A Novel Architecture for Situation Awareness Systems

Franz Baader[1], Andreas Bauer[2,3], Peter Baumgartner[2,3],
Anne Cregan[3], Alfredo Gabaldon[4], Krystian Ji[3], Kevin Lee[3,5], David Rajaratnam[3,5],
and Rolf Schwitter[6]

[1] Technische Universität Dresden, Germany, baader@tcs.inf.tu-dresden.de
[2] Australian National University, *Firstname.Lastname*@anu.edu.au
[3] National ICT Australia (NICTA*), Australia, *Firstname.Lastname*@nicta.com.au
[4] New University of Lisbon, Portugal, Center for AI, ag@di.fct.unl.pt
[5] University of New South Wales, Australia
[6] Macquarie University, Australia, rolfs@ics.mq.edu.au

Abstract. *Situation Awareness* (SA) is the problem of comprehending elements of an environment within a volume of time and space. It is a crucial factor in decision-making in dynamic environments. Current SA systems support the collection, filtering and presentation of data from different sources very well, and typically also some form of *low-level* data fusion and analysis, e.g., recognizing patterns over time. However, a still open research challenge is to build systems that support *higher-level* information fusion, viz., to integrate domain specific knowledge and automatically draw conclusions that would otherwise remain hidden or would have to be drawn by a human operator. To address this challenge, we have developed a novel system architecture that emphasizes the rôle of formal logic and automated theorem provers in its main components. Additionally, it features controlled natural language for operator I/O. It offers three logical languages to adequately model different aspects of the domain. This allows to build SA systems in a more declarative way than is possible with current approaches. From an automated reasoning perspective, the main challenges lay in combining (existing) automated reasoning techniques, from low-level data fusion of time-stamped data to semantic analysis and alert generation that is based on linear temporal logic. The system has been implemented and interfaces with Google-Earth to visualize the dynamics of situations and system output. It has been successfully tested on realistic data, but in this paper we focus on the system architecture and in particular on the interplay of the different reasoning components.

1 Introduction

Situation Awareness (SA from now on) is concerned with the perception of elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future [End95]. Having complete, accurate and current SA is highly desirable for managing real-time dynamic systems including military and emergency scenarios, air traffic control and other transport networks. Poor SA is a key contributor to critical human errors, usually tracing back to cognitive overload or poor information transfer between operators.

* NICTA is funded by the Australian Government's *Backing Australia's Ability* initiative.

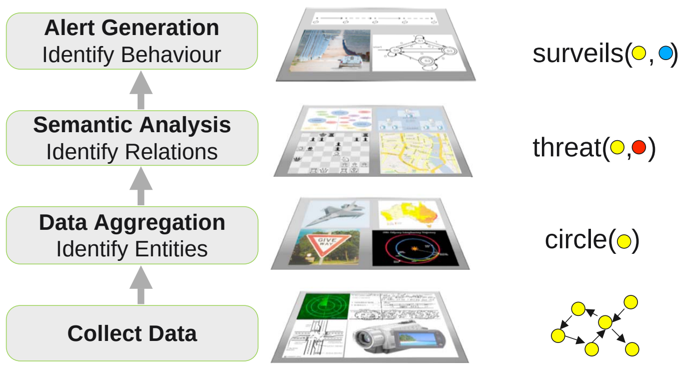


Fig. 1: SAIL high-level system architecture.

According to Endsley's model [End95], the levels of SA are perception, comprehension and projection. Currently, the challenge of integrating heterogeneous information into a single composite picture of the environment at the semantic level of human comprehension and projection remains open. To address this challenge, we have developed a novel system architecture and implemented a system as a part of the *Situation Awareness by Inference and Logic* (SAIL) project. It provides functionality in three successive tiers (Fig. 1) corresponding roughly to Endsley's levels of SA.

Data Aggregation involves monitoring data sources (e.g. track data obtained from radar) to identify entities of a situation and their low-level properties (e.g. that a trajectory contains a circle). This corresponds to the perception level of SA.

Semantic Analysis involves interpretation and evaluation of the entities in conjunction with background knowledge, producing an understanding of the overall meaning of the identified entities: how they relate to each other, what kind of situation it is, what it means in terms of one's mission goals (e.g. that a fighter plane is threatening some object). This corresponds to the comprehension level of SA.

Alert Generation involves monitoring and projecting how events may unfold over time. Based on the interpretation of a situation, this functionality identifies possible evolutions of the current situation (e.g. of an aircraft currently surveilling a border). If a potentially high-impact situation is recognised to arise, an alert is sent to the operator. This relates to the projection level of SA.

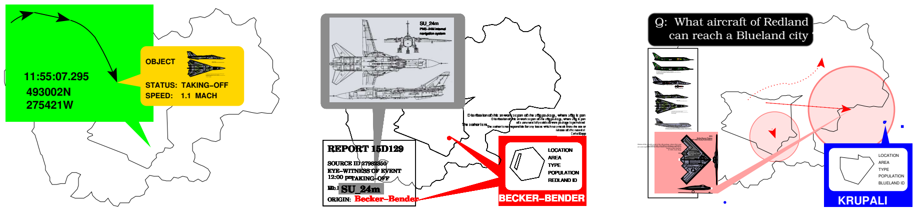
We emphasize the role of declarative techniques in all three layers, each one realized essentially by specification in a formal logic. This bears the advantage that we have a precise semantics for each layer, defined in terms of formal logic rather than the implemented behaviour of a specific set of tools. As formal semantics can be captured in an abstract manner, our approach does not tie users to specific implementations. However, to make the employed logics operational, we capitalize on the state of the art by utilizing two of the latest available (tableau-based) reasoners, E-KRHyper [PW07] and RacerPro [HM03]. We exploit E-KRHyper's model-building abilities to deliver its result as a description-logic ABox to RacerPro. One caveat here lies in the dynamic nature of our application, which requires reasoning about data that changes over time. Unfortunately, the currently available (first-order and description logic) reasoners do not offer suitable services, so we solve this problem via a novel architecture, the *SAIL* architecture, that utilizes a control component to invoke the reasoners in specific ways. Other important aspects of SAIL concern the use of controlled natural language (CNL) as the primary means of interaction with human operators, the integration of a public-domain GIS system into E-KRHyper for basic geometrical calculations. The architecture has been implemented in the *SAIL system* and tested successfully on realistic data. Although we refer occasionally to the system, the focus of this paper is on its architecture.

1.1 Related Work

Several systems for SA have been developed that support the management of various information sources (sensor data, textual information, databases, etc.) for purposes such as information exchange and graphical presentation to facilitate decision making. Information retrieval techniques to filter and rank a potentially overwhelming stream of information are often applied to facilitate this process. Typical examples of academic prototypes, from the military domain, are [GHGJ$^+$07,SRS$^+$07]. However, such systems lack capabilities that enable a deep, semantical modelling of the domain and drawing conclusions on top of it. Therefore, information integration to assess and project a situation into the future still has to occur largely in the "heads of the decision makers", which leads to the *Semantic Challenge* in SA [NL05]. These issues are a recurring theme in current SA research and attract considerable attention worldwide (see a recent overview paper by domain experts [BKS$^+$06]). It is not surprising that (in particular) *description logics* [BCM$^+$03] and corresponding reasoners have attracted a lot of attention in the SA community. Although some SA tools, such as [SRS$^+$07,GHGJ$^+$07] reportedly make use of description logic reasoners, it becomes clear that deep, semantic reasoning is not yet implemented in a satisfying manner. Moreover, the cited reports make little mention of the lower levels of the information fusion process; there is an implicit, underlying assumption that low-level data, such as that provided by physical sensors, is already in a semantically and syntactically well structured form, associating to real-world events the concrete objects, the time of occurrence, and data sources, which may be stored in a database or ontology. However, how to actually obtain this meta information is not detailed upon (cf. [MKL$^+$05]). To the best of our knowledge, no prior SA system has managed to use an integrated semantic approach based on logical inference and reasoning, from the sensor-data level to the higher levels of situation assessment, and beyond (e.g., impact assessment, projection into the future, etc.). This is one of the key differences between our proposed system, whose functioning can be described largely based on formal logic, and the existing ones, which combine ad-hoc representations with formal reasoning.

1.2 Running Example

We will use the scenario depicted in Fig. 2 as a running example to illustrate the various components of the SAIL architecture. Our work is embedded in a project run in collaboration with the Defence Science and Technology Organisation (DSTO), Australia.



(a) Surveillance systems (e.g. radar) detect an aircraft taking off at 11:55.

(b) A witness reports seeing an SU_24M taking-off from Becker-Bender at 12:00.

(c) A SAIL-system user submits the CNL query "What aircraft of Redland can reach a city of Blueland?"

Fig. 2: Example Scenario: multiple sources of information are fused and analysed in the detection of a potentially hostile aircraft taking off.

The scenario discussed here is a small excerpt from a scenario developed by DSTO and NATO partners.

Suppose we are interested in monitoring a geographical region for potentially hostile activity by a neighbouring country. Our system receives low-level track data from electronic surveillance systems (e.g. radar) that combined with Geographic Information System (GIS) data allow us to keep track of the movement of vehicles in the region. In this scenario, radar data indicates that an aircraft takes off at 11:55 and travels at a speed of 1.1 Mach (Fig. 2a). Another source of information to the system are natural language reports from human witnesses in the region. These reports frequently contain information that the other sources cannot deliver, such as the specific type of aircraft. In our scenario, such a report indicates that an SU_24M was seen taking off from Becker-Bender at 12:00.

Interacting with the system is a user issuing queries in (Controlled) Natural Language. In this example, a submitted query is "What aircraft of Redland can reach a city of Blueland?" (Fig. 2c). The SAIL system 1) uses its various sources of information to speculate that the aircraft detected by radar and the aircraft described in the report are the same; 2) uses what it knows about the detected SU_24M and general background knowledge, such as aircraft capabilities (e.g. travel distance range), geographical information, and the types of aircraft used by different countries, to determine that the detected SU_24M is Redland's and that it is currently capable of reaching a city of Blueland, and 3) includes it in the answer to the user's query. Moreover, in this example the SU_24M and any other aircraft that have been detected are monitored for *aggressive* behavior. If an aircraft is determined to be aggressive, the system issues an alert.

2 SAIL Architecture

Fig. 3 depicts the SAIL system architecture. SAIL takes two kinds of input: *data streams* and *eye-witness reports*. Data streams provide time-stamped sensor data about aircraft, ships, etc, giving their location, speed, acceleration and so on, shown as the boxes $\mathsf{SD}_i, \mathsf{SD}_{i+1}, \mathsf{SD}_{i+2}, \ldots$. In our scenario, new data arrives about every 0.33 seconds and

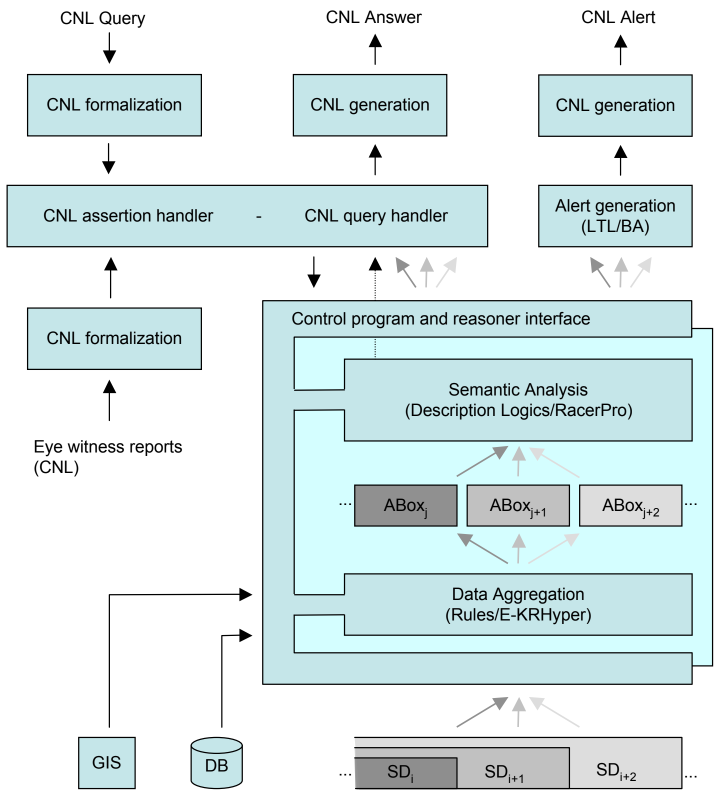

CNL Query
CNL Answer
CNL Alert
CNL formalization
CNL generation
CNL generation
CNL assertion handler - CNL query handler
Alert generation (LTL/BA)
CNL formalization
Control program and reasoner interface
Semantic Analysis (Description Logics/RacerPro)
Eye witness reports (CNL)
ABox_j
ABox_j+1
ABox_j+2
Data Aggregation (Rules/E-KRHyper)
GIS
DB
SD_i
SD_i+1
SD_i+2

Fig. 3: SAIL system architecture. Abbreviations: GIS - GIS system, SD_i - sensor data, LTL - linear temporal logic, BA - Büchi automaton, CNL - controlled natural language.

concerns about 30 objects. SAIL accumulates this information: at each time point i the system stores data from previous time points SD_j, for $j \leq i$. For practical reasons – to cope with the amount of data accumulating over time – SAIL supports a user-configurable *time window* and abandons data time-stamped prior to that window. Eye-witness reports are represented in a time-stamped relational way, similarly to the sensor data. As they are originally expressed in a form of controlled natural language (CNL), some preprocessing is needed to arrive at a relational form (cf. Sec. 2.4). The control program presented in Algorithm 1 coordinates the processing of all that.

The control procedure takes two arguments - a TBox *tbox* and a configurable window size n. There are two arrays *sd* and *data_trace*. Each array is of size n and each element in the array is initially set to an empty set. The main part of the procedure consists of a while-loop that runs indefinitely. In each iteration, the procedure is blocked until the next input arrives from the CNL assertion handler. It then proceeds by reading *user_queries* in nRQL format from the query handler. Positions of array *sd* are filled with sensor data in sequence from index 0 to $n-1$ as t increments with each iteration.

```
input: a TBox $tbox$, window size $n$
$t \leftarrow 0$;
for $i \leftarrow 0$ to $n-1$ do                    /* initialize sensor data windows */
    $sd[i] \leftarrow \emptyset$;
end
while $true$ do
    $sd[t \bmod n] \leftarrow$ await input data from CNL assertion handler;
    $user\_queries \leftarrow$ read nRQL queries from CNL query handler;
    $SD_t \leftarrow \bigcup_{i=0}^{n-1} sd[i]$;
    $ABox_t \leftarrow$ InvokeDAModule($SD_t$);
    $tptp\_answer \leftarrow$ InvokeSAModule($tbox, ABox_t, user\_queries$);
    $data\_trace \leftarrow$ InvokeSAModule($tbox, ABox_t$);
    send $tptp\_answer$ to CNL query handler;
    send $data\_trace$ to alert generation module;
    $t \leftarrow t+1$;
end
```
Algorithm 1: Control procedure for query answering and data trace generation

It simply loops back to position $(t \bmod n)$ if $t > n$, thus disregarding input data collected at time point $t - n$. That is, we retain at most n time points of input data prior to the current time point t.

Collected input data in the array sd are aggregated to form SD_t. This data, together with GIS and database information, are processed by the data aggregation module via `InvokeDAModule`, to produce an ABox $ABox_t$ marked with the current time point t. This new ABox is loaded together with the input TBox $tbox$ to the semantic analysis module via `InvokeSAModule` for further processing, where user queries $user\_queries$ will be executed. The answer produced will be in TPTP format and is immediately sent to the CNL query handler for conversion to CNL output. Similarly, a data trace will be generated via `InvokeSAModule` for time point t which will be sent to the alert generation module for further processing (as described in Sec. 2.3 below).

2.1 Data Aggregation

Data aggregation is the process of gathering information and expressing it in a summary form. With a wide enough time window, this allows to detect object properties over time, like, for instance, to detect a “circle” in an object’s trajectory, or that a certain object re-visits a certain point again and again. Eye-witness reports, initially expressed in CNL, also feed into this layer. For instance, a CNL sentence like “An SU_24M started from Eaglevista at 09:00” could be used to enrich information about a previously unidentified object that it is an SU_24M.

The core component for data aggregation is the theorem prover E-KRHyper. E-KRHyper is sound and complete for first-order logic with equality. It is an implementation of the E-hyper tableau calculus [BFP07]. E-KRHyper has been described in more detail in [PW07].

E-KRHyper is invoked by the control program whenever the time window moves or a new eye-witness report comes in. It then updates the data aggregated so far based on the new information. E-KRHyper accepts if-then rules, of which first-order clause logic is a special case.[7] Rules are used to specify data aggregation in a declarative way. Here are some examples, relevant to the event in Fig. 2a:

```
object_appears(Obj, Now) :- % An Object appears at the current time, Now.
    current_time(Now),      % Current time - supplied by control program
    object(Obj, Now),       % Get some Object existing at Now
    previous_time(Now, T),  % Previous time - supplied by control program
    \+ object(Obj, T).      % Check that Object was not there at T

take_off(Event, Obj, Now) :- % a new Event: an Object takes off Now
    object_appears(Obj, Now),
    in_air(Obj, Now),        % in_air computed by GIS
    concat(['ev_',Obj,'_',Now],Event). % creates unique Event id
```

The rules are applied in a bottom-up way to the sensor data and eye-witness reports (after conversion into logical form by the CNL assertion handler) until a fixed point is reached. A certain subset of the fixpoint is then extracted and passed on to the semantic analysis layer.

To see how this works, assume that the appearance of the aircraft in Fig. 2a is represented as `object(ac1, '11:55')`, and assume that `in_air(ac1, '11:55')` can be established with the help of the GIS (see below) from the radar data. E-KRHyper then derives the result `take_off('ev_ac1_11:55', ac1, '11:55')`,[8] which is passed on to the semantic analysis layer with the help of rules with a head predicate "`abox`":

```
abox(take_off(Event)) :- take_off(Event, Obj, Time).
```

Such rules specify concept assertions (like `take_off('ev_ac1_11:55')`) or role assertions (like `event_time('ev_ac1_11:55', '11:55')` in the sense of description logics. [9] As indicated in Fig. 3 by using indices *j* rather than *i*, the sequence of ABoxes $\mathsf{ABox}_j, \mathsf{ABox}_{j+1}, \ldots$ does not necessarily correspond to the time points of the sensor data. This is, because it is neither necessary nor feasible to update the data aggregation with the same rate as the sensor data come in. By default, a new ABox is computed every second or when a new eye-witness report comes in.

Preserving information over time. Each ABox $\mathsf{ABox}_j, \mathsf{ABox}_{j+1}, \ldots$ represents information for the time point "now". Would ABoxes include their predecessors (as is the

[7] The syntax is similar to Prolog, but extends it by a logical-or operator "`;`" which means disjunction in the head of rules. E-KRHyper also supports stratified negation "\+" and evaluation of arithmetic goals. These features are helpful for computations on timepoints and spatial distances. They are used locally inside the data aggregation layer and do not interfere with the overall logic.

[8] We use integers to represent time and writing time points as in `'11:55'` is just for readability here. Names like `'ev_ac1_11:55'` are obtained with a concat-atoms like built-in function.

[9] A collection of such assertions, an ABox, specifies a theory in the sense of first-order logic. Like any first-order theory an ABox may be incomplete, that is, it might entail a given first-order sentence, like e.g. a concept instance `is_AWACS(o1)`, entail its negation, or neither of them.

case with the sensor data) then a massive frame problem is to be expected. However, ABox assertions that are consistent with all later ABoxes can be kept without problems. For instance, it might be useful to keep the “take off” event synthesized in the example above until its object no longer exists. This can be realized by writing rules with special predicate symbol `reassert` in the head:

```
reassert(take_off(Event, Obj, CreationTime)) :-
     take_off(Event, Obj, CreationTime), current_time(Time),
     object(Obj, Time).  % reassert as long as its object exists
```

Like the `abox` predicate, the `reassert` predicate is interpreted by the control program in a special way, by just feeding its extension in the current computed model into the next invocation of E-KRHyper. This achieves the desired effect.

The formal meaning of this type of data aggregation can be given in a similar way as for production system languages (cf. [Ras94]) and E-KRHyper merely serves as an implementation of that. However, as pointed out above, this particular choice is not mandatory. We have chosen E-KRHyper because of our familiarity with the system. The use of equality reasoning is not crucial, but the ability to generate models in a bottom-up manner is, in order to extract ABoxes from the computed models. Another important detail is that the rules we use fall into a formula class that E-KRHyper is a decision procedure for.

Databases and GIS. SAIL’s data aggregation layer also integrates databases and a public-domain GIS. The (relational) databases here contain data about capabilities of aircraft, ships, etc, such as maximum speed, range, and so on. As they are rather small, instead of coupling a proper DBMS, we simply expressed the knowledge base directly in E-KRHyper’s input language, as a set of facts.

The sensor data consists of spatial information about objects that are moving over time. The computation of their properties will often involve spatial computations in the GIS, numerical computations and database lookups. A simple example is to compute whether a certain aircraft can reach a certain destination. This requires a database query of the aircraft’s range, the aircraft’s time in the air, and the distance to the destination. For that, it is useful to be able to compute basic spatial properties of objects, such as whether an object is in air, within a certain region (for example, a country) or targets some city. To this end, we have integrated the popular open source GDAL/OGR GIS library into the SAIL data aggregation layer. The GIS utilises vector data about geographic features, described in terms of geometric objects such as points, lines, and polygons. The interface to the GIS is realized by special predicates and functions, which can be used in rule bodies to invoke its services.

For example, determining whether a city can be reached by a given aircraft can be performed by using a combination of builtins; first to define the geographic region that the aircraft can reach, and then to test whether the city is within that region:

```
reachableCity(Aircraft, Range, City) :-
    Reach is ogr_g_buffer(Aircraft, Range),
    ogr_g_within(City, Reach).
```

2.2 Semantic Analysis

The semantic analysis layer is where situation assessment occurs. This component of the system utilizes a logical description of domain properties at a conceptually higher-level than what the data aggregation layer produces. While the data aggregation layer generates assertions about, e.g., the location of objects, the semantic analysis layer attaches higher level meaning to the situation, e.g. whether the object is behaving aggressively. In addition to the information coming from the data aggregation layer, the semantic analysis layer has at its disposal a background knowledge base (an ontology) containing information about the different aircraft, ships, and other vehicle types available to various countries, the capabilities of the vehicles, e.g. weapons, travel range; the status of the relationship between countries, e.g. ally, neutral, hostile; and other similar domain background knowledge. The knowledge base also contains a collection of definitions of *events*, e.g., move, fly, sail, depart, take_off, etc, that is in part inspired by the event-semantics used in the NLP community. From this knowledge base and the information delivered by the data aggregation layer, the semantic analysis layer computes, by logical reasoning, a deeper, more concise high-level description of the current situation. In the following we give only a flavor of the concrete knowledge base with a small example around "aggressive behavior".

The particular representation and reasoning formalism used in this layer is description logics (DL) [BCM+03]. A typical DL knowledge base has two components: an ABox and a TBox. ABox assertions are of the form $C(x)$ or $R(x,y)$, where C and R denote *concept* and *role* respectively, and x,y are individuals. Assertions are provided to the semantic analysis layer as background information, as eye-witness reports or generated by the data aggregation layer. For example, the assertions below state two facts - $t1$ being a physical object and $t1$ being a target of another object $o1$:

$$physical\_object(t1) \qquad has\_target(o1,t1)$$

TBox axioms are of the form $C \sqsubseteq D$ or $C \doteq D$. The former requires that every individual belonging to C also belongs to D. The latter equivalence axiom requires that both $C \sqsubseteq D$ and $D \sqsubseteq C$ hold. TBox axioms are pre-defined in the semantic analysis layer. They are combined with the ABoxes from the data aggregation layer to draw additional inferences.

For example, the following axiom defines an aggressive object as an individual that has a target of either a physical object or a space region:

$$aggressive \doteq \exists has\_target.(physical\_object \sqcup space\_region)$$

It then follows that $o1$ is an aggressive object, $aggressive(o1)$. Note that the axiom above is defined as an equivalence, which means we are not only interested in the sufficient condition for classifying an object as aggressive, but the sufficient and necessary conditions. For example, suppose we discover the fact $aggressive(o1)$ from an eye-witness. The above axiom would infer that there is *some* unnamed object which is a target of $o1$ and is either a physical object or a space region.

Also related to "aggressive" is the thematic role *has_target*, which is used to represent the target in an aggression event. Like the concept *aggressive*, this role is also non-primitive and is similarly defined in this layer in terms of other primitive concepts and roles. Specifically, non-primitive roles are defined by means of rules in the DL system language. For instance, the following rule defines *has_target*:

```
(firerule (and (?EM move) (?EM ?Ag has_theme) (?Ag fighter)
           (?Ag ?Org associated_with) (?Org s_blueland enemy_organization)
           (?EM ?Y has_direction) (?Y s_blueland associated_with))
   ((related (new-ind aggr ?Ag ?Y) ?Y has_target)))
```

This rule can be read as follows: if there is a *move* event whose theme (agent), `?Ag`, is a fighter aircraft associated with an enemy organization of Blueland and `?Ag` is moving towards `?Y` (a physical object or a space region) which is associated with Blueland, then this agent `?Ag` has `?Y` as a target of aggression.

The connection between the data aggregation layer and the semantic layer is achieved through primitive concepts/roles. Primitive concepts and roles are not fully defined in DL, but are continuously populated by instances computed by the data aggregation layer. The implementation of the rules filling the primitives needs to be sound wrt. their intended meaning. Completeness, however, is not required and is generally impossible to achieve (i.e., limited possibilities of observing the world). The DL reasoner can then also deduce assertions involving defined concepts. The eye-witness reports may directly yield assertions for defined concepts.

Implementation. Our implementation of the semantic analysis module utilizes RacerPro [HM03]. RacerPro is invoked by the same control program that drives the data aggregation layer. With RacerPro running in server mode, the control program loads the TBox (i.e. the ontology) containing axioms defining high-level concepts like *aggressive* and an ABox containing static background knowledge, then it enters into a loop. In each iteration of the loop, the control program loads into RacerPro the most recent ABox produced by the data aggregation layer and then executes a number of DL rules, such as the one above, that essentially extend the ABox with additional, inferred facts. Once this is done, the reasoner has a full knowledge base and is ready for query processing. Three classes of queries are issued to the reasoner: 1) Localization queries, which are automatically issued by the control program and whose answer is used to display the various objects currently being tracked on a Google-Earth interface. 2) User queries, issued in CNL as described in Sec. 2.4 below. (It is also possible for a user to pose queries directly in the language of the reasoner.) 3) Alert queries, which conceptually belong to the Alert layer but are realized by description logic reasoning (See Sec. 2.3).

2.3 Alerts

In contrast to queries, whose answering is triggered by questions submitted by the user, alerts are raised automatically by the SAIL system. For instance, the user may want to be notified by the system whenever an aircraft crosses a predefined border or an air corridor. In general, an alert describes a critical situation, whose occurrence should be pointed out to the user immediately, without requiring additional interaction. An alert can be created by formally specifying the critical situation in linear time temporal logic (LTL) [Pnu77]. The reason for using *temporal* logic is that the occurrence of a critical situation usually depends on the dynamic behaviour of objects. Single time points are described by the ABoxes generated by the data aggregation layer of SAIL and extended by the semantic analysis layer. These form the atomic propositions in the LTL formulas, i.e., statements about the properties of named objects formulated in terms of the concepts and roles occurring in the ontology. Once an alert is formally specified, a *monitor* is created, which, based on the observations so far, decides whether or not the alert should be raised.

From a formal point of view, an LTL formula φ specifying an alert defines a set L_φ of infinite "words", where each letter in such a word can be seen as a description of the actual state at a given time point. These words correspond to "good" behaviour, i.e., the alert must be raised if the observed sequence of state descriptions does not belong to L_φ. To be more precise, at any given time point, we have only observed a finite prefix of such an infinite sequence. Such a prefix is "bad" if it cannot be extended to an infinite sequence that belongs to L_φ, i.e., however the future behaviour looks like, we know that it cannot become "good." In this case, the alert must be raised. Conversely, the prefix is "good" if all extensions belong to L_φ. In this case, the system no longer needs a monitor for this alert. If none of this is the case, then the system must continue monitoring.

Following an approach developed in the area of runtime verification [BLS06], the monitor for an alert specified by φ is a finite state machine (FSM) with output ("alert", "shut down", "continue"), which can be constructed from the Büchi automaton corresponding to φ (i.e., accepting L_φ).

As an example of an alert specification, consider the following critical situation. If we detect that an enemy aircraft has taken off, and if this aircraft crosses our border, an alarm signal should be raised. The following LTL formula is used to express this:

$$\mathbf{G}(\mathit{in\_air}(p) \Rightarrow \neg \mathit{cross\_border}(p)\ \mathbf{U}\ \mathit{landed}(p)).$$

The temporal operator **G** asserts that the formula following it should hold at all future time points, and the until-operator **U** asserts that the event $\mathit{cross\_border}(p)$ does not happen before $\mathit{landed}(p)$ is observed. Note that this formula is parametrised with an object name p. The idea is that it is instantiated by all the named objects that are *in_air*. In our running example, alerts are illustrated by monitoring for aggressive events with the simple specification $\mathbf{G}(\neg \mathit{aggressive}(e))$.

Moreover, in our application, for all aircraft p, we keep track of the values of, say, $\mathit{in\_air}(p)$ such that we can, essentially, revert to a propositional representation of the formula. From that, we automatically generate a FSM that reads a trace, which consists of the different truth values of the propositions over time (and obtained via description logic reasoning), and returns in each state whether so far a good prefix was observed (i.e., "shut down"), a bad one (i.e., "alert"), or neither (i.e., "continue"). Depending on the formula, not all the three different types of states may appear in a generated FSM. For example, in the resulting FSM from the above formula depicted in Fig. 4, there is only one alert-state, indicated by the label $(-1,1)$, and two continue-states, indicated by the labels $(0,0)$ and $(1,1)$. The labels on the transitions are such that only positively interpreted propositions appear, whereas negative interpretations are implicit, for example, the label "inair_p" asserts the following valuation,

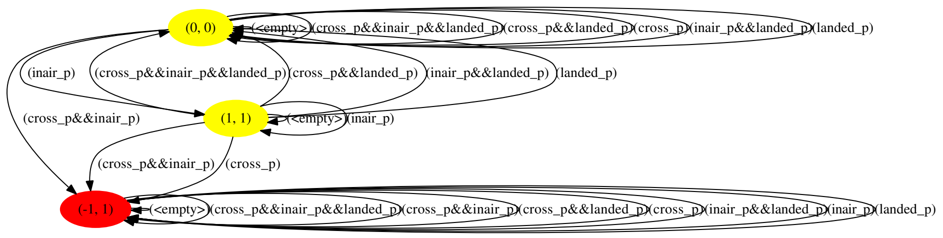


Fig. 4: FSM for $\mathbf{G}(in\_air(p) \Rightarrow \neg cross\_border(p)\ \mathbf{U}\ landed(p))$.

$\{in\_air(p) = true, cross\_border(p) = false, landed(p) = false\}$. Note that the automatically generated FSMs are always complete in the sense that for all possible Boolean combinations of propositions there always exists a transition at each state. The transitions in the FSMs can often be simplified, for efficiency reasons. For instance, if a state has no outbound states but loops, we replace them with a single loop that is always enabled irrespective of the truth values of the propositions.

Instead of a direct implementation, FSMs are realized by reduction to description logic reasoning. This is done in a similar fashion as the encodings of LTL search control for planning in [Gab03]. For each state S in the FSA, we introduce a concept C_S into the semantic analysis knowledge base. For each of these concepts, we construct a DL formula by taking all the transitions (S_i, ψ_i, S) in the FSM and building a corresponding disjunction of the form $\bigsqcup_i C_{S_i} \sqcap \psi_i$. This formula essentially defines the extent (set of individuals in the corresponding state) of the concept C_S in the next time point. Every time a new ABox is loaded into the semantic analysis module, we use the current values of the concepts C_{S_i} and recompute the value of C_S by computing the answer to the query $\bigsqcup_i C_{S_i} \sqcap \psi_i$. This answer is then stored and loaded in the next time point so that the current values of the state concepts are always available for computing the values in the next time point. For example, for the formula above, we have three concepts C_1, C_2, C_3 meant to contain objects in the corresponding three states. The formulas for updating the concepts are, resp.: $(C_1 \sqcap \neg in\_air) \sqcup (C_2 \sqcap landed)$, $(C_2 \sqcap \neg landed \sqcap \neg cross\_border) \sqcup (C_1 \sqcap in\_air)$, $C_3 \sqcup (C_1 \sqcap cross\_border) \sqcup (C_2 \sqcap cross\_border)$. A comprehensive formal semantics of the alerts layer can also be given through a combination of DL and LTL as described in [BGL08].

2.4 Controlled Natural Language (CNL) Interface

A computer-understandable controlled natural language (CNL) is an engineered subset of a natural language designed to reduce ambiguity and vagueness that are inherent in full natural language. Our controlled natural language is based on an unification-based grammar similar to [FKK08,ST08] and relies on a neo-Davidsonian representation of events, and a small number of thematic roles that are used to link these events with other discourse entities (see [Par94] for an introduction).

The purpose of the CNL interface in the SAIL architecture is to allow humans who are not trained in formal logic to add eye-witness reports to the system and to query the

DL knowledge base in CNL. This high-level interface abstracts away from primitive and defined concepts and from the formal notation used to encode these concepts in the knowledge base. Working with such an interface has the advantage that the user can employ the familiar terminology of the application domain to interact with the system, and we expect that this will reduce the cognitive load of the user considerably in a situation awareness context.

Implementation. The CNL processor of the SAIL system consists of a controlled lexicon and a bi-directional grammar. It translates declarative sentences and questions written in CNL into a formal representation in TPTP syntax [SS98], and it generates answers and alert messages in CNL. (We have chosen TPTP syntax for ease of interfacing reasoners.) The kernel of the CNL grammar is built around declarative sentences which have the following simple functional structure:

Subject + Predicate + [Object] + {Modifiers}

This functional pattern can be instantiated via a set of well-defined CNL constituents, for example:

Subject: (Su_24M) Predicate: (reaches) Object: (Bendeguz) Modifier: (within 6 minutes).

Starting from this simple sentence pattern, more complex sentences can be built in a systematic way using a number of constructors (for example coordinators and quantifiers). The CNL processor is able to resolve anaphoric references during the parsing process using the DL knowledge base. This results in a paraphrase that shows the user how the anaphoric expressions were interpreted. As an example consider the following eye-witness report:

SU_24M takes off from Becker-Bender at 09:00. The A50-1 takes off from Krupali at 09:30.
The fighter (SU_24M) flies towards Bendeguz. The AWACS (A50-1) flies towards Eaglevista.

Apart from a paraphrase, the CNL processor generates first a TPTP representation for this eye-witness report. This representation is then translated further into a suitable form to augment the existing information in the data aggregation layer.

The DL knowledge base can be queried in CNL. Questions usually have an inverted word order but the processing of questions can be interpreted as a variation of the processing of declarative sentences. Thus large parts of the same grammar can be used. Here is an example of a typical *wh*-question:

What aircraft of Redland is able to reach a city of Blueland?

The CNL processor translates this question into a TPTP formula and stores this formula as a template for generating answers:

```
input_formula(sail,conjunctive_query,((
 (? [A]: (named(A, s_redland) & (object(B, aircraft) &
                              property(B, associated_with, A)))) &
  (? [C]: ( (? [D]: (named(D, s_blueland) & (object(C, city) &
                              property(C, associated_with, D)))) &
  (? [E]: (property(E, has_agent, B) & (poss(E) & (event(E, reach) &
          (property(E, has_theme, C) & contemp(E, u)))))))))
  => answer(B))).
```

The TPTP formula is then translated into a conjunctive nRQL query [HM03]:

```
(retrieve (?1) (and (?1 aircraft) (?1 s_redland associated_with)
  (?2 ?1 has_agent) (?2 reach) (?2 ?3 has_theme) (?3 city)
  (?3 s_blueland associated_with)))
```

This query is sent to the DL reasoner, RacerPro, for question answering. RacerPro returns the found instances. The CNL processor takes the stored TPTP formula and transforms it into a representation for a declarative sentence using these instances, and one or more complete sentences are generated as an answer.

3 Conclusions

We presented SAIL, a novel system architecture for situation awareness. It differs from other approaches by emphasizing the role of formal logics and automated reasoning systems. This supports a highly declarative approach to building situation awareness systems. To our knowledge, SAIL is the first approach of this kind, and we see the underlying approach of combining and extending *available* reasoners beyond their native capabilities as our main contribution. A core feature of our architecture is that it enable computation with data that changes over time, which is crucial for situation awareness but not natively supported by the reasoners. Additional components of the implementation are a GIS-system, a controlled natural language interface and Google-Earth visualization of trajectories and alerts.

We have implemented the SAIL system with the functionality described above. It copes with data streams from a realistic scenario in real time. The raw sensor data that feeds into the Data Aggregation layer arrives as a stream of time-stamped tuples `(T,Name,Ptfm,Alleg,Type,Lat,Long,Alt,VelX,VelY,VelZ,AccX,AccY,AccZ)` each describing a detected object's position with real geo-coordinates, physically real velocity and acceleration quantities, and, if known, its platform, allegiance (hostile, friendly or neutral), and type. As mentioned in Section 2, the stream delivers such information on about 30 objects approximately every 1/3sec. E-KRHyper's rule base consists of about 100 rules and ABoxes are generated at a rate of 2 ABoxes per "data-minute", where each ABox contains around 100-400 assertions. Each ABox is combined with the background knowledge assertions and the background knowledge axioms to form a single DL knowledge base. The background knowledge assertions are composed of 43 concept assertions and 28 role assertions. The background knowledge axioms are composed of 18 concept definitions, 14 inclusion axioms (GCIs) and 4 disjointness axioms. RacerPro is able to compute the result of each query in less than 2 real seconds.

As future work it would be interesting to consider applications in domains like air traffic control or disaster management.

Acknowledgements. We thank the reviewers and our DSTO project partner for valuable comments on earlier drafts of this paper.

References

[BCM$^+$03] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BFP07] P. Baumgartner, U. Furbach, and B. Pelzer. Hyper tableaux with equality. In F. Pfenning, editor, *Procs. CADE-21*, *LNAI 4603*. Springer, 2007.

[BGL08] F. Baader, S. Ghilardi, and C. Lutz. LTL over description logic axioms. In G. Brewka and J. Lang, editors, *Procs. KR 2008*. AAAI Press, 2008.

[BKS$^+$06] E. Blasch, I. Kadar, J. Salerno, M. M. Kokar, S. Das, G. M. Powell, D. D. Corkill, and E. H. Ruspini. Issues and challenges in situation assessment (level 2 fusion). *Journal of Advances in Information Fusion*, 1(2):122–139, 2006.

[BLS06] A. Bauer, M. Leucker, and Chr. Schallhart. Monitoring of real-time properties. In S. Arun-Kumar and N. Garg, eds., *Procs. FSTTCS-26*, *LNCS 4227*. Springer, 2006.

[End95] M. R. Endsley. Towards a theory of situation awareness in dynamic systems. *Human Factors*, 37:32, 1995.

[FKK08] N. E. Fuchs, K. Kaljurand, and T. Kuhn. Attempto Controlled English for Knowledge Representation. In *Reasoning Web*, LNCS 5224. Springer, 2008.

[Gab03] A. Gabaldon. Compiling control knowledge into preconditions for planning in the situation calculus. In *Procs. IJCAI'03*, Acapulco, Mexico, 2003.

[GHGJ$^+$07] R. A. Ghanea-Hercock, E. Gelenbe, N. R. Jennings, O. Smith, D. N. Allsopp, A. Healing, H. Duman, S. Sparks, N. C. Karunatillake, and P. Vytelingum. Hyperion—next-generation battlespace information services. *The Computer Journal*, 50(6):632–645, 2007.

[HM03] V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *International Semantic Web Conference*, *LNCS 2870*. Springer, 2003.

[MKL$^+$05] Chr. J. Matheus, M. M. Kokar, J. J. Letkowski, C. Call, K. Baclawski, M. Hinman, J. Salerno, and D. Boulware. Lessons learned from developing SAWA: A situation awareness assistant. In *FUSION'05: 7th International Conference on Information Fusion*. IEEE, 2005.

[NL05] C. Nowak and D. Lambert. The semantic challenge for situation assessments. In *8th International Conference on Information Fusion*. IEEE, July 2005.

[Par94] T. Parsons. *Events in the Semantics of English: A Study in Subatomic Semantics*. MIT Press, Cambridge, 1994.

[Pnu77] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

[PW07] B. Pelzer and Chr. Wernhard. System description: E-KRHyper. In F. Pfenning, editor, *Procs. CADE-21*, *LNCS 4603*. Springer, 2007.

[Ras94] L. Raschid. A semantics for a class of stratified production system programs. *Journal of Logic Programming*, 21(1):31–57, 1994.

[SRS$^+$07] P. R. Smart, A. Russell, N. R. Shadbolt, M. C. Shraefel, and Leslie A. Carr. Aktivesa: A technical demonstrator system for enhanced situation awareness. *The Computer Journal*, 50(6):704–716, 2007.

[SS98] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[ST08] R. Schwitter and M. Tilbrook. Meaningful Web Annotations for Humans and Machines using Controlled Natural Language. *Expert Systems*, 25(3):253–267, 2008.