_3 does not imply that $\neg some\_condition(s_3)$ holds; rather, only $\backslash some\_condition(s_3)$ holds. We refer to [9, 13] for more details.

The semantics can be paraphrased as a first-order query evaluated over the relational structure ("database") $\mathcal{S}_\mathcal{A}$=$(\Delta^\mathcal{I}, C^\mathcal{I}, \ldots, R^\mathcal{I}, \ldots)$, with $\Delta^\mathcal{I} =_{def} \mathsf{inds}(\mathcal{A})$, $C^\mathcal{I} =_{def} \{\, i \,|\, i \in \mathsf{inds}(\mathcal{A}), \mathcal{A} \models C(i) \,\}$, $R^\mathcal{I} =_{def} \{\, (i,j) \,|\, \mathsf{inds}(\mathcal{A}), \mathcal{A} \models R(i,j) \,\}$, for all relevant roles R and all relevant (not only atomic!) concepts C. The first-order query evaluated over $\mathcal{S}_\mathcal{A}$ gives the semantics of the NRQL query above (by construction of $\mathcal{S}_\mathcal{A}$, NAF negation "$\backslash$" can be replaced by classical negation, "$\neg$"):

$$\begin{aligned} \{\, (s_1, s_2) \,|\, \exists s_1, s_2 :\ & state(s_1) \wedge state(s_1) \wedge future(s_1, s_2) \wedge \\ & some\_condition(s_1) \wedge some\_condition(s_2) \wedge \\ & \neg\exists s_0 :\ state(s_0) \wedge next(s_0, s_1) \wedge \\ & \qquad some\_condition(s_0) \wedge \\ & \neg\exists s_3 :\ state(s_3) \wedge next(s_2, s_3) \wedge \\ & \qquad some\_condition(s_3) \wedge \\ & \neg\exists s_3 :\ state(s_3) \wedge future(s_1, s_3), future(s_3, s_2) \wedge \\ & \qquad \neg some\_condition(s_3) \,\} \end{aligned}$$

However, the most important question is – *Where do events come from?* Events can be recognized by recognizer concepts or queries / rules. Recognizer concepts rely on the ABox individual realization service and thus on *explicit* ABox assertions. However, the situation is different for queries and rules. First of all, a NRQL rule can create new ABox assertions, or even new individuals. These new assertions can trigger further rule applications. Moreover, NRQL is not depending on *explicit relations* in forms of

concept or role assertions, but also works on *implicit relations*, since NRQL offers the equivalent of intensional database relations similar to *Datalog rules*. These implicit relations are defined by means of so-called *defined queries*. For example, the binary recognizer rule for the Allen relation $before$:

$before(x, y) \leftarrow event(x), event(y), end\_state(x, s_1), start\_state(y, s_2), future(s_1, s_2)$

can either be understood as a rule which adds ("materializes") $before$ role assertions to the ABox, or as a *definition of the query* $before$ (a "query macro") which is expanded and replaced by its definition *whenever it is referenced in some other query body*, e.g., in an event recognizer rule or query.[5] Hence, a query or rule body referencing $before$ does not necessarily require explicit $before$ role assertions in the ABox, but can work with these *implicit or intensional* $before$ *relations*. As a result, the bodies of the referencing rules can get very complex and become demanding for the query optimizer and processor (an exponential blowup is possible). Moreover, defined queries must be acyclic in NRQL. If recursion is required, NRQL rules have to be used instead of defined queries.

In principle it is possible to define complex event recognizers as defined queries rather than rules. But, as there would be no ABox individual representing the complex event itself, these queries would be n-ary predicates, or binary predicates such as $ordinary\_office\_day(s_1, s_2)$ on the states s_1 and s_2. The n-ary solution is obviously bad, and the binary solution has the drawback that it would be impossible to refer to its subevents, only the states are available. This is strictly required since otherwise no Allen relations to other events could be computed. Consequently, also the simple events should be modelled as binary predicates. Thus, the approach requires an entirly different modeling and is thus not considere further in this paper (even Allen relations could no longer be understood as roles holding between events, but would become quadrary defined queries for pairs of endpoints of events).

Conclusion: In principle, the Allen relation recognizers as well as the event recognizers can be rules or defined queries. The latter option is rejected for event recognizers since we want explicit visibility of the events in the ABox. Also, the modeling would be quite different. Allen relations as defined queries were already investigated in [3] where we have observed a rather bad performance, which was partly caused by repeated recomputation of Allen relations, and partly due to the query / rule body blowup caused by unfolding of the Allen definitions. Hence, this time we are materializing Allen relations via rules at the price that the ABoxes get bigger. We evaluate this decission in Section 3. But first we want to shed some light on the question "Where do events come from?" in order to provide guidance for developers of similar or related DL-based event recognition systems.

First Idea - Pre-Construction of all Events by the DDG The set of states of a given day is finite. Is it possible for the DDG to pre-construct all possible events in the DDA in advance, and then rely as much as possible on the *standard ABox individual realization service* to recognize events, similar to the situational reasoning [2]? For n states, there are most $m_0 = n(n-1)/2$ simple events. Hence, there are at most $m_1 = \sum_{2 \leq k \leq m_0} \binom{m_0}{k} = 2^{m_0} - m_0 - 1$ complex events which can be constructed from these m_0 simple events. These events can in turn become subevents of more complex events, and so on. For 3 states, we already get 120 events for $m_0 + m_1 + m_2$,

[5] This does not work in concepts, e.g., $\exists before \ldots$ would not be aware of the defined query.

and 1329227995784915872903807060280344455 events on the next level. The set of pre-constructable events is in fact infinite, but it may be possible to compute an upper bound based on the definitions of the recognizers. Although it seems reasonable to stop at, say, level 3, still the constructed ABoxes are not managable. In the IYOUIT scenario, a more realistic assumption is that of approx. 30 states per day and user. Already level 1 becomes infeasible then. Even worse, an additional m_i^2 number of Allen relations have to be asserted.

Since all relevant event aggregates are already explicitly present in the ABox, recognizer concepts become possible:

$$\begin{aligned} leaving\_home\_entering\_office \doteq\ & \exists start\_state.\exists at\_place.home \sqcap \\ & \exists end\_state.\exists at\_place.office \end{aligned}$$

Unfortunately, certain important event properties, such as maximality and homogeneity, cannot be expressed with concepts. But at least for the basic context attribute values which are explicitly asserted (which require no reasoning), these properties can be taken care of by the DDG (which does not perform reasoning and so has *in general* no knowledge about the temporal extent of some $CA \times CV$ property holding), and assure or directly assert that kind of information.

In most cases, recognizer concepts are insufficient for the representation of complex events. The main limitation is the inability of concepts to describe anything else than tree structures (regarding the role successors). For example, it is impossible to specify that *during* the in_office_event, first the $meeting\_with\_boss$, and immediately after (= $meets$ relation), but still during the in_office_event, the $meeting\_with\_customer\_event$ occurred. An attempt:

$$\begin{aligned} & stressful\_office\_day \doteq \\ & \quad \exists has\_part.(\ in\_office\_event \sqcap \\ & \qquad \exists during.(\ meeting\_with\_boss\_event \sqcap \\ & \qquad\quad \exists meets.meeting\_with\_customer\_event \sqcap \ldots)) \end{aligned}$$

However, with that definition it cannot be taken for granted that the $meeting\_with\_customer\_event$ still takes place during the in_office_event. Other attempts to define such a concept will have similar defects. Although it is impossible to fix the start state and end state by means of existentials, the temporal duration of the complex event as well as the Allen relations to other (possibly complex) events are in fact known, since the aggregate was constructed by the DDG which asserted the correct corresponding $start\_state$ and end_state successors. This is also the motivation for the has_part role. Without the additional individual representing the complex event as an aggregate, start and end state of the aggregate could not be fixed, and consequently, Allen relations could not be computed. Moreover, one of its subevents would have to act as a proxy for the whole aggregate [14], what seems inadequate in this scenario.

Although many temporal constellations between subevents cannot be reliably recognized with defined concepts since a lot of *false positives* will be detected, some complex events can indeed be realized in that way, e.g., *those complex events for whose description a tree-like temporal Allen network is sufficient.* Another required provision in order to minimize the amount of false positives is that one has to *restrict the visibility of Allen relation successors to those events which are part of the same aggregate* – otherwise the traversal of an Allen role assertion R via $\exists R. \ldots$ could lead into an event being subevent of some *other* complex event, yielding another false positive. However,

this forces the DDG to create *copies* of subnetworks and increases the ABox size by an additional factor. Hence, the approach does not seem reasonable.

Again, a query or rule can help overcome the expressivity problems:

$$\begin{aligned} \mathit{stressful\_office\_day}(x) \leftarrow\ & \mathit{has\_part}(x,p_1), \mathit{has\_part}(x,p_2), \mathit{has\_part}(x,p_3), \\ & \mathit{in\_office\_event}(p_1), \mathit{meeting\_with\_boss}(p_2), \\ & \mathit{meeting\_with\_customer\_event}(p_3), \\ & \mathit{meets}(p_2,p_3), \mathit{during}(p_1,p_2), \mathit{during}(p_1,p_3) \end{aligned}$$

Conclusion: For complex events, the pre-construction by the DDG causes two problems. First, the number of precomputed events and Allen relations gets enormous, and second, it is not always possible to define resonable expressive recognizer concepts for complex events. The latter point is not resolvable unless specialized temporal logics are used, or certain OWL extensions are accepted [15]. Only queries or rules can help then. For simple events, some recognizer concepts can be defined, which is a big advantage, since rule / query management in DL systems has not yet reached the same level of matuarity and state of the art as concept management. But even for simple events, certain event properties can only be adressed using the first-order facilities of nRQL. *In principle, DL-safe rule languages are sufficent, since all events have been pre-constructed.*

Second Idea - Start from States, Construct Events with Non-Safe Rules Whereas the one extreme was just described (all events are pre-constructed by the DDG), the other extreme is to have no pre-constructed events at all in the DDA, but only states with their temporal relations.

As a consequence, the number of recognizer rules gets very large, and rule management becomes an important issue. A big plus of the nRQL rules are the first-order capabilities in this settings. We already demonstrated how to recognize maximal and homogeneous events. Since only states together with their temporal relations are present in the DDA, non-safe rules which can introduce new event individuals have to be used (we have rejected the option to implement these recognizers as defined queries above). nRQL offers the $\mathit{new\_ind}$ operator (which can be understood as a function symbol). A typical basic event recognizer / constructor rule constructing $\mathit{event\_concept}$ instances looks as follows:

$$\begin{aligned} & \mathit{event\_concept}(\mathit{new\_ind}(\mathit{event},s_1,s_2)), \\ & \mathit{start\_state}(\mathit{new\_ind}(\mathit{event},s_1,s_2),s_1), \\ & \quad \mathit{end\_state}(\mathit{new\_ind}(\mathit{event},s_1,s_2),s_2) \leftarrow \mathit{state}(s_1), \mathit{state}(s_2), \\ & \qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{future}(s_1,s_2), \text{ (or } \mathit{next}\text{)} \\ & \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{plus some more conditions on } s_1,s_2 \end{aligned}$$

For example, given states $s_1 = i, s_2 = j$, it constructs the ABox assertions $\{\mathit{event}_{i,j} : \mathit{event\_concept}, (\mathit{event}_{i,j}, i) : \mathit{start\_state}, (\mathit{event}_{i,j}, j) : \mathit{end\_state}\}$. Non-safe rules are in principle subject to termination problems, since new individuals are introduced (on which rules may fire which introduce individuals, on which rules may fire which introduce individuals, on which ...). In nRQL, this termination problem is delegated to the client program which has to drive the rule application, since nRQL does not offer an automatic rule application strategy. The client program thus has to run a loop of the form "`while (applicable-rules()) {execute-applicable-rules();}`" calling the nRQL rule API functions to drive the rule application. One iteration of

this loop is called a *rule application cycle.* Thus, all possible termination problems are caused by the client which runs the loop. Due to the non-monotonic features, it is possible to write NRQL rules for which the loop actually terminates (this would not be the case in monotonic rule languages!). For example, the above rule can disable itself after all constructible events were constructed by adding to its precondition

$$\backslash\pi(s_1, s_2)\ (event\_concept(x), start\_state(x, s_1), end\_state(x, s_2))$$

The need for an external client program driving the rule application can be seen as a drawback. However, recently the MINILISP functional expression language for server-side programming has been added and reached a mature state [16]. It is thus possible to implement a rule application strategy directly on the RACERPRO server. Since MINILISP is termination safe it does not allow to implement unbounded loops. However, it allows bounded loops and structure traversion. In most cases, a fixed upper limit on the number of required rule application cycles can be computed, e.g. by analyzing the rules and the number of present states.

The *recognizer rules for complex events* look very similar to the $stressful\_office\_day$ example rule already discussed; the conclusion, however, has to construct a new event individual and also cannot rely on $has\_part(x, p)$ for its parts p in the precondition, of course. Please note that also the Allen relations have to be computed for each constructed event. The universal closed-domain quantifier of NRQL (the "$\backslash\pi \ldots$"-construction) is also important for complex event recognizer rules, since maximality and more often *absence of certain other events, i.e., between two subevents* has to be enforced.

From a theoretical perspective, this approach is the cleanest and most powerful one. Only the events which are really detected are constructed (in contrast, in the previous approach the DDG has no knowledge about what is really needed and constructs all kinds of unnecessary events). Therefore, this approach was choosen in [3]. As explained above, the bad performance observed was partly caused by Allen relations as defined queries. However, also the big amount of rules required leaves little place for recognizer concepts. Having to construct and check homogenity and maximality even for simple events introduces a big number of universal closed-domain quantifications which are expensive too evaluate. We thus argue that the DDG should already construct the basic events and take care of these properties whenever it is capabable to do so, thus leaving only the hard recognition problems for the reasoner.

Conclusion: Constructing events with non-safe rules is conceptually very clean. The whole process is driven solely by the ontology (considering rules part of the ontology). But, it has performance problems with the current state-of-the-art technology. This does not invalidate the approach as it might run fast on future systems, however, we are looking for a system working with today's technology. The number of rules becomes very large and thus issues which are curently not well-supported by DL systems, e.g., rule managment, become predominant. Allen relations can no longer be computed in advance by the DDG, but rather have to be recomputed after each rule application cycle.

Third Idea - The Best of Both Worlds Having analyzed both extremes, we proceed as follows: The DDG already pre-constructs all relevant simple events, and only the complex events and simple events whose recognition requires ontology reasoning are

constructed via rules. Some Allen relations can in principle already be asserted by the DDG for those pre-constructed events. Nevertheless we must be able to compute Allen relations after each rule application cycle.

Hence, for every s_1, s_2, CA and CV, the DDG pre-constructs a simple event if and only if $time(s_1) < time(s_2)$, the event is homogeneous, and has a maximal extent w.r.t. the $CA \times CV$ attribute-value pair. In case the recognition of $CA \times CV$ requires reasoning, the event cannot be pre-constructed, and recognition rules (or concepts) have to be written. In the current IYOUIT scenario, recognizer rules are only required for complex events. The number of simple events is bounded by $|CA||CV|n(n-1)/2$, where n is the number of states. Since all simple events in the DDA are now maximal and homogeneous, it also becomes reasonable to attach the CA attributes to the event individuals instead of the states. For every possible $CA \times CV$ pair, a simple event recognizer concept can be defined:

$$event_{CA,CV} \doteq event \sqcap \exists CA.CV\,.$$

Consequently, the CV taxonomy is automatically inherited to the taxonomy of simple events (something we would not get automatically with recognizer rules, although NRQL is able to compute query and rule subsumption [13]). These auxilary concepts faciliate the modeling of complex event recognizer rules.

The DDG did not pre-construct change events. Their absence is not really a drawback; they can still be recognized, e.g. the change of CA from CV_1 to CV_2 can be detected with a rule such as

$$\ldots \leftarrow event_{CA,CV_1}(e_1), event_{CA,CV_2}(e_2), meets(e_1, e_2)$$

However, in most cases changes only need to be detected in order to define complex rules, and thus, these change detections can simply become part of the definitions of the complex event recognizer rules. Still, some code is needed to drive the rule application. The remarks from the previous paragraph apply (we are using MINILISP this time).

Conclusion: The idea seems promissing and combines the best of both worlds. It keeps the number of rules maintainable, allows to employ defined concepts as recognizer for simple events and does not lose expressivity. As for the previous approach, the efficent computation of Allen relations is very important.

4 Computation of Allen Relations

Since the set of states is fixed, the DDG can precompute the temporal order between any two states by means of role assertions using the roles $next$, $equal$ (with $next \doteq prev^{-1}$), and a transitive superrole $future$, $next \doteq future$, $future^+ \doteq future$, $past \doteq future^{-1}$. Note that the explicit relation representation in terms of $next$ and $equal$ requires $n + n(n-1)/2$ role assertions (and even more if also $prev$, $future$ and $past$ are made explicit in case one wanted to get rid of the TBox axioms for the roles). Alternatively, the DDG can attach a filler of the *concrete domain (CD) attribute* $time$ to each state. A NRQL rule can make the $next$ relation explicit, or a defined query could be used:

$$next(s_1, s_2) \leftarrow < (time(s_1), time(s_2)),$$
$$\backslash\pi(s_1, s_2)\ (< (time(s_1), time(s_3)), < (time(s_3), time(s_2)))$$

The $<$-atom is a so-called *CD constraint checking atom*. Whereas the evaluation of this atom requires a concrete domain unsatisfiabilty check which may be expensive, there is also a cheapter option in NRQL, a so-called *data substrate query* which can

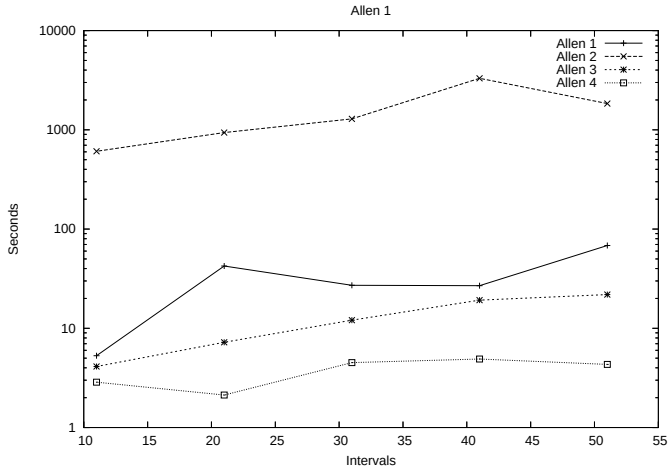

Fig. 1. Benchmark Results

check such constraints on a set of *data literals* much faster, since model checking of some data substrate structure is sufficent [17].

In case $next$ is materialized, we also automatically get the relations $prev$, $future$, and $past$, due to the role declarations. However, if the rule is used as a defined query, then some more definitions are required for the other roles, e.g. for $future$ we can simply remove the "$\backslash\pi\ldots$" line from the $next$ definition. The implicit and intrinsic relations between states keep the ABox small in number of role assertions, they are thus worth considering, and should be subject to a benchmark. On the other hand, the explicit relation representations require bigger ABoxes, but have the benefit to also provide index structures for query answering, and moreover can be used to avoid repeated computations of relations. It is not clear how to proceed without an evaluation of the alternatives. Allen relations are computed from the relations between the states of the events, as already explained with the $before$ relation. The ability to *efficiently compute Allen relations* is crucial for the performance and scalability of the whole approach. Thus, for the benchmark, we are first evaluating the performance of the Allen computation. A DDA with 50 random intervals is created, which is reduced to 40 intervals by removing 10 intervals, and so on, until only 10 intervals remain. We are considering four different settings for the benchmark:

Allen 1: explicit state relations as $next$ role assertions, 7 Allen rules (one rule per Allen relation and its converse),
Allen 2: implicit state relations via defined query $next$ and CD atom $<$
Allen 3: implicit state relations via defined query $next$ and data substrate atom $<$
Allen 4: implicit state relations and computation of the Allen relations with *one* rule instead of 7, by means of a MINILISP λ expression which analyzes the $time$ attribute fillers of the states of the events e_1, e_2 and *computes* the corresponding Allen relation programmatically with a conditional expression $expr$:

$$\lambda(e_1, e_2) \bullet expr \leftarrow event(e_1), event(e_2)$$

Figure 1 speaks a clear language (note the logarithmic scale): the procedural solution (Allen 4) is by far the fastest with approx. 600 Allen relations per second (ARPS),

then comes the data substrate solution with 170 ARPS, then the solution with explicit state relations in the ABox with approx. 66 ARPS, and finally the CD solution with 1.8 (!) ARPS. Note that the biggest ABox contains 4225 Allen role assertions. Also, it is important to know how much time is spend for Allen computation if a new event is introduced into the existing network. We thus added 5 random intervals into an existing network of 25 intervals and measured the time required for Allen relation computation: Allen 1 = 12.4 s, Allen 2 = 612 s, Allen 3 = 5.7 s, Allen 4 = 1.5 s. Of course, the bigger the network gets into which the new individuals are inserted, the more time is required. The benchmark results make clear that the definition of the Allen relations as defined queries with the help of the concrete domain in our previous work [3] was a bad decision.

5 Conclusion

While lots of temporal logics have been designed [18–23] which could have been applied in this scenario, few of them have been implemented, less have implementations with good performance, and thus, none of these can be used for practical applications today. In contrast, Description Logic (DL) systems have made tremendous progress. Still, realizing a DL-based event recognition which exhibits a good performance is a highly demanding task. There are no simple answers to most of the modeling questions. With nowadays quite complex DL and Semantic Web technology, there are often various realization and representation options. Even for experts, the consequences of certain design decisions can be very hard to oversee. Without performing benchmarks and evaluations, no solid ground can be reached in applying this technology to real world application problems. We argue that cases studies and papers like this one are important since they conserve and convey a lot of "how to" knowledge which may prevent users from reinventing the wheel and from modeling errors. We thus wrote this paper in a "reflective" style. Although we have used RACERPRO in this study, we argue that the problems and solutions discussed and presented are not specific to RACERPRO.

In future work, alternative (but implemented) approaches should be checked out, e.g., instead of using unsafe rules, recent progress regarding *multimedia and image interpretation with horn rules in an abduction framework* shall be exploited [24, 25] where new individuals and corresponding assertions are abduced rather than constructed. This technology was brought to a mature state in the BOEMIE EU project.[6] Moreover, in the spirit of the classical event recognition system NAOS [26], a new temporal representation and querying engine has been integrated into RACERPRO: The so-called *time net* offers an alternative way to define complex event recognizers. The feasabilty of these options for IYOUIT should be evaluated.

Regarding IYOUIT, there are open challenges. In order to exploit events for the offering of context dependent services on a mobile phone, the events have to be recognized incrementally and online, not offline [27]. As such, the work on incremental plan recognition is relevant [28]. Ideally, this must happen immediatly, e.g. as in [29]. Also there is work in progress regarding the handling of events spanning multiple days, e.g., an oversea business trip. The handling is challinging since events get "split" at day boundaries and have to be re-merged, etc.

[6] `www.boemie.org`

References

1. Böhm, S., Koolwaaij, J., Luther, M., Souville, B., Wagner, M., Wibbels, M.: Introducing IYOUIT. In: Proceedings of the International Semantic Web Conference (ISWC'08), October 27–29, 2008, Karlsruhe, Germany. Volume 5318 of LNCS., Springer Verlag (October 2008) 804–817
2. Luther, M., Fukazawa, Y., Wagner, M., Kurakake, S.: Situational reasoning for task-oriented mobile service recommendation. The Knowledge Engineering Review **23**(1) (March 2008) 7–19 Special Issue on Contexts and Ontologies: Theory, Practice and Applications.
3. Wessel, M., Luther, M., Wagner, M.: The difference a day makes – recognizing important events in daily context logs. In: Proceedings of the CONTEXT'07 Workshop on Contexts and Ontologies. Volume 298., Roskilde, Danmark, CEUR.org (2007)
4. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook – Theory, Implemenation and Applications. Cambridge University Press (2003)
5. Nurmi, P., Koolwaaij, J.: Identifying meaningful locations. In: Proceedings of the 3rd International Conference on Mobile and Ubiquitous Systems (MobiQuitous'06), IEEE Computer Society (July 2006)
6. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Description Logic framework for information integration. In: Proceedings of Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998. (1998) 2–13
7. Möller, R., Wessel, M.: Terminological default reasoning about spatial information: A first step. In: Proceedings of the 1999 International Conference on Spatial Information Theory (COSIT '99), Springer-Verlag (1999)
8. Horrocks, I., Tessaris, S.: Querying the Semantic Web: a formal approach. In Horrocks, I., Hendler, J., eds.: Proc. of the 13th Int. Semantic Web Conf. (ISWC 2002). Number 2342 in Lecture Notes in Computer Science, Springer-Verlag (2002) 177–191
9. Wessel, M., Möller, R.: A High Performance Semantic Web Query Answering Engine. In: Proc. Int. Workshop on Description Logics (DL '05). (2005)
10. Luther, M., Liebig, T., Böhm, S., Noppens, O.: Who the Heck is the Father of Bob? – A Survey of the OWL Reasoning Infrastructure for Expressive Real-World Applications. In: Proceedings of the 6th European Semantic Web Conference (ESWC'09). Volume 5554 of LNCS., Springer, Heidelberg (2009) 66–80 To Appear.
11. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. Communications of the ACM **26**(11) (1983) 832–843
12. Artale, A., Bettini, C., Franconi, E.: Homogeneous concepts in a temporal Description Logic. In: Proceedings of the International Workshop on Description Logics (DL'94). (May 1994) 36–41
13. Haarslev, V., Möller, R., Wessel, M.: RacerPro User's Guide and Reference Manual Version 1.9.0 (May 2005)
14. Haarslev, V., Wessel, M.: GenEd—an editor with generic semantics for formal reasoning about visual notations. In: 1996 IEEE Symposium on Visual Languages, Boulder, Colorado, USA, Sep. 3-6. (1996) 204–211
15. Motik, B., Grau, B.C., Sattler, U.: The representation of structured objects in DLs using Description Graphs. In Baader, F., Lutz, C., Motik, B., eds.: Proceedings of the 21st International Workshop on Description Logics (DL2008). Volume 353 of CEUR Workshop Proceedings., CEUR-WS.org (2008)
16. Kaplunova, A., Möller, R., Wessel, M.: Leveraging the expressivity of grounded conjunctive query languages. In: Proc. International Workshop on Scalable Semantic Web Systems. (2007)

17. Wessel, M., Möller, R.: Flexible software architectures for ontology-based information systems. Journal of Applied Logic, Special Issue: Empirically Successful Computerized Reasoning **7**(1) (2009)
18. Artale, A., Franconi, E.: A temporal description logic for reasoning about actions and plans. J. Artif. Intell. Res. (JAIR) **9** (1998) 463–506
19. Grathwohl, M., de Bertrand de Beuvron, F., Rousselot, F.: A new application for Description Logics: Disaster management. In Lambrix, P., Borgida, A., Lenzerini, M., Möller, R., Patel-Schneider, P.F., eds.: Proceedings of the 1999 International Workshop on Description Logics (DL'99). Volume 22 of CEUR Workshop Proceedings., CEUR-WS.org (1999)
20. Artale, A., Franconi, E.: A survey of temporal extensions of Description Logics. Annals of Mathematics and Artificial Intelligence **30**(1-4) (2001) 171–210
21. Artale, A., Lutz, C.: A correspondence between temporal description logics. Journal of Applied Non-Classical Logics **14**(1-2) (2004) 209–233
22. Artale, A., Parent, C., Spaccapietra, S.: Evolving objects in temporal information systems. Ann. Math. Artif. Intell. **50**(1-2) (2007) 5–38
23. Artale, A., Lutz, C., Toman, D.: A Description Logic of change. In Veloso, M.M., ed.: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07). (2007) 218–223
24. Neumann, B., R., M.: On scene interpretation with Description Logics. In Christensen, H., Nagel, H.H., eds.: Cognitive Vision Systems: Samping the Spectrum of Approaches. Volume 3948 of LNCS. Springer (2006) 247–278
25. 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)
26. Mohnhaupt, M., Neumann, B.: Understanding object motion: Recognition, learning and spatiotemporal reasoning. Robotics and Autonomous Systems **8**(1-2) (1991) 65–91
27. Luther, M., Böhm, S.: Situation-Aware Mobility: An Application for Stream Reasoning. In: Proceedings of the 1st International Workshop on Stream Reasoning. CEUR Workshop Proceedings (May 2009) To Appear.
28. Artale, A., Franconi, E.: Hierarchical plans in a Description Logic of time and action. In: Proceedings of the International Workshop on Description Logics (DL'95). (June 1995)
29. Andre, E., Herzog, G., Rist, T.: On the simultaneous interpretation of real world image sequences and their natural language description: The system soccer. In: Proceedings of the 8th European Conference on Artificial Intelligence (ECAI'88). (1988) 449–454