Situation Awareness via Abductive Reasoning from Semantic Sensor Data: A Preliminary Report

Krishnaprasad Thirunarayan, Cory A. Henson, and Amit P. Sheth
Kno.e.sis Center, Wright State University, Dayton, OH 45435
t.k.prasad@wright.edu, cory@knoesis.org, amit@knoesis.org

ABSTRACT

Semantic Sensor Web enhances raw sensor data with spatial, temporal, and thematic annotations to enable high-level reasoning. In this paper, we explore how abductive reasoning framework can benefit formalization and interpretation of sensor data to garner situation awareness. Specifically, we show how abductive logic programming techniques, in conjunction with symbolic knowledge rules, can be used to detect inconsistent sensor data and to generate human accessible description of the state of the world from consistent subset of the sensor data. We also show how trust/belief information can be incorporated into the interpreter to enhance reliability. For concreteness, we formalize Weather domain and develop a meta-interpreter in Prolog to explain Weather data. This preliminary work illustrates synthesis of high-level, reliable information for situation awareness by querying low-level sensor data.

KEYWORDS: Semantic Sensor Web, Situation Awareness, Knowledge Base, Abductive Reasoning, Meta-interpreter, Inconsistency, Trust/Belief.

1. INTRODUCTION

Embedded Networked Sensing involves sensors tightly coupled to the physical world, to monitor and interact with it. In order to provide suitable context of interpretation for sensor data, it is important to associate spatio-temporal-thematic information with them [1][2]. Sensor Web Enablement (SWE) [3] provides an XML-based standards approach to represent sensors, sensor data and related services, while Semantic Sensor Web [1] extends it further by applying Semantic Web technology and spatial, temporal, and thematic ontologies to represent and reason with sensor data. Recall that the Semantic Web is an evolving extension of the World Wide Web in which the semantics, or meaning, of information on the Web is formally defined [4]. These formal definitions, captured in ontologies, provide a more expressive framework to describe sensor observations and their relations to real-world situations.

According to Mica Endsley, *situation awareness* is 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 [17]. In the context of sensor data about the real world, this translates to: (1) detecting and recognizing objects and events, (2) determining how they are interrelated, and (3) predicting how things are going to change over a period of time going forward. Reconstructing the current state of the world from sensor data is akin to abductive reasoning in logic, that is, reasoning from observed effects to possible causes. Predicting the future states or course of action is akin to deduction in logic, that is, reasoning from causes to effects. Thus, situation awareness applications can benefit from symbolic representation of the state of the world, and using both abductive and deductive reasoning in a unified framework.

AFRL visionaries have verbalized their research agenda in support of Situation Awareness under the *Layered Sensing* banner as follows [5]: "Layered Sensing provides military and homeland security decision makers at all levels with timely, actionable, trusted, and relevant information necessary for situational awareness to ensure their decisions achieve the desired military/humanitarian effects. Layered Sensing is characterized by the appropriate sensor or combination of sensors/platforms, infrastructure and exploitation capabilities to generate that situation awareness and directly support delivery of 'tailored effects'."

Our goal is to investigate semantic techniques to abstract sensor data into human accessible symbolic form, with potential for having human-in-the-loop for decision making, by incorporating abductive reasoning techniques to explain observed effects and data inconsistencies in

978-1-4244-4586-8/09/$25.00 ©2009 IEEE

terms of potential causes and sensor abnormalities (faulty sensors or malicious sensors) respectively. To enhance reliability, trust/belief annotations will be introduced and manipulated. Meta-interpreter and logic programming techniques will be exploited to develop an executable specification of a suitable reasoner.

With this background on how Semantic Sensor Web, Situation Awareness, and Abductive Reasoning relate to the Layered Sensing vision, we organize the remainder of the paper as follows. Section 2 discusses general approach to using abductive reasoning with semantic sensor web for situation awareness. Specifically, it tries to delineate various orthogonal aspects, to focus on each of them separately, for clarity. Section 3 describes our prototype meta-interpreter implementation in Prolog to detect and isolate inconsistency in the weather data, and to explain consistent subset of weather phenomenon in terms of weather features. It also provides sample queries for illustrative purposes. Section 4 concludes with suggestions for future work.

2. GENERAL APPROACH

Raw sensor data can be associated with spatio-temporal information to provide context. Semantic Web technology (ontologies) can be employed to represent information in a standard, reusable way. Domain models provide a means to map low-level numeric sensor data to more meaningful symbolic form, and specify fusion of multiple, heterogeneous sensor data into human accessible abstractions. This domain knowledge can be naturally expressed in a declarative form and formalized in terms of logic rules. Furthermore, to prototype and deploy practical applications, one can encode a large subset of this knowledge in a logic programming language or a semantic web rule language, to facilitate reasoning from it. For instance, if the necessary domain knowledge can be expressed using Conditional Rules in Horn Logic, deductive reasoning from this encoding can be carried out straight forwardly by Prolog interpreter. In other words, the encoding can serve as an *executable specification* for querying. On the other hand, if the application requires abductive reasoning, meta-programming constructs provided by the logic programming infrastructure can be exploited to write *meta-interpreters* for reusing the same encoding [6][7].

The association between causes and effects, such as diseases and symptoms, faults and their manifestations, actions and their outcomes, weather features and the associated weather phenomena, etc. can potentially be encoded as declarative knowledge using conditional rules. Deductive reasoning can be used to determine/project effects from causes, and abductive reasoning can be used to explain effects/observations in terms of plausible causes/events. In the literature, first-order logic has been used to formalize deductive reasoning. For abduction, there are two popular standard approaches: fault-model based approach and design-model based approach. In fault-model based approach, rules specify how faults manifest in terms of observations. This is commonly used in the context of natural systems. A prototypical application is in medical diagnosis (through the formalization of disease-symptom knowledge) [8]. In design-model based approach, rules specify the expected normal behavior in terms of the designed behavior (constraints) of the various components. This is commonly used in the context of artificially engineered systems. A prototypical application is in digital circuit diagnosis. Logical inconsistency between the expected/predicted behavior and the observed behavior implies abnormal component [9]. The abductive approach to perception formalized/specified in an extension of first-order logic in [10] resembles the design-model based approach.

In our domain of discourse, consequences of deductive reasoning are *deterministic* in nature. On the other hand, abductive reasoning usually generates multiple explanations for observations. This is in part due to *incomplete* knowledge about the state of the world. However, one can deductively reason from plausible explanations, and on that basis, actively seek to verify projected/predicted effects of these explanations through additional observations, to pare it down to a manageable set [10]. In other words, there are benefits to integrating deduction and abduction using meta-interpreters by attempting to formalize generation of “possible suspects”, “probable suspects”, and eventually using suitable “exoneration criteria” to determine “the guilty”. The feasibility and the efficacy of this agenda depend on how readily we can capture the knowledge in symbolic terms, and how well matched or tailorable the control strategy implemented by the logic programming system is with respect to what may be needed.

In sensors domain, observations in the form of heterogeneous data (for example, weather phenomena such as temperature, humidity, pressure, wind speed, etc), in a region of space over a duration of time, can be explained by hypothesizing possible events (for example, weather features such as snow storm, blizzard, flurry, icy condition, etc). Furthermore, if multiple sensor motes carrying homogeneous sensors have been deployed in close proximity, one can expect their readings to be consistent (that is, within some tolerance due to temporal and spatial locality). In other words, inconsistency in sensor observations within a small neighborhood devoid

of any abnormal events signals either *faulty* or *malicious* sensor. (Here we are unable to distinguish between these two situations.) In the presence of such abnormality, sensor data can be filtered to obtain a consistent subset of observations, before generating explanations. (Alternatively, one can explain maximally consistent subsets of observations.) For example, a *low temperature* and *precipitation* data can be explained by *snowfall*, while additional observation involving *windspeed* may enable us to refine this explanation further to a *blizzard* or a *flurry*. *High windspeed* with *snowfall precipitation* and *low temperature*, measured by the respective normal sensors in a region at some time, can be accounted for by asserting *blizzard* condition. In general, additional consequences of a hypothesis can be checked, to confirm or contradict the hypothesis. If *blizzard* hypothesis is not corroborated by neighboring sensors, we may revise the hypothesis to abnormality in sensors, requiring additional probing and diagnosis.

Not all sensor data can be relied upon equally. To reflect its dependence on sensor hardware reliability factors, sensor's propensity to be maliciously manipulable, effects of environmental factors, etc., we can explicitly incorporate and manipulate trust and belief values associated with sensors and sensor data. In this preliminary report, we assume that a variable trust value associated with a sensor manifests itself as a fixed belief value associated with its observations, and is given to us explicitly. We use these belief values to filter sensor data to be explained.

In the next section, we illustrate the role of abductive reasoning and semantic sensor web for situational awareness and diagnosis, by showing how the above informal analysis of weather data can be formalized in Prolog.

3. ABDUCTIVE LOGIC PROGRAM

We use a simple example from weather domain to systematically motivate the development of a Prolog meta-interpreter to encode and explain sensor data, and isolate inconsistencies in the sensor data. We ignore standardization issues such as the use of semantic web formalism and ontologies for encoding knowledge, to focus on the reasoning aspects, which is orthogonal.

At the lowest-level, we abstract numeric sensor data into more meaningful symbolic form using the following mapping.

freezing:	*temperature <= 32F*
high-winds:	*windspeed >= 30mph*
low-winds:	*windspeed < 30mph*
high-visibility:	*visibility >= 1 mile*
med-visibility:	*¼ mile < visibility < 1 mile*
low-visibility:	*visibility <= ¼ mile*

The abstracted observations are specified in Prolog as facts involving the ternary predicate *obs(ObservationId, WeatherPhenomenonAbstraction, Time).* Without loss of generality, we assume that all the readings are from the same location. (Otherwise, we need to add an additional space parameter and handle it similarly to how the time parameter is handled.)

```
obs(o1, freezing, t1).
obs(o2, high-winds, t1).
obs(o3, low-winds, t1).
obs(o4, freezing, t2).
obs(o5, low-winds, t2).
```

Belief values can be associated with these observations, using annotated literals, by introducing a new "double colon" infix operator.

```
:- op(700, xfx, ::).

obs(o1, freezing, t1)::70
obs(o2, high-winds, t1)::50
obs(o3, low-winds, t1)::20
obs(o4, freezing, t2)::40
obs(o5, low-winds, t2)::40
```

The fragment of the weather domain model that associates observed weather phenomena with its causative weather feature can be formalized using conditional rules. A word about Prolog notation: the formal parameters that begin with an uppercase letter, such as O and T, are universally quantified logic variables, while the remaining formal parameters that begin with a lower case letter are symbolic constants. The role of belief values is to enable filtering of observations to be explained, and is orthogonal to the domain model being captured here.

```
obs(O, freezing, T)       :- feature(blizzard, T)
obs(O, high-winds, T)     :- feature(blizzard, T)
obs(O, freezing, T)       :- feature(flurry, T)
obs(O, freezing, T)       :- feature(flurry, T)
```

Weather phenomena are explained by hypothesizing weather feature(s). (Note that the specific example shown can be easily dealt with using the simpler Parsimonious Covering Theory [8] but the technique we are exploring has much wider applicability.) The predicates that can be used in an explanation are called *abducibles*. The meta-interpreter is permitted to assume abducible facts to account for the observations, if it cannot deduce the observations otherwise. Typically, abducibles do not

have any facts or rules associated with them. Note that in Prolog, an underscore used as a formal parameter is a wildcard and represents a variable that can be bound to any value.

```
abduct(feature(_ , _)).
```

Prolog does not support logical negation. In order to specify that two facts cannot co-occur without jeopardizing consistency, we introduce additional integrity constraints. In the weather domain, it is easy to list several such constraints:

```
precipitation = snow  => visibility != high
precipitation = snow  => temperature = freezing
windspeed = high  <=> windspeed != low
```

In Prolog encoding, a special proposition *cfail* is derived from inconsistent literals, by making the latter imply the former through explicit conditional rules such as:

```
cfail :- obs(_, high-winds, T), obs(_, low-winds, T).
```

In a more realistic setting, we would bring in tolerances on time values (and space values), rather than require the time values (and space values) in various contradictory literals to be identical.

To develop an abductive meta-interpreter, we begin with an axiomatization of the built-in meta-predicate *call/1* that uses the binary meta-predicate *clause/2* to access internal representation of a logic program. (Recall that a logic program is a sequence (disjunction) of facts and rules, where each rule has a head and a body, which in turn is made up of a sequence (conjunction) of literals [6][7]).

```
call((G1,G2)) :- call(G1), call(G2).
call(true).
call(G) :- clause(G,B), call(B).
```

In order to handle abducibles and ordinary defined predicates appropriately and efficiently (and ignore built-in predicates, for simplicity), we propose the following abductive meta-interpreter definition (involving cuts) with signature *ami(Goals, AbduciblesSoFar, AugmentedAbduciblesSoFar).* In the process of demonstrating an observation, if an (unproven) abducible is encountered, the abducible fact is augmented to the explanation being constructed.

```
ami((G1,G2), InAbd, OutAbd) :-
        !, ami(G1, InAbd, T), ami(G2, T, OutAbd).
ami(true, InAbd, InAbd) :- !.
ami(G, InAbd, OutAbd) :- abduct(G), !,
        (member(G,InAbd)
            -> OutAbd = InAbd ;
              OutAbd = [G | InAbd]).
ami(G, InAbd, OutAbd) :-
        clause(G,B), ami(B, InAbd, OutAbd).
```

The top-level query to explain Observations is:

```
?- ami(Observations, [], AnExplanation).
```

To ensure that an explanation is consistent, that is, it does not contain features that cannot co-exist, we define a variant of the above code that tries to prove the distinguished proposition *cfail* from the generated explanation, where \+ represents negation by failure operator.

```
explain(G,E) :- ami(G, [], E), consistent(E).

consistent(EL) :- \+ proveFrom(cfail, EL).

proveFrom((G1,G2), EL) :-
        !, proveFrom(G1, EL), proveFrom(G2, EL).
proveFrom(true, _) :- !.
proveFrom(G, EL) :- abduct(G), !, member(G, EL).
proveFrom(G, EL) :- clause(G,B), proveFrom(B, EL).
```

Each observation can be enriched with a belief value, and the meta-interpreter can be supplied with a belief threshold to support filtering of sensor data, and obtain reliable observations that deserve explaining. Furthermore, notice that this modification can be done in an incremental fashion by defining a ternary predicate *explainObsBelLst/3* that takes as input a list of observations annotated with belief value and a belief threshold to be crossed, and returns multiple explanations on backtracking.

```
explainObsBelLst(OBL, Threshold, E)  :-
filter(OBL, Threshold, G), !, explain(G, E).

filter([],_,true).
filter([Ob::B | T],Th,(Ob,T2)) :-
                        B >= Th, !, filter(T,Th,T2).
filter([_::B | T],Th, T2)   :- B < Th,  !, filter(T,Th,T2).
```

For completeness, we include definitions of auxiliary coercion predicates used in the final meta-interpreter below: *filterList/3* maps an observation-belief list to a smaller list containing observation-belief pairs, each with belief value over the threshold, *lstStrc/2* maps a list to its equivalent comma separated structure, and *stripBelief/2* eliminates the belief-value from the observation-belief pair.

```
filterList([],_,[]).
filterList([Ob::B | T],Th,[Ob|T2]) :-
                    B >= Th, !, filterList(T,Th,T2).
filterList([_::B | T],Th, T2)   :-
                    B < Th,  !, filterList(T,Th,T2).

lstStrc([],true).
lstStrc([H|T],(H,T2)) :- lstStrc(T,T2).

stripBelief([],[]).
stripBelief([Ob::_|T],[Ob|T2]) :- stripBelief(T,T2).
```

To detect and diagnose inconsistency in sensor data, we generalize the binary predicate *proveFrom/2* to a 4-ary predicate *proveFrom/4* that further computes the observations causing the integrity constraint violations. The role of the four parameters can be summarized as follows: *proveFrom (FailureGoal, ObsLst, ConflictingInObs, ConflictingOutObs).*

```
proveFrom((G1,G2), Obs, IObs, OObs) :-
          proveFrom(G1, Obs, IObs, T),
          proveFrom(G2, Obs, T, OObs).
proveFrom(true, _, IObs, IObs).
proveFrom(obs(A,B,C), Obs, IObs,
                    [obs(A,B,C) | IObs]) :-
          member(obs(A,B,C),Obs).
proveFrom(G, Obs, IObs, OObs) :-
          \+ (G = obs(_,_,_)),
          \+ (G = (_,_)), \+ (G = true),
           clause(G,B), proveFrom(B, Obs, IObs, OObs).
```

All inconsistent subsets of observations can be determined by brute-force search. However, in future, indexing schemes may have to be developed to locate "needles in the hay stack" efficiently. The signature of the predicate to find inconsistent set of observations is *inconsistency(ObservationList, AllInconsistencies).*

```
inconsistency(OL, AllIc) :-
          setof(Ic, proveFrom(cfail,OL,[],Ic), AllIc).
inconsistency(OL,[]) :-
          \+ proveFrom(cfail, OL, [], _).
```

Note that *Ic* is bound to *[]* if and only if *OL* is bound to a consistent value. Also, this predicate always succeeds eventually.

We now define our final predicate *explanationPlusInconsistency/4* that takes an observation-belief list and a belief threshold, and returns an explanation of the consistent subset of observations and the (remaining) inconsistent subset of observations. That is, the signature of the predicate is

```
explanationPlusInconsistency( ObservationBeliefList,
          BeliefThreshold,        AnExplanation,
          AllInconsistencies).
```

All inconsistencies are eliminated before an explanation is generated. (Alternatively, one can generate explanations for maximally consistent subsets of observations.)

```
explanationPlusInconsistency(OBL,BT,E,AllIc) :-
          filterList(OBL,BT,OL),
           inconsistency(OL,AllIc),  append(AllIc, Ic),
          subtract(OL, Ic, GL),  lstStrc(GL,G),
          explain(G, E).
```

To illustrate the workings of the meta-interpreter, we provide a collection of queries and expected responses.

Query1: Explain a single freezing observation, at time *t*.

```
?- ami(obs(o3, freezing, t), [], E).

E = [feature(blizzard, t)] ;
E = [feature(flurry, t)] ;
```

Query 2: Explain a freezing observation, at time *t1*, and a low-winds observation, at time *t2*.

```
?- ami((obs(o1, freezing, t1), obs(o4, low-winds, t2)),
[], E).

E = [feature(flurry, t2), feature(blizzard, t1)] ;
E = [feature(flurry, t2), feature(flurry, t1)] ;
```

Query 3 illustrates that further subsumption reasoning is required to get minimal explanations.

Query 3: Explain a freezing observation and a high-winds observation, both at time *t*.

```
?- ami((obs(o1, freezing, t),obs(o2, high-winds, t)), [], E).

E = [feature(blizzard, t)] ;
E = [feature(blizzard, t), feature(flurry, t)] ;
```

Query 4 involves inconsistent observations.

Query 4: Explain a freezing observation, high-winds observation, and low-winds observation, all at time *t*.

```
?- explain((obs(o1, freezing, t), (obs(o1,high-winds, t),
obs(o2, low-winds, t))), E).

false.
```

Queries 5 and 6 illustrate the uses of observations annotated with belief values, belief threshold, and the Prolog primitive needed to collect all explanations in one place.

Query 5: Explain and group a freezing observation, high-winds observation, and low-winds observation, all at time *t*, with a belief threshold of 50.

?-setof(E,explainObsBelLst([obs(o1, freezing, t)::70, obs(o1,high-winds, t)::50, obs(o2, low-winds, t)::30], 50, E), All).

All = [[feature(blizzard, t)]].

Query 6: Explain a freezing observation, at time *t1*, high-winds observation, at time *t2*, low-winds observation, at time *t3*, with a belief threshold of *10*.

?- explainObsBelLst([obs(o1, freezing, t1)::70, obs(o1,high-winds, t2)::50, obs(o2, low-winds, t3)::30], 10, E).

E = [feature(flurry, t3), feature(blizzard, t2), feature(blizzard, t1)] ;

E = [feature(flurry, t3), feature(blizzard, t2), feature(flurry, t1)] ;

The following queries show how reasoning from sensor data can uncover inconsistent subset of sensor data (whose provenance can ultimately enable "tracking the culprit sensors") and provide higher-level situation awareness (in terms of weather features in this scenario). Recall also that our example abstracts from spatial information, but the example can be easily generalized to use location information, similarly to the treatment of time.

Query 7: Find inconsistency from a freezing observation, high-winds observation, low-winds observation, all at time *t1*, and another low-winds observation, at time *t2*.

?- inconsistency([obs(o1, freezing, t1), obs(o1,high-winds, t1), obs(o2, low-winds, t1), obs(o3, low-winds, t2)], Ic).

Ic = [[obs(o2, low-winds, t1), obs(o1, high-winds, t1)]].

Query 8: Explain (and find inconsistency from) a freezing observation, high-winds observation, low-winds observation, all at time *t1*, and a low-winds observation, at time *t2*, and a belief threshold of *25*.

?- explanationPlusInconsistency([obs(o1, freezing, t1)::70, obs(o1,high-winds, t1)::50, obs(o2, low-winds, t1)::40, obs(o3, low-winds, t2)::20], 25, E, I).

E = [feature(blizzard, t1)],
I = [obs(o2, low-winds, t1), obs(o1, high-winds, t1)] ;

E = [feature(flurry, t1)],
I = [obs(o2, low-winds, t1), obs(o1, high-winds, t1)] ;

Query 9: Explain (and find inconsistency from) a freezing observation, high-winds observation, both at time *t1*, and a low-winds observation, at time *t2*, with a belief threshold of *25*.

?- explanationPlusInconsistency([obs(o1, freezing, t1)::70, obs(o1, high-winds, t1)::50, obs(o2, low-winds, t2)::50], 25, E, I).

E = [feature(flurry, t2), feature(blizzard, t1)],
I = [] ;

Query 10: Explain (and find inconsistency from) a freezing observation, high-winds observation, low-winds observation, all at time *t1*, and a freezing observation, high-winds observation, low-winds observation, all at time *t2*, with a belief threshold of *30*.

?- explanationPlusInconsistency([obs(o1, freezing, t1)::70, obs(o2, high-winds, t1)::50, obs(o3, low-winds, t1)::20, obs(o4, freezing, t2)::40, obs(o5, high-winds, t2)::70, obs(o6, low-winds, t2)::60], 30, E, I).

E = [feature(blizzard, t2), feature(blizzard, t1)],
I = [[obs(o6, low-winds, t2), obs(o5, high-winds, t2)]] ;

E = [feature(flurry, t2), feature(blizzard, t1)],
I = [[obs(o6, low-winds, t2), obs(o5, high-winds, t2)]] ;

Query 11 illustrates how the above explanations can be grouped together straightforwardly.

Query 11: Explain and group (and find inconsistency from) a freezing observation, high-winds observation, low-winds observation, all at time *t1*, and a freezing observation, high-winds observation, low-winds observation, all at time *t2*, with a belief threshold of *30*.

?- setof(E, explanationPlusInconsistency([obs(o1, freezing, t1)::70, obs(o2, high-winds, t1)::50, obs(o3, low-winds, t1)::20, obs(o4, freezing, t2)::40, obs(o5, high-winds, t2)::70, obs(o6, low-winds, t2)::60], 30, E, I), All).

I = [obs(o6, low-winds, t2), obs(o5, high-winds, t2)],

All = [[feature(blizzard, t2), feature(blizzard, t1)], [feature(flurry, t2), feature(blizzard, t1)]].

4. CONCLUSIONS AND FUTURE WORK

We illustrated the role of abductive reasoning and semantic sensor web for situational awareness and diagnosis, by showing how weather sensor data can be represented and reasoned with in a logic programming language such as Prolog. Specifically, we focused on the core issues related to reasoning using a meta-interpreter based approach.

In the long run, this work can be extended in a number of different directions to improve its expressive power, effectiveness, efficiency, and standardization potential.

The abductive framework enables generating hypotheses from incomplete information, and can be generalized to seek additional information to resolve ambiguities and inconsistencies. *Active perception* [10] involves looking for "confirmatory tests" and "discriminating observations" to select likely explanation(s) from the set of all possible explanations. The research question is how to represent the relevant information and orchestrate the reasoning steps. Abductive formulation seems to provide a more natural setting for advanced situational awareness applications because the latter requires constructing possible states of the world from available observations. However, we need to research how to structure the space of explanations in order to make them more manageable, possibly through ranking. This may also require integrating symbolic and numeric/probabilistic information. In general, bringing human-in-the-loop can help determine unlikely explanations and supply missing inputs to fix the problem.

When trying to design algorithms for practical problems, it is often the case that the general task is computationally intractable, but one can isolate reasonable restrictions for which efficient algorithms are feasible. In general, abductive reasoning is provably NP-hard [19]. So, to restore tractability, expressive power of the representational formalism must be limited. In other words, we need to identify realistic, commonly occurring, easy cases of interest, and develop specialized data structures, explanation ranking, filtering, and reasoning strategies. For instance, in medical domain, parsimonious covering theory uses bipartite graphs relating symptoms and diseases [8]. In this context, logic-based formalization serves as a reference specification (correctness criteria).

To improve efficiency of abductive reasoning in logic programming framework, we need to explore and adapt existing query optimization techniques. One can combine the benefits of top-down (logic programming and AI perspective) and bottom-up (database perspective) approaches, by using strategies such as *tabling* and *magic sets*. Recall that tabling enables caching of intermediate results that solves the incompleteness problem (looping on left-recursion) in top-down evaluators, and magic sets enables query bindings to be propagated to the base relations (pushing selects through joins) in bottom-up evaluators. In light of expressive power – efficiency tradeoffs, one can explore techniques to compile AI-style queries to be executed using a DBMS rather than the native interpreter [20].

For large scale deployments of sensor networks, spatio-temporal information associated with sensor data may provide a natural way of *indexing* and *partitioning* sensor data. Distributed programming model such as Map-Reduce paradigm [11] and distributed computing infrastructure such as Cluster/Cloud [12] can be exploited to process such independent data islands using intrinsically parallel computations.

In order to standardize sensor data processing, we will eventually need to exploit semantic web technology. Specifically, the annotated sensor data and its context needs to be rendered in RDF and OWL format using spatial, temporal and domain-specific ontologies [1][2]. To provide a standard interface, this annotated sensor data should be made accessible through a semantically enabled Sensor Observation Service, as defined by the OGC Sensor Web Enablement [18]. The reasoning strategy we have developed needs to be reimplemented in an inference engine that can process semantically enriched XML-based syntax [21]. Another approach would be to translate data encoded in a Semantic Web format, such as the Resource Description Format (RDF) [13] or the Web Ontology Language (OWL) [14], into Prolog syntax. Currently there are several tools for performing this translation, including the SWI-Prolog Semantic Web Library [15] and Thea [16], an OWL library for SWI-Prolog. There are also rule-based RDF query languages such as RDFLog that can be implemented using efficient logic programming techniques discussed earlier [22].

Sensor data has been associated with fixed belief values in the example shown. In general, the belief values need to be computed dynamically to reflect dependence on sensor hardware reliability (sensor quality, measurement tolerance, degradation over time, etc.), quantifiable environmental factors, qualitative information about potential malicious behavior, etc. as a function of time. There is nothing intrinsically difficult about incorporating

this information into our framework if trust values associated with various sensors can be modeled and quantified as a function of time, and belief values associated with its observations can be formally specified. Similarly, it will be necessary to compose trust/belief values associated with primitive, heterogeneous data, to obtain belief values associated with the aggregated/inferred feature values. Once we have a good handle on how to deal with these prototypical scenarios satisfactorily, we will explore evaluation of such approaches for different sensor data domains.

REFERENCES

[1] A. Sheth, C. Henson, and S.Sahoo, "Semantic Sensor Web," In: IEEE Internet Computing, July/August 2008, pp. 78-83.

[2] K. Thirunarayan, and J. K. Pschorr, "Semantic Information and Sensor Networks", In: Proceedings of the 24th Annual ACM Symposium on Applied Computing (ACM SAC 2009), March 2009.

[3] M. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC Sensor Web Enablement: Overview and High Level Architecture (OGC 07-165)," Open Geospatial Consortium white paper, 28 Dec. 2007.

[4] W3C Semantic Web Activity, http://www.w3.org/2001/sw/

[5] M. Bryant, Paul Johnson, Brian Kent, Michael Nowak, and Steve Rogers, "Layered Sensing", May 2008. http://www.wpafb.af.mil/shared/media/document/AFD-080820-005.pdf

[6] Y. Shoham, ARTIFICIAL INTELLIGENCE TECHNIQUES IN PROLOG, Morgan Kaufmann Publishers, 1994.

[7] I. Bratko, PROLOG PROGRAMMING FOR ARTIFICIAL INTELLIGENCE, Third edition. Pearson Education, Addison-Wesley, 2001.

[8] Y. Peng and J. A. Reggia, Abductive Inference Models for Diagnostic Problem-Solving, Springer-Verlag, 1990.

[9] W. Hamscher, L. Console, and J. deKleer, Readings in Model-Based Diagnosis, Morgan Kaufmann, 1992.

[10] M.P.Shanahan, "Perception as Abduction: Turning Sensor Data into Meaningful Representation", Cognitive Science, vol. 29, 2005, pp. 103-134.

[11] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", Commun. ACM, Vol. 51, No. 1., January 2008, pp. 107-113.

[12] Hadoop, http://hadoop.apache.org/core/

[13] Resource Description Framework (RDF), http://www.w3.org/TR/rdf-concepts/

[14] Web Ontology Language (OWL), http://www.w3.org/TR/owl-ref/

[15] J. Wielemaker, J., SWI-Prolog Semantic Web Library, 2005. http://www.swiprolog.org/packages/semweb.html

[16] V. Vassiliadis, Thea. A web ontology language - OWL library for [SWI] Prolog. Web-published manual, 2006, http://www.semanticweb.gr/TheaOWLLib/.

[17] M. R. Endsley, Toward a theory of situation awareness in dynamic systems. Human Factors 37(1), 1995, pp. 32-64.

[18] C. Henson, J. Pschorr, A. Sheth, and K. Thirunarayan, "SemSOS: Semantic Sensor Observation Service," International Symposium on Collaborative Technologies and Systems (CTS2009), Workshop on Sensor Web Enablement (SWE2009), Baltimore, Maryland, 2009.

[19] T. Bylander, D. Allemang, M. Tanner, J. Josephson, "*The computational complexity of abduction*," Artificial Intelligence, Volume 49, Issue 1-3, 1991, pp. 25-60.

[20] A. Sheth and A. O'Hare, "The Architecture of BRAID: A System for Bridging Al/DB Systems," Proceedings of the 7th IEEE Intl. Conference on Data Engineering, Kobe, Japan, April 1991, pp. 570-581.

[21] J. Almendros-Jimenez, A. Becerra-Teron, F. Enciso-Banos, "*Magic Sets for the XPath Language*," Journal of Universal Computer Science, Vol. 12, No. 11, 2006, pp. 1651-1678.

[22] F. Bry, T. Furche, C. Ley, B. Linse and B. Marnette, "Taming Existence in RDF Querying," Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems, LNCS Vol. 5341, 2008, pp. 236-237.