

A Novel Architecture for Situation Awareness Systems

Franz Baader

Technische Universität Dresden, Germany
baader@tcs.inf.tu-dresden.de

Andreas Bauer · Peter Baumgartner
Australian National University, and
NICTA, Australia

Firstname.Lastname@anu.edu.au

Anne Cregan · Alfredo Gabaldon · Krystian Ji · Kevin Lee
University of New South Wales, and
NICTA, Australia

Firstname.Lastname@nicta.com.au

Rolf Schwitter

Macquarie University, and
NICTA, Australia

Rolf.Schwitter@nicta.com.au

July 1, 2008

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. The current 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 and system implementation as part of *the Situation Awareness by Inference and Logic* (SAIL) project. It differs from other approaches by emphasizing the rôle of formal logics and automated theorem provers in all its main components. This supports a declarative approach to building SA systems across different domains. The paper describes a particular SAIL system and its architecture.

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.

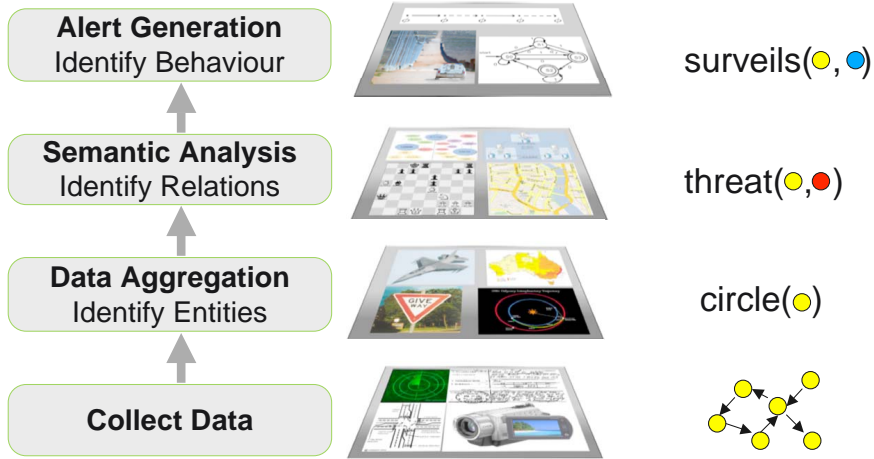


Figure 1: SAIL high-level system architecture.

According to Endsley’s model, 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 system implementation as a part of the *Situation Awareness by Inference and Logic* (SAIL) project. It provides functionality in three successive tiers (Figure 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) and if it recognises the potential for a high-impact situation 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. To make these logics operational, we capitalize on the state of the art by utilizing two of the latest available reasoners, E-KRHyper [PW07] and RacerPro [HM03]. The 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 that utilizes a control component to invoke the reasoners in specific ways.

Figure 1 does not display another important aspect of SAIL: controlled natural language (CNL) as the primary means of interaction. We describe that below, in the context of a more detailed explanation of the SAIL architecture and its realization.

1.1 Related Work

There exist various and variously detailed models for SA in the literature. Since Endsley’s landmark paper [End95] mentioned above, various adaptations and domain-specific alterations of her model have been developed. The *user fusion model* used in the military domain and presented in [Bla00] opts for six different levels including preprocessing of signal data on the lower levels, and processes as well as user interaction on the higher levels. The *comprehension level*, however, coincides in both models in terms of *situation assessment*. Situation assessment forms an important part of the information fusion process in SA applications, and a recent paper by domain experts, [BKS⁺06], discusses problems that currently exist in SA support-tools. Besides methodological issues such as user interface design, a recurring theme are the semantic problems.

For instance, one system discussed is the SAWA system (cf. [MKL⁺05]) which employs loosely integrated reasoners following a deductive (called “logical”) approach, as well as a “procedural” approach where queries are predefined by the developers for performance reasons and simplicity. Similar to the systems described in [GHGJ⁺07, SRS⁺07], logical reasoning turned out to be impractical when combining “off the shelf” theorem-provers and reasoners without a unified, common semantics consistent between all the employed components. The solution chosen by SAWA was to fall back to the simpler, but far less flexible procedural approach. In consequence, information integration had to occur mostly “in the heads” of decision makers.

Addressing these problems has been termed the *semantic challenge* in SA (cf. [NL05]) and is currently an active subject of international research. Recently *description logics* and corresponding reasoners have attracted a lot of attention, spotlighted by the *Semantic Web* efforts of the World Wide Web Consortium’s (W3C) central semantic web language OWL, the *Web Ontology Language*. In a nutshell, OWL is a standardised

knowledge representation language for ontology building, which is formally based on a description logic [BCM⁺03]. Although some SA tools, such as [SRS⁺07, GHGJ⁺07] reportedly make use of OWL already, it seems 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.). Although prior SA systems have clearly demonstrated the feasibility and usefulness of automated SA support, the actual semantics of these systems cannot be described in a coherent formal manner, leading to hard to overcome problems in their respective reasoning components. This, we believe, is one of the key differences between our proposed system, whose functioning can be described entirely based on formal logic, and the existing ones, which combine ad-hoc with formal means of reasoning.

1.2 Running Example

We will use the scenario depicted in Figure 2 as a running example to illustrate the various components of the SAIL architecture. Our work is embedded in the SAIL project run in collaboration with the Defence Science and Technology Organisation, Australia [?, ?, ?]. The scenario discussed here is a small excerpt from The Technical Cooperation Program’s Information Fusion Technical Panel 1 scenario.

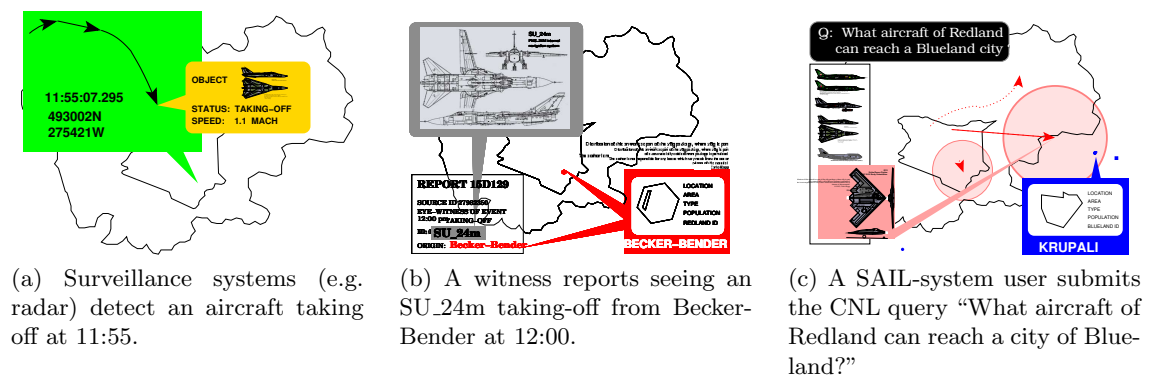


Figure 2: Example Scenario: multiple sources of information are fused and analysed in the detection of a potentially hostile aircraft taking off.

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 is 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, 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 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

Figure 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 $SD_i, SD_{i+1}, SD_{i+2}, \dots$. In our scenario, new data arrives about every 0.33 seconds and concerns about 30 objects. SAIL accumulates this information: at each timepoint i the system stores data from previous timepoints SD_j , for $j \leq i$. For practical reasons 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. Section 2.4). The control program in Figure 3 coordinates the processing of all that.

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. Eye-witness reports ultimately also feed into this layer.

The core component for data aggregation is a first-order logic theorem prover, E-KRHyper. E-KRHyper is a sound and complete theorem prover for first-order logic with equality. It is an implementation of the E-hyper tableau calculus [BFP07], which integrates a superposition-based handling of equality [NR01] into the hyper tableau calculus [BFN96]. E-KRHyper has been described in more detail in [PW07]. E-KRHyper

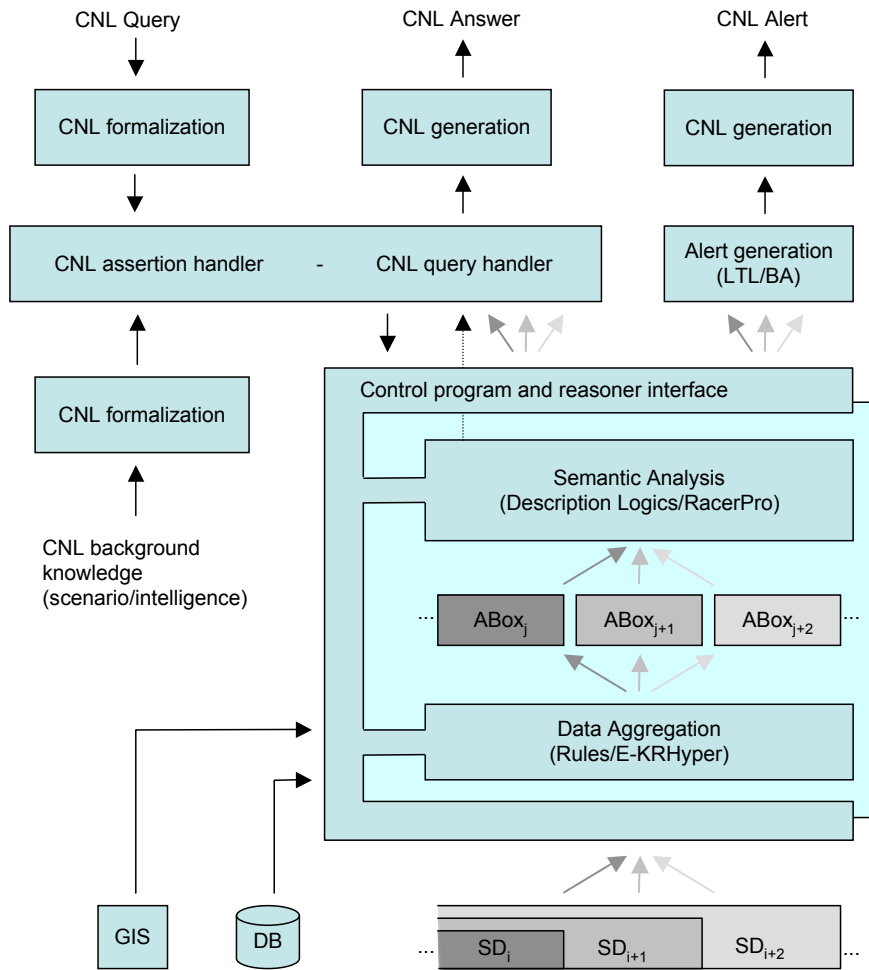


Figure 3: SAIL system architecture. Abbreviations: GIS - GIS system, SD_i - sensor data, LTL - linear temporal logic, BA - Büchi automaton.

is invoked by the control program whenever the time window moves or a new eye-witness report comes in. E-KRHyper accepts if-then rules, of which first order clause logic is a special case.¹ Rules are used to specify data aggregation in a declarative way. Here are some examples, relevant to the event in Figure 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

```

¹The syntax is similar to Prolog, but extends it by a logical-or operator “;” which can be used in the head of rules. E-KRHyper also supports stratified negation and evaluation of arithmetic goals. The latter are helpful for computations on timepoints and spatial distances.

```

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 (see below)
    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 in logical form by the CNL assertion handler) until a fixed point is reached. A certain subset of the fixpoint then is extracted and passed on to the semantic analysis layer.

To see how this works, assume that the appearance of the aircraft in Figure 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')`, which is passed on to the semantic analysis layer with the help of `abox-rules`, like this:

```

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

```

Technically, `abox-rules` specify concept assertions (like `take_off('ev_ac1_11:55')`) or role assertions (like `event_time('ev_ac1_11:55', '11:55')` and `event_object('ev_ac1_11:55', ac1)`). They are interpreted by the control program in a special way.

Sometimes it is useful to keep conclusions persistently over time windows. For instance, “take off” events are kept until its object no longer exists. This is realized by writing rules with special predicate symbol `reassert` in the head, similar to the `abox-rules` above.

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, and are expressed in E-KRHyper’s input language, as a set of facts.

The sensor data consist of spatial information about objects that are moving over time. The computation of their properties will often involve numerical computations, spatial computations and database lookups. A simple example is to determine whether an object is “in air”, by looking at its altitude. Another 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 the latter, it is useful to be able to compute basic spatial properties of objects, such as whether an object is within a certain region (a country, say) 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.

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. For instance, while the ABoxes $ABox_i$ contain assertions about, e.g., the location of objects, this layer attaches higher level meaning to the situation, e.g. whether the object is behaving aggressively. Situation awareness then is achieved by means of logical inference. In addition to the information contained in the ABoxes 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 high-level concepts, e.g. a logic formula describing what it means for an object to behave aggressively. 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.

The particular representation and reasoning formalism used in this layer is Description Logics (DL) [BCM⁺03]. High-level concepts and roles (unary and binary relations, resp.) are introduced by means of DL axioms that define them in terms of the primitive concepts that are populated by the data aggregation layer and also in terms of other high-level concepts defined in this layer. The following is one of the axioms about “aggressive” behavior. It says that all aggressive events have a physical object or a space region as a target:

$$aggressive \equiv \exists has_target.(physical_object \sqcup space_region)$$

Also related to “aggressive” is the 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 (fictional country) 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.

Implementation. Our implementation of the semantic analysis module utilizes a general purpose, off-the-shelf DL reasoner. In particular, for our current implementation we have

enlisted RacerPro². 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 stands 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 are also automatically submitted by the control program in order to check if an alert condition is satisfied at the current time. If this is the case, the user interface displays an alert.

2.3 Alerts

In contrast to queries, whose answering is triggered by a question 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 air plane 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. Thus, the atomic propositions in the LTL formulas defining alerts are ABox assertions, 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 timepoint, 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 its 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

²<http://www.racer-systems.com/>

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 with output (alert, shut down, continue), which can be constructed from the Büchi automaton corresponding to φ (i.e., accepting L_φ). Description logic reasoning is required to determine which is the actual letter, and thus which transition the finite state machine needs to make. In the current implementation of the SAIL system, instead of directly constructing and running the finite state machines realizing the monitors, their behaviour is expressed in Description Logics, which has the advantage that the monitors can be realized using Racer Pro. This is similar to the rewriting approach used in [HR02, Gab03].

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 alert can be shut down as soon as the aircraft has landed outside our territory. The following LTL formula expresses this:

$$\mathbf{G}(in_air(p) \Rightarrow \neg cross_border(p) \mathbf{U} landed(p)),$$

The temporal operator \mathbf{G} asserts that the formula following it should hold at all future time points, and the until-operator \mathbf{U} asserts that the event $cross_border(p)$ does not happen before $landed(p)$ 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 $\mathbf{G}(\neg aggressive(e))$.

2.4 Controlled Natural Language (CNL) Interface

The SAIL architecture features a CNL ³ interface that allows 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. The CNL processor translates declarative sentences and questions written in CNL into a standard representation language for first-order logic, TPTP [SS98], and generates answers and alert messages in CNL. The kernel of the CNL grammar is built around declarative sentences which have the following simple functional structure:

$$\text{Subject} + \text{Predicate} + [\text{Object}] + \{\text{Modifiers}\}$$

³<http://www.ics.mq.edu.au/~rolfs/controlled-natural-languages/>

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

Subject: (Su-24M-1) 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-1 takes off from Becker-Bender at 09:00. The A50-1 takes off from Krupali at 09:30.
The fighter (Su-24-M1) 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. 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_blueand 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.

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. However, available (state-of-the-art) reasoners do not offer suitable services for reasoning about data that changes over time. To solve this problem, SAIL integrates

in its (three) tiers different reasoners in such a way that computation with time and space is enabled. To our knowledge, this is the first approach of its kind and supports a highly declarative approach to building SA systems. In addition, SAIL features a control natural language interface as the primary means of interaction.

Acknowledgements. We thank 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.
- [BFN96] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, number 1126 in Lecture Notes in Artificial Intelligence. European Workshop on Logic in AI, Springer, 1996.
- [BFP07] Peter Baumgartner, Ulrich Furbach, and Björn Pelzer. Hyper tableaux with equality. In Frank Pfenning, editor, *CADE-21 – The 21st International Conference on Automated Deduction*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 492–507. Springer, 2007.
- [BKS⁺06] Erik Blasch, Ivan Kadar, John Salerno, Mieczyslaw. M. Kokar, Subrata Das, Gerald M. Powell, Daniel D. Corkill, and Enrique H. Ruspini. Issues and challenges in situation assessment (level 2 fusion). *Journal of Advances in Information Fusion*, 1(2):122–139, 2006.
- [Bla00] E. P. Blasch. Assembling an information-fused human-computer cognitive decision making tool. *IEEE Aerospace and Electronic Systems Magazine*, pages 11–17, June 2000.
- [BLC⁺08] Peter Baumgartner, Dale Lambert, Anne Cregan, Chris Nowak, Alfredo Gabaldon, Adam Saulwick, Krystian Ji, Andrew Zschorn, Kevin Lee, David Rajaratnam, and Rolf Schwitter. Sail conceptual, reasoning and integration frameworks. Technical report, DSTO Technical Report, 2008.
- [BLS06] Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In S. Arun-Kumar and N. Garg, editors, *Proc. 26th Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4337 of *LNCS*. Springer, December 2006.
- [End95] M. R. Endsley. Towards a theory of situation awareness in dynamic systems. *Human Factors*, 37:32, 1995.
- [Gab03] Alfredo Gabaldon. Compiling control knowledge into preconditions for planning in the situation calculus. In *Procs. 18th International Joint Conference on Artificial Intelligence (IJCAI’03)*, Acapulco, Mexico, 2003.

- [GHGJ⁺07] Robert A. Ghanea-Hercock, Erol Gelenbe, Nicholas R. Jennings, Oliver Smith, David N. Allsopp, Alex Healing, Hakan Duman, Simon Sparks, Nishan C. Karunatilake, and Perukrishnen Vytelingum. Hyperion—next-generation battlespace information services. *Comput. J.*, 50(6):632–645, 2007.
- [HM03] Volker Haarslev and Ralf Möller. Racer: A core inference engine for the semantic web. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*. Springer, 2003.
- [HR02] Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In Joost-Pieter Katoen and Perdita Stevens, editors, *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2002.
- [LBS⁺08] Dale Lambert, Peter Baumgartner, Adam Saulwick, Anne Cregan, Chris Nowak, and Rolf Schwitter. Initial sail methodology and architecture. Technical report, DSTO Technical Report, 2008.
- [MKL⁺05] Christopher J. Matheus, Mieczyslaw M. Kokar, Jerzy J. Letkowski, Catherine Call, Kenneth Baclawski, Michael Hinman, John Salerno, and Douglas 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.
- [NR01] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 371–443. Elsevier and MIT Press, 2001.
- [Pnu77] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.
- [PW07] Björn Pelzer and Christoph Wernhard. System description: E-KRHyper. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 508–513. Springer, 2007.
- [Sau08] Adam Saulwick. Controlled natural language for situation awareness by inference and logic. Technical report, DSTO Technical Report, 2008.
- [SRS⁺07] Paul R. Smart, Alistair Russell, Nigel 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.