# Improving Situation Awareness with Logic Programming: Preliminary Report

Patrick Thor Kahl
NIWC Atlantic
North Charleston, SC, USA
patrick.kahl@navy.mil

Anthony P. Leclerc
NIWC Atlantic
North Charleston, SC, USA
anthony.leclerc@navy.mil

College of Charleston
Charleston, SC, USA
leclerca@cofc.edu

In the military, an ever increasing amount of input data is in need of rapid processing to present as quickly and completely as possible the relevant operational picture to decision makers. Automated processing of more and more of this data is being demanded in the ongoing technology race for superior command and control capability. Combined with simple parsing tools, a logic programming based approach is well-suited to processing structured textual data, typical of certain intel and status reports, for the purpose of aggregating and inferring new information. This is particularly the case when common sense, default reasoning, or expert knowledge is required. We present a preliminary report on our efforts to improve high-level situation awareness through the use of answer set programming in the automated processing of well-structured reports.

## 1 Introduction

Ironically, technological advances tend to give military decision makers *less* time for decisions while providing *more* information. Like most of life's challenges, determining what is relevant is key to dealing with the great influx of data. Doing so in a timely fashion and knowing what to do with it can be the difference between success and failure. The need for rapid processing of data has increased the demand for automation. Machine learning has proven effective for the development of classifiers that rival or even exceed human capabilities. Add data structure, search algorithms, etc., and automating the filtering task begins to appear manageable. However, making relevant inferences from the data has remained mostly a human endeavor. This is where we expect logic programming to make an impact.

Ideally, information concerning the operating environment should be as accurate and complete as possible. However, the picture presented to the decision maker must be concise, conveying only what is relevant at the appropriate level of detail. Additional detail should be readily available on demand. Our focus has been on high-level information as it relates to better comprehension of the situation.

The paper is organized as follows. We begin with a brief summary of background information and related work. Details are then provided for an example chosen from a recent paper that was used to evaluate our methodology. We follow with an overview of our preliminary efforts to improve on the work of other researchers with respect to the example, demonstrating advantages of the answer set programming approach. We close with a synopsis and plans for future work.

## 2 Background and Related Work

### 2.1 Situation Awareness

The base notion of *situation awareness (SA)* is simply being aware of the relevant aspects of the situation/environment of interest, or as Endsley [10] put it, "knowing what is going on." Most animals exhibit a sense of awareness that fits this description. Endsley went further and defined three levels of SA:

Preliminary Report. Submitted to the Defense Technical Information Center (DTIC) at https://discover.dtic.mil.
Funding for this work was provided by the NIWC Atlantic Naval Innovative Science and Engineering (NISE/219) program.

1. **Perception**—observing/recognizing key elements in the environment
2. **Comprehension**—realizing how those elements relate to goals/plans/needs
3. **Projection**—predicting the next state/what will happen in the near future

Technology continues to provide a wealth of tools (e.g., electronic sensors) for improving level 1 SA. Realizing higher levels of SA has remained a predominantly manual effort, though one may argue that improvements in communication have facilitated the opportunity for cooperative effort. Researchers [17, 21] have recently promoted the use of inference as an aid in developing higher level SA. In light of this, the use of logic programming, answer set programming in particular, seems ideally suited for this task.

### 2.2 Answer Set Programming

By 1991, Gelfond and Lifschitz [14] had established the core syntax and semantics of what we now call *answer set programming (ASP)*. In less than a decade, efficient software [19, 9] for finding the answer sets of ASP programs enabled its use in real-world problem solving, leading to its recognition as a new programming paradigm [16]. A basic understanding of ASP is assumed in this paper, though those unfamiliar will find [6] a good introduction.

## 3 Example Scenario

An abridged form of the military scenario described in [21] was used to test our methodology.

- Two opposing forces: BLUFOR and OPFOR
- BLUFOR organizational structure:
  - 1st Battalion (unit BN1)
    - Company 1 (unit BN1_CO1)—*has support role for units* BN1_CO2 *and* BN1_CO3
      - UAV Platoon 1 (unit BN1_CO1_PL1)
      - Anti-armor Platoon 2 (unit BN1_CO1_PL2)
      - Armor Platoon 3 (unit BN1_CO1_PL3)
    - Company 2 (unit BN1_CO2)—*has support role for units* BN1_CO1 *and* BN1_CO3
    - Company 3 (unit BN1_CO3)—*has support role for units* BN1_CO1 *and* BN1_CO2
- BLUFOR rules of engagement:
  ROE-1. If OPFOR unit O advances toward BLUFOR unit B, unit B raises its status to STANDBY.
  ROE-2. If OPFOR unit O is in firing range of BLUFOR unit B, unit B raises its status to ENGAGE.
  ROE-3. If BLUFOR unit A has support role for BLUFOR unit B and unit B enters ENGAGE status, unit A raises its status to STANDBY.
- BLUFOR reports (in order of occurrence):
  1. UAV Platoon 1 reports discovery of a type T-72 tank with hull number R2301.
  2. UAV Platoon 1 reports two more T-72 tanks, one with hull number R2302.
  3. 1st Battalion reports intel saying R2301 and R2302 belong to OPFOR units RC1 and RC2, resp.

From this point, the authors of [21] presented several cases where logical inference can be used to add to situation awareness information. Case 1 is simply the deduction that enemy units are detected in the operating area, posing a threat to 1st Battalion units. We will explore the other two cases by continuing the sequence of reports.

- Case 2:
  4. UAV Platoon 1 reports tank R2301 is advancing toward Company 3.

The deduction here is that OPFOR unit RC1 is advancing toward BLUFOR unit BN1_CO3; thus, rule ROE-1 applies, so unit BN1_CO3 raises its status to STANDBY.

- Case 3:
  5. UAV Platoon 1 reports tank R2301 is in firing range of Company 3.

The deduction here is that OPFOR unit RC1 is in firing range of BLUFOR unit BN1_CO3; thus, ROE-2 applies, and unit BN1_CO3 raises its status to ENGAGE. Consequently, ROE-3 applies, and units BN1_CO1 and BN1_CO2 raise their status to STANDBY. The authors argued that subordinate units share the support roles of their parent units, so units BN1_CO1_PL1, BN1_CO1_PL2, and BN1_CO1_PL3 also raise their status to STANDBY.

## 4 Different Methodologies

For a long time there has been a concerted effort to standardize the types, structure, and language of reports, with objectives to include better information sharing and reduced ambiguity through the use of common terminology and format. These efforts produced a number of artifacts, including the *Coalition Battle Management Language (CBML)* [5], which was designed with automated processing in mind.

The authors of [21] used CBML to create the reports described in the preceeding example. The actual encoding is done using XML [1], with reports comprised of structured and tagged textual data. Figure 1 is an example of the encoding for one such report.

```
<Report xsi:type="WhoTypeType">
   <ReporterWho>
     <OrganisationRef xsi:type="UnitRef">
       <OID>3e868fd6d209</OID>
     </OrganisationRef>
   </ReporterWho>
   <ReportedWhen xsi:type="ReportedWhenAbsoluteTimingType">
     <ReportingDatetime>20100525163000.000</ReportingDatetime>
     <EffectiveStartDatetime>20100525163000.000</EffectiveStartDatetime>
   </ReportedWhen>
   <ReportingData>
     <OID>1044</OID>
     <ReportingDataCategoryCode>REP</ReportingDataCategoryCode>
     <CredibilityCode>RPTFCT</CredibilityCode>
     <ReliabilityCode>A</ReliabilityCode>
   </ReportingData>
   <Who>
     <ObjectItem xsi:type="OtherMateriel">
       <OID>1026: Vehicle</OID>
       <NameText>Vehicle_1026</NameText>
       <HullNumberText>R2301</HullNumberText>
     </ObjectItem>
   </Who>
   <ObjectTypeRef xsi:type="VehicleTypeRef">
     <OID>1025: T-72 Tank</OID>
   </ObjectTypeRef>
</Report>
```

Figure 1: Example Report (taken from [21])

The authors of [21] used Jena [3] and SPARQL [2] to allow for report parsing and applying inference rules for answering queries. Their paper states that the results show the feasibility of their approach, but further work is needed "to realize automatic processing." We provide examples of their inference rules when we compare approaches in the next section.

### 4.1 Our Approach

Our objective in this phase of work was to demonstrate the validity of our approach by repeating the experiment using our methodology, and comparing our results with those of [21]. Figure 2 shows an overview of our report parsing process used for the example scenario.

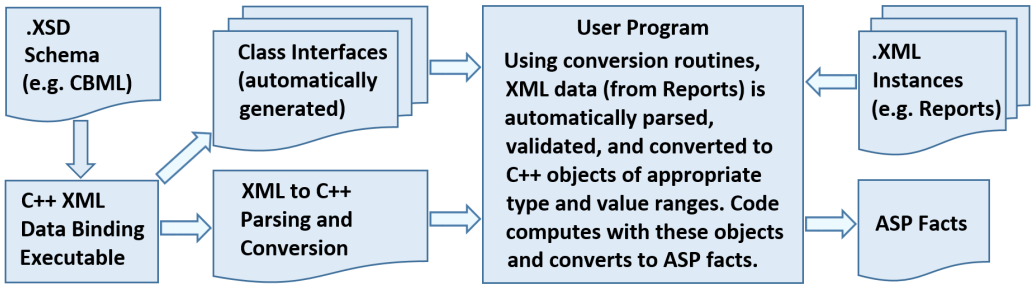


Figure 2: Flow Diagram: Inputs, C++ XML Binding, User Program, and Output (ASP Facts)

C++ XML data binding takes an XML schema (.xsd file) and produces an equivalent set of C++ data types corresponding to the components defined in the schema. This means that XML strings are C++ strings, XML enumerations are C++ enums, XML complex types are C++ classes. XML instance data (.xml file) is automatically serialized, validated, and stored in data types that are type-safe. A developer using C++ XML data binding need only manipulate C++ objects to access or modify the instance data.

The ASP facts derived from the reports include atoms associating sequence numbers with timestamps in reports, ordered sequentially with respect to date and time. For example,

```
ts(1,"20100525163000.000").
```

associates sequence number 1 with the timestamp contained in the report from Figure 1. Other relevant facts generated from that report are:

```
objName("1026: Vehicle", "Vehicle_1026").
objHullno("1026: Vehicle", "R2301").
objType("1026: Vehicle", "1025: T-72 Tank").
```

The facts derived from reports are combined with inference rules based on BLUFOR organizational structure, rules of engagement, and domain knowledge to form an ASP program. For example, the ASP encoding of the rules of engagement are shown in Figure 3 with corresponding Jena rules for comparison.

| Rule ID | ASP Rule | Jena Rule |
|---|---|---|
| ROE-1 | `hasStatus(B,"STANDBY",T) :- advancing2BF(O,B,T).` | `[rule1: (?o AdvancingTowards ?b) (?o Hostile ?b) -> (?b hasStatus "STANDBY")]` |
| ROE-2 | `hasStatus(B,"ENGAGE",T) :- inRangeBF(O,B,T).` | `[rule2: (?o InRange ?b) (?o Hostile ?b) -> (?b hasStatus "ENGAGE")]` |
| ROE-3 | `hasStatus(A,"STANDBY",T) :- hasSuptRole(A,B), hasStatus(B,"ENGAGE",T).` | `[rule3: (?a SupportRole ?b) (?b hasStatus "ENGAGE") -> (?a hasStatus "STANDBY")]` |

Figure 3: Encodings of Rules of Engagement: ASP vs. Jena (simplified for presentation)

One aspect of the ASP encoding that we believe is superior to the Jena encoding is the addition of the sequence number parameter (T) to many of the atoms. This is more expressive, allowing us to represent temporal aspects, such as when the status of a unit changes from STANDBY to ENGAGE.

Another advantage to our ASP encodings comes from the ability to use default negation. For example, three tanks were reported in the scenario given, but only two had hull numbers observed, allowing us to classify them as belonging to OPFOR units based on the intel report. With ASP, we can use defaults to deal with such incomplete information as would be appropriate if the unclassified tank were to advance toward a BLUFOR unit:

```
objOwnerKnown(X) :- objBelongsTo(X,_).
advancing2BF("UNK",B,T) :- objAdvancing2BF(X,B,T), not objOwnerKnown(X).
```

Defaults can also facilitate making programs more elaboration tolerant. For example, a general rule for inferring whether an object represents a threat might be encoded as follows:

```
objThreat2BF(X,B,T) :- objDetectedBy(X,B,T), not objStatus(X,"NOP"),
                       objBelongsTo(X,O), not forceside(O,"BLUFOR"),
                       not -objThreat2BF(X,B,T).
```

Note that the last part of the rule allows for providing explicit exceptions[1] to the rule.

Once assembled, the answer sets of the ASP program are computed using an ASP solver, e.g., `clingo` [4]. Relevant inferences found therein can then be added to the SA information presented to the decision maker.

## 4.2 Experiment Results

The ability to infer the same level of information from reports as in [21] using our approach was confirmed for the three cases given. For case 1, we successfully determined the threats to 1st Battalion, including the addition of an unknown unit threat due to the third tank (ignored in [21]). For case 2, we inferred the status of Company 3 (BN1_CO3) being raised to STANDBY. Our use of sequence numbers also allowed indicating when this change occurred. For case 3, we were able to infer the updated status of 1st Battalion's companies and platoons. Again, we were able to add temporal information due to our use of sequence numbers.

Although our main objective was achieved, we remain critical of certain aspects of our approach. Though we believe adding temporal information to our rules is a superior approach, we realize that proper ordering of steps is dependent on the accuracy and precision of the timestamps given in the reports. To elaborate on a particular concern, consider the problem of assigning the proper sequence order to distinct events from two reports where the timestamp in one is precise only to the date and the other is precise to the second if all events occur on the same day. A related concern is the dependence on continuous and timely reporting of changes within the operating environment since the notions of expiration and duration for events/status have not been explicity encoded. Finally, we have not developed a policy for the selection of reports for inclusion in the overall process. This is acceptable for early research, but it may prove infeasable for a system to take into account all the reports of an extended campaign.

[1] This form of exception is referred to as a *strong* exception. A *weak* exception, e.g., `not ab(f(X,B,T))`, may be added for increased generality. See [13] for details on strong and weak exceptions to defaults.

## 5 Conclusions and Future Work

We demonstrated that the use of ASP as the underlying inference language for automated processing of reports holds promise in improving SA for decision support systems. The ability to include temporal information in the reports and encode default rules that can be easily overridden seems to favor our approach over the Jena/SPARQL approach of other researchers. Further work is required to determine if the flexibility of our method is sufficient for more complex scenarios. If more powerful temporal reasoning is required, methods similar to those described in [7] may be appropriate. Ensuring reports are correctly interpreted remains a challenge in spite of efforts to standardize format, terminology, etc. Context-specific natural language processing may prove fruitful, perhaps using methods like [18, 22].

This preliminary work did not address the highest level of SA, that of being able to reliably predict events/states in the near future. We hope to use ASP planning and diagnostic techniques to move forward in that area. Long term goals include integrating elements of our work with military decision support systems. As early as 2001 [20], ASP was used as part of a decision support system for the space shuttle. ASP continues to find successful application in a number of areas [11, 12, 15]. We hope our efforts will add to this growing field.

## References

[1] (2004): *XML Schema Part 0: Primer 2nd Edition*. Available at `https://www.w3.org/TR/xmlschema-0/`.

[2] (2013): *SPARQL*. Available at `https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/`.

[3] (2019): *Apache Jena*. Available at `https://jena.apache.org/`.

[4] (2019): *Clingo*. Available at `https://potassco.org/clingo/`.

[5] Curtis Blais, Dick Brown, Sidney Chartrand, Saikou Diallo, Kevin Heffner, Stan Levine, Samuel Singapogu, Marc St-Onge & Dan Scolaro (2010): *Coalition Battle Management Language (C-BML) Phase 1 Information Exchange Content and Structure Specification*.

[6] Gerhard Brewka, Thomas Eiter & Mirosław Truszczyński (2011): *Answer Set Programming at a Glance*. *Communications of the ACM* 54(12), pp. 92–103.

[7] Pedro Cabalar, Roland Kaminski, Torsten Schaub & Anna Schuhmann (2018): *Temporal Answer Set Programming on Finite Traces*. *Theory and Practice of Logic Programming* 18(3-4), pp. 406–420.

[8] Jürgen Dix, Ulrich Furbach & Anil Nerode, editors (1997): *Proc. 4th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*. *LNCS* 1265, Springer.

[9] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer & Francesco Scarcello (1997): *A Deductive System for Non-Monotonic Reasoning*. In Dix et al. [8], pp. 364–375.

[10] Mica Endsley (1995): *Toward a Theory of Situation Awareness in Dynamic Systems*. *Human Factors* 37(1), pp. 32–64.

[11] Esra Erdem, Michael Gelfond & Nicola Leone (Fall 2016): *Applications of Answer Set Programming*. *AI Magazine* 37(3), pp. 53–68.

[12] Martin Gebser, Philipp Obermeier, Torsten Schaub, Michel Ratsch-Heitmann & Mario Runge (2018): *Routing Driverless Transport Vehicles in Car Assembly with Answer Set Programming*. *Theory and Practice of Logic Programming* 18(3-4), pp. 520–534.

[13] Michael Gelfond & Yulia Kahl (2014): *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge Univ. Press.

[14] Michael Gelfond & Vladimir Lifschitz (1991): *Classical Negation in Logic Programs and Disjunctive Databases*. *New Generation Computing* 9(3/4), pp. 365–386.

[15] Eray Gençay, Peter Schüller & Esra Erdem (2019): *Applications of non-monotonic reasoning to automotive product configuration using answer set programming*. J. of Intelligent Manufacturing 30(3), pp. 1407–1422.

[16] Victor W. Marek & Mirosław Truszczyński (1999): *Stable Models and an Alternative Logic Programming Paradigm*. In Krzysztof R. Apt, Victor W. Marek, Mirosław Truszczyński & David S. Warren, editors: The Logic Programming Paradigm: A 25-Year Perspective, Artificial Intelligence, Springer, pp. 375–398.

[17] Manisha Mishra, David Sidotii, Gopi Avvari, Pujitha Mannaru, Diego Martínez Ayala, Krishna Pattipai & David Kleinman (2017): *A Context-Driven Framework for Proactive Decision Support with Applications*. IEEE Access 5, pp. 12475–12495.

[18] Arindam Mitra, Peter Clark, Oyvind Tafjord & Chitta Baral (2019): *Declarative Question Answering over Knowledge Bases containing Natural Language Text with Answer Set Programming*. In: Proc. 33rd AAAI Conf. on Artificial Intelligence (AAAI-19), AAAI.

[19] Ilkka Niemelä & Patrik Simons (1997): *Smodels–An Implementation of the Stable Model and Well-founded Semantics for Logic Programs*. In Dix et al. [8], pp. 420–429.

[20] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson & Matthew Barry (2001): *An A-Prolog Decision Support System for the Space Shuttle*. In I.V. Ramakrishnan, editor: Proc. 3rd Intl. Symp. on Practical Aspects of Declarative Languages (PADL 2001), Springer, pp. 169–183.

[21] Fang-Ping Pai, Lee-Jang Yang & Yeh-Ching Chung (2017): *Multi-layer Ontology Based Information Fusion for Situation Awareness*. Applied Intelligence 46, pp. 285–307.

[22] Dhruva Pendharkar & Gopal Gupta (2019): *An ASP Based Approach to Answering Questions for Natural Language Text*. In José Júlio Alferes & Moa Johansson, editors: Proc. 21st Intl. Symp. on Practical Aspects of Declarative Languages (PADL 2019), LNCS 11372, Springer.