# 2

# FORMALIZATION OF SITUATION AWARENESS

Kenneth Baclawski
*College of Computer Science*
*Northeastern University*
*Boston, Massachusetts*
Ken@Baclawski.com

Mieczyslaw K. Kokar
*Electrical and Computer Engineering*
*Northeastern University*
*Boston, Massachusetts*
Kokar@coe.neu.edu

Christopher J. Matheus
*Versatile Information Systems*
*Framingham, Massachusetts*
chris@matheus.com

Jerzy Letkowski
*Western New England College*
*Springfield, Massachusetts*
jletkows@wnec.edu

Marek Malczewski
*Composable Logic*
*Nashua, New Hampshire*
marekmal@coe.neu.edu

**Abstract**

Situation awareness means simply that one knows what is going on around oneself. In operational terms, this means that one knows the information that is relevant to a task. Maintaining a coherent awareness of the situation is essential to successful task completion. We propose a formal basis for situation awareness that draws on sources and makes use of techniques from the logic, human-computer interaction and data fusion communities. Our framework includes formalizations of the data fusion process as well as the notion of a situation. We express our formalization using various languages, including UML, DAML and the Slang formal methods language, each of which has its own unique contribution to our framework.

## 1 INTRODUCTION

Maintaining a coherent situation awareness (SAW) concerning all units operating in an area of interest (battlefield, emergency situation, anti-terrorism

campaign, and so on) is essential for achieving success. The process of achieving SAW is called Situation Analysis. The basis for SAW is a knowledge of the objects that are located within a given region. Considerable effort has been expended on this problem by the Data Fusion community, and effective hardware and software are now available.

Although a knowledge of the objects is essential, it does not, by itself, constitute SAW. It is also necessary to know all the relations among the objects that are relevant to the current operation. In many ways this is a more difficult problem than determining the objects. While the number of objects may be large, the number scales linearly with the size of the region. The same cannot be said for relations, whose possibilities increase exponentially as the number of objects increases.

The SAW problem is further complicated by the fact that some information that is input to a SAW system can be in many different formats (sensory inputs, text, intelligence) and that the exact structure of the information cannot be predicted at the design time of the SAW system (i.e., we don't quite know which pieces of information will be available at a particular time instant). The goals of SAW can also change at any time and the SAW system needs to be informed about this, too. To address this kind of a problem a SAW system must be able to incorporate into its processing information about types of objects, their features, rules for recognizing relations and intelligence information. Furthermore, it has to accomplish this at run-time.

The goal of this research is to develop a formal basis for Situation Awareness using a formal methods system, Specware, and the DAML+OIL ontology language. Eventually, we plan to show how relevant symbolic information can be conveyed to a Situation Awareness system and what can be inferred based upon this input.

In the remainder of this paper we first give some background of the notions of situation semantics and SAW in Section 2. We then describe our approach to the formalization of SAW in Section 3. This section also discusses the formalization languages that we use, and it gives the rationale for their use. The general abstract formalization of situation awareness is introduced in Section 4. The full formalization is much too large for this paper, so we only give an overview of it using UML. However, to illustrate the process of formalization, we introduce a much simpler example of a situation by using family relationships in Section 5. The family relationship situation is represented using all three languages.

## 2 BACKGROUND

A number of philosophers and logicians introduced concepts similar to that of a situation, including von Mises [von97] in 1949 and Bunge [Bun77] in the 1970s. They observed that an open system is not fixed and possibly not com-

pletely available. Therefore, the outcome of an action in the context of an open system is always uncertain. However, the earliest formal notion of *situation* (although not situation awareness) was introduced by Barwise as a means of giving a more realistic formal semantics for speech acts than what was then available [Bar81]. In contrast with a "world" which determines the value of every proposition, a situation corresponds to the limited parts of reality we perceive, reason about, and live in. A situation will determine answers in some cases, but not all. Furthermore, in situation semantics, basic properties, relations, events and even situations are reified as objects [Bar89]. While Barwise's situation semantics is only one of the many alternative semantic frameworks that are currently available, its basic themes have been incorporated into most frameworks.

The specific term *situation awareness* is most commonly used by the Human-Computer Interaction (HCI) community (cf. [EG00]). The concerns of this community are to design computer interfaces so that a human operator can achieve SAW in a timely fashion. From this point of view, SAW occurs in the mind of the operator. In almost any fairly complex system, such as military aircraft and nuclear reactors, manual tasks are being replaced by automated functions. However, human operators are still responsible for managing SAW. This raises new kinds of problems due to human limitations in maintaining SAW. The SAW literature gives many examples of incidents and accidents, which could have been avoided if operators had recognized the situation in time. These problems can be categorized [BZSF96] as follows:

- System data problems - data may be presented in such a way that it is hard to reason upon (spread across many displays, presented with great deal of details that makes impossible for human to process it) or may be hidden within the automated functions.
- Human limitations - humans make errors because of a lack of concentration due to interruptions or a heavy workload.
- Time-related problems - problems may not occur instantly but they may arise for a long period of time, in which case it is hard to determine that the system is in dangerous state. Often systems are constantly changing, so that it may be hard to decide which pieces of data are important and which need to be analyzed.

For more complicated scenarios, it is apparent one can no longer rely on human operators to perform the entire SAW task alone. Some form of knowledge management assistance is necessary.

Situation awareness is also used in the data fusion community (except that they call it *situation assessment*). Data fusion is an increasingly important element of diverse military and commercial systems. It uses overlapping information to detect, identify and track relevant objects in a region. The term

| Data Fusion Level | Association Process | Estimation | Entity Estimated |
|---|---|---|---|
| L.0–Sub-Object Assessment | Assignment | Detection | Signal |
| L.1–Object Assessment | | Attribution | Physical Object |
| L.2–Situation Assessment | Aggregation | Relation | Aggregation (Situation) |
| L.3–Impact Assessment | | Plan Interaction | Effect |
| L.4–Process Refinement | Planning | (Control) | (Action) |

Table 1: Characterization of Data Fusion Levels

"data fusion" is used because information originates from multiple sources. More succinctly, data fusion is the process of combining data to refine state estimates and predictions [SBW99].

The terminology of data fusion has been standardized by the Joint Directors of Laboratories (JDL) Data Fusion Group, and this group maintains a Data Fusion Model. In this model, data fusion is divided into 5 levels as shown in Table 1. Note that SAW is Level 2 data fusion in this model. The JDL model defines SAW to be the "estimation and prediction of relations among entities, to include force structure and cross force relations, communications and perceptual influences, physical context, etc." Level 2 processing typically "involves associating tracks (i.e., hypothesized entities) into aggregations. The state of the aggregate is represented as a network of relations among its elements. We admit any variety of relations to be considered – physical, organizational, informational, perceptual – as appropriate to the given need." The table and all quotations in this paragraph are from [SBW99].

In our formalization we will make use of elements of all three of the frameworks mentioned above (i.e., Logic, HCI and JDL), although we will emphasize the terminology and point of view of the JDL model.

# 3 FORMALIZATION PROCESS

It would be wonderful if there was a single formal methods framework and language that would be the "best practice" for every possible use. Unfortunately, the reality is that different frameworks and languages are necessary because each one has features and advantages that are only available for it. The following are the languages we used in formalizing SAW along with some of their features:

- Unified Modeling Language (UML) [BJR00]
  - A "best practice" graphical representation.
  - Widely adopted in industry and academics.
  - Supported by mature CASE tools.
  - Open standard maintained by the OMG.
  - The semantics is not yet formally specified.
- DARPA Agent Markup Language (DAML) [DAM01]
  - Emerging Web-based interchange format.
  - Logic-based and formally specified semantics.
  - Designed for ontologies and annotations.
  - Does not yet include rules.
- Specware Formal Methods System [W$^+$98]
  - System for formal specifications. The Specware language is called Slang.
  - Supports theory management and refinement via category theory (colimit operation).
  - Integrated with a theorem prover (SNARK) [SNA02, SWC02].

The graphical representation of UML is the most important feature that we use, but the other features are also useful. However, UML is not logic-based, so it is necessary to introduce at least one other language. DAML is not only logic-based and formal, but it is also an emerging Web-based interchange format.

Another important feature of DAML that is not shared by UML is the notion of *monotonicity*. A logical system is monotonic if adding new facts can never cause previous facts to be falsified. Of course, one must be careful to define which facts are being considered in this process so that it makes sense. DAML (or more precisely the logical system within which DAML is defined) is monotonic: asserting a new fact can never cause a previously known fact to become false.

By contrast, UML and other OO systems are typically not monotonic. There are many forms of nonmonotonic logic, but the one that is closest to UML and OO systems is a logic that assumes a *closed world*. A simple example can illustrate how monotonicity affects inference. Suppose that one specifies that every person must have a father. Consider what would happen if no father was specified for a particular person object. In UML this situation would be considered to be a violation of the requirement that every person must have a father, and a suitable error message would be generated. In a monotonic

logic, on the other hand, one cannot make any such conclusion. The person who appears not to have a father really does have one, it just isn't known who he is.

As discussed above, situations generally represent only a partial knowledge of the world. In particular, this means that one might not know who everyone's father is. Accordingly, the monotonic logic used by DAML is more appropriate for SAW than the closed world assumption used in UML.

Developing automated translators between these languages can be challenging. For an analysis of the problem of translating between UML and DAML, including a discussion of the issue of monotonicity, see [BKK+02].

One feature that is needed for situation awareness is the ability to derive relations based upon knowledge about objects. This requires the use of rules. At this point DAML does not have rules, although an effort to include rules in the language is underway. To accommodate this requirement, we use a formal method language Slang. In Slang rules are represented as axioms. Consequently, rules can be used by a Slang-aware theorem prover in its reasoning process.

# 4 FORMALIZATION OF SITUATION AWARENESS

The top level formalization of situation awareness is shown in Figure 1. A *situation* consists of a collection of *situational objects*, a set of *relation evolutions* or *streams* that capture relations among objects over time, and perhaps some *goals* that define what the user is interested in achieving. Note that the situation is itself a situational object permitting reasoning about situations themselves.

*Physical objects* are situational objects that have *region evolutions*. A region evolution defines the physical location and space occupied by an object at any point in time. The subclasses of physical objects depend upon the domain. In the diagram we depict a military domain where the physical objects are made up of military units (e.g. platoons, tanks, observation posts) and obstacles (e.g. minefields, rivers, trenches) which we have split, for convenience, into two separate ontologies. Similarly, relation evolutions depend upon the domain. In a military scenario these would be relations such as `firingAt(x,y)` or `advancingTowards(x,y)`.

The evolution of objects and relations is typically defined by using *events* rather than by ordered sequences of objects and relations. The idea is that one does not have to represent entire object or relation streams. It is enough to represent those parts of a situation that have changed in a manner that is not predictable. For example, most vehicles follow tracks that can be defined by a series of way-points where the vehicle stops, starts or changes its direction.

The formalization, as expressed in Slang, is built progressively starting with basic generic specifications such as ordered sets and proceeding through more

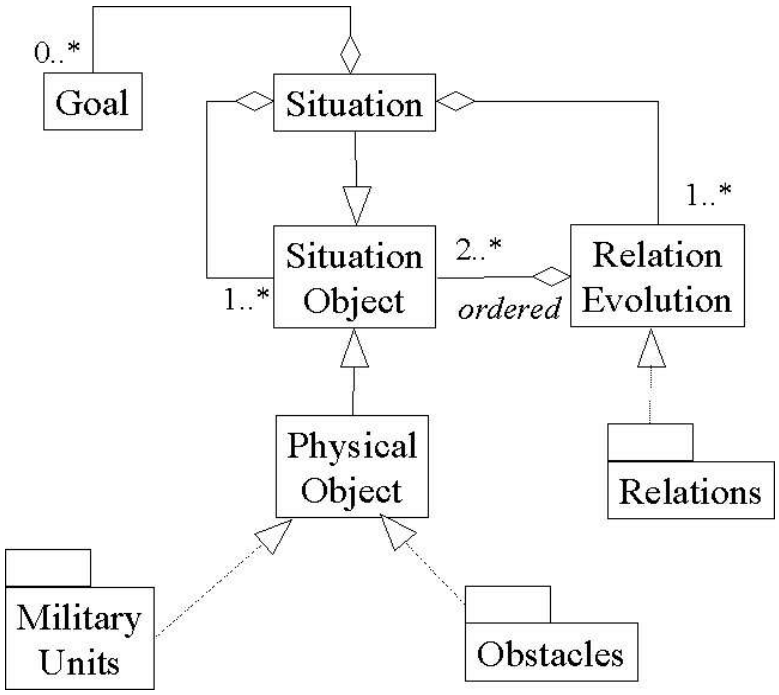


Figure 1: The Top Level Formalization of Situation Awareness

specific specifications. The colimit operation is used to construct specifications in a modular fashion. The following are the main stages:

1. Basic specs: Order, Reals, Attributes. These are mostly well known, so they were not shown.
2. Physical reality: Time, Location. These are also well known, so only abbreviated specifications are shown.
3. Streams: Sensors, Features, Objects, Relations. These are built using colimits based on a general specification for a stream.
4. Extraction: Computing one kind of stream from another kind.
5. Fusion: Merging two or more streams into one stream.
6. Situations: Various streams are selected based on a task or goal.

We focus on the later stages (higher levels) in this paper. For more on the formalization of data fusion see the work of Kokar and his colleagues [KW02b, KW02a, KTW01, KK01, KTW99, KBF00].

The Slang language is based on sorts and operations on sorts. Subclasses (or more precisely, subsorts) are specified using unary operations, and relations are specified using binary (or more generally n-ary) operations. Constraints are specified using axioms written in first-order predicate logic.

Level 1 data fusion is a process whereby sensor *measurements* are processed and combined (not necessarily in this order) to determine *objects* located in *space* and *time*. Sensor measurements are usually discrete and therefore only accurate to the extent of the rate at which they are performed. Generally objects will have a finite extent both in space and time (i.e., an object will occupy a region in space and will exist for a period of time). Objects also have features, and these features can also vary in time. The variation of an object in time is called its *region evolution*. One can also view time (and space) variation as a sequence of "snapshots" that form a *stream*. This also reflects the fact that sensor measurements are usually discrete in time. We use the terms *stream* and *evolution* interchangeably.

In higher order theories an entity stream could be specified in a number of ways. For example, `Stream` could be a sort `Stream = (L * T -> E)`, or a sort `Stream = (L * T * E -> Boolean)`, where `L` is the sort representing geographic location, `T` is the sort representing time and the asterisk represents the cartesian product. In the former case each element $s \in Stream$ assigns only one entity `E` to a point in space-time. In the latter case, a point in space-time can have multiple values. The same effect could be achieved by defining the sort `Stream = (L * T -> 2`$^{\texttt{E}}$`)`.

While higher order specifications are easier to understand, most theorem provers have considerable difficulty dealing with them, if they can be handled at all. It is also relatively difficult to represent them in databases. Accordingly, we restrict our specifications to first order. In the case of a `Stream`, it is specified using the operation `observe: Stream * L * T * E -> Boolean` (i.e., a relation with four fields). When `observe(s,l,t,e)` is true, it means that for the stream `s`, the entity `e` has been observed at time `t` in location `l`. Note that entities can overlap: more than one entity can be observed at the same point in space-time.

When specifying axioms in Slang, one writes `fa` for the universal quantifier (i.e., "for all") and `ex` for the existential quantifier (i.e., "there exists"). Quantification is over one or more variables, each of which varies over a specific sort. Thus `fa(x:T)` means "for all `x` in `T`".

The following is the specification of a `Stream` in Slang:

```
%% Stream
%% Template for a data or object stream.

spec STREAM is
  import LOCATION   % geographic location
```

```
  import TIME       % temporal location
  sort Stream       % streams
  sort E            % elements that vary over space and time.

  op observe: Stream * L * T * E -> Boolean

  %% A stream is determined by its observations,
  axiom measure_extensionality is
    fa(s1:Stream,s2:Stream)
       (fa(l:L,t:T,e:E)
          observe(s1,l,t,e) = observe(s2,l,t,e)) => s1 = s2
end-spec
```

`Extraction` is the process whereby raw measurements are converted to objects, features, relations and so on. Assuming that extraction is being performed in real-time, then extraction must satisfy a *causality* condition: one cannot use future measurements for current extraction. Of course, if the extraction is being done offline, then this requirement can be relaxed.

To specify object extraction we make use of the colimit operation. This operation allows one to build specifications in a modular fashion. A colimit is a combination of a set of specs that may have features (i.e., sorts and operations) in common. Importing one spec into another is the simplest example of a colimit. More complex forms of colimit allow one to perform a form of "template instantiation." In the specification for object extraction, the general notion of a Stream is specialized to that of an object stream using a colimit. The resulting spec is then imported into the spec for object recognition as follows:

```
%% Object streams are the outputs
%% of the object recognition process.

def OBJECTSTREAM : Spec = Specware.translate STREAM by
  [ "Stream" |-> "OStream", "E" |-> "O" ]

%% Object recognition extracts objects from measurements.

spec OBJECTRECOGNITION is
  import MEASUREMENTSTREAM
  import OBJECTSTREAM

  %% Object recognition using a sensor measurement stream.
  op recognize: MStream -> OStream

  %% Causality for object recognition.
```

```
  axiom object_recognition_causality is
    fa(m1:MStream,m2:MStream,t:T)
      (fa(p:T,l:L,v:V)
        le(p,t) => measure(m1,l,p,v) = measure(m2,l,p,v))
      =>
      (fa(p:T,l:L,o:O)
        le(p,t) => object(recognize(m1),l,p,o)
                   = object(recognize(m2),l,p,o))
end-spec
```

Level 2 data fusion is a process whereby *relations* are deduced from the objects determined by level 1 processing or from other relations. Relations vary in time (i.e., evolve over time), but they do not have spacial extent. A *situation* is a collection of *situation objects* that includes object and relation streams as well as other situations.

The following is a specification of relation streams in Slang:

```
%% Relation streams are the outputs of
%% the relation recognition process.
%% Relation streams are temporal but not geographic,

spec RELATIONSTREAM is
  import OBJECTSTREAM
  sort RSymbol          % Relation symbols
  sort RStream          % Relation stream for one relation

  %% The meaning of relates(r,t,o1,o2) is that in the relation
  %% stream r, the objects o1 and o2 are related at time t.
  op relates: RStream * T * O * O -> Boolean

  %% Each relation stream has a unique relation symbol.
  op relationSymbol: RStream -> RSymbol

  %% A relation stream is determined
  %% by its symbol and its values.
  axiom relation_stream_extensionality is
  fa(r1:RStream,r2:RStream)
    (fa(t:T,o1:O,o2:O)
       relates(r1,t,o1,o2) = relates(r2,t,o1,o2))
    & relationSymbol(r1) = relationSymbol(r2) => r1 = r2
end-spec
```

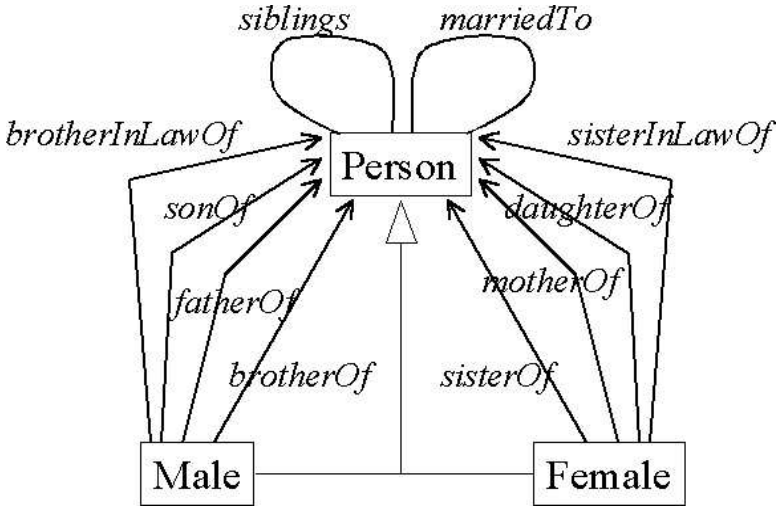


Figure 2: The Family Relationship Ontology in UML

# 5 SITUATION EXAMPLE

We now show an example of a simple situation involving family relationships. For simplicity, we have suppressed the time and space evolution of the objects and relationships. The formalization makes use of all three languages, and the translations between them were automated to the extent that this was possible. The specification has two parts: the ontology (schema) and the annotation (specific situation). We first show the situation in UML, then in DAML and finally in Slang.

## 5.1 Family Situation in UML

The Family Relationships is shown in Figure 2. The class Person is the common generalization of the Male and Female classes. The various associations involving Person are inherited by its subclasses. These associations are not independent of one another, but these dependencies cannot be expressed graphically in UML (although they can be specified using OCL). A particular family situation is shown in Figure 3.

## 5.2 Family Situation in DAML

The family ontology was translated from UML to DAML using DUET [DUE02], and part of the resulting DAML ontology is as follows:

```
<a:Ontology rdf:about="Family"/>
<a:Restriction rdf:about="Family#DUET0"/>
    <a:toClass rdf:resource="Family#Female"/>
    <a:onProperty rdf:resource="Family#daughter"/>
</a:Restriction>
<a:Restriction rdf:about="Family#DUET1">
    <a:toClass rdf:resource="Family#Person"/>
    <a:onProperty rdf:resource="Family#daughterOf"/>
</a:Restriction>
<a:Restriction rdf:about="Family#DUET2">
    <a:toClass rdf:resource="Family#Female"/>
    <a:onProperty rdf:resource="Family#mother"/>
</a:Restriction>
```

Relationships are called `Properties` in DAML, and the usual mechanism in DAML for specifying domain and range constraints is to impose a `Restriction` on the relationship.

The particular situation was also translated to DAML as follows:

```
<f:Male rdf:ID="John"/>
<f:Male rdf:ID="Paul"/>
<f:Female rdf:ID="Lidia">
    <f:marriedTo rdf:resource="#John"/>
</f:Female>
<f:Male rdf:ID="Peter">
    <f:fatherOf rdf:resource="#Paul"/>
    <f:fatherOf rdf:resource="#Lidia"/>
</f:Male>
```

Note that DAML annotations look very much like ordinary XML documents. In most cases, it is straightforward to translate from an XML DTD to a DAML ontology. One cannot completely automate this process because the DTD does not have all the necessary information. In particular a DTD will not distinguish a class from an association. Nevertheless, a tool has been developed that assist a developer in the task of converting XML DTDs and XSD schemas to DAML [Nei02].

## 5.3 Family Situation in Slang

The family ontology is written in Slang as follows:

```
sort Person
op sonOf: Person * Person -> Boolean
op daughterOf: Person * Person -> Boolean
op brotherOf: Person * Person -> Boolean
```

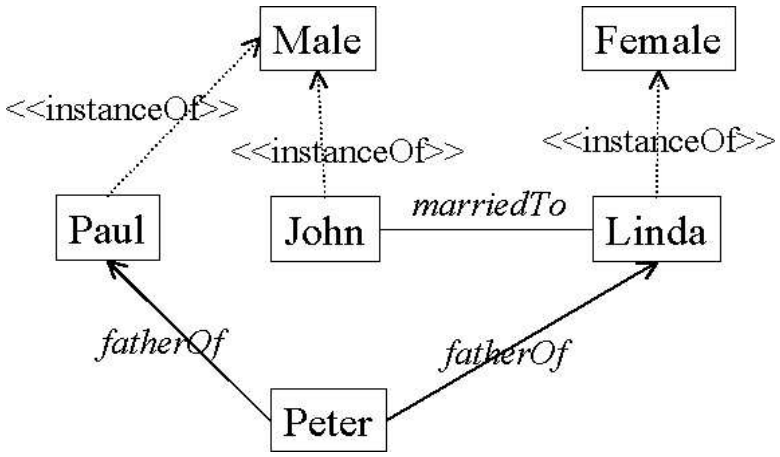


Figure 3: The Family Relationship Situation in UML

```
op sisterOf: Person * Person -> Boolean
op siblings: Person * Person -> Boolean
op brotherInLawOf: Person * Person -> Boolean
op fatherOf: Person * Person -> Boolean
op motherOf: Person * Person -> Boolean
op marriedTo: Person * Person -> Boolean
op male: Person -> Boolean
op female: Person -> Boolean

axiom son_is_male_childOf_his_father is
    fa(X:Person,Y:Person) male(Y) & fatherOf(X,Y) => sonOf(Y,X)
axiom sister_female_same_father is
    fa(X:Person,Y:Person,Z:Person) female(X) & fatherOf(Z,X) &
        fatherOf(Z,Y) => sisterOf(X,Y)
axiom sister_female_same_mother is
    fa(X:Person,Y:Person,Z:Person) female(X) & motherOf(Z,X) &
        motherOf(Z,Y) => sisterOf(X,Y)
axiom brother_male_same_father is
    fa(X:Person,Y:Person,Z:Person) male(X) & motherOf(Z,X) &
        motherOf(Z,Y) => brotherOf(X,Y)
axiom siblings_are_brother_sister is
    fa(X:Person,Y:Person)
      brotherOf(X,Y) or sisterOf(X,Y) => siblings(X,Y)
axiom marriage_is_symmetric is
    fa(X:Person,Y:Person) marriedTo(X,Y) => marriedTo(Y,X)
axiom brotherInLawOf_male_sibling_of_siblings_spouse is
```

```
fa(X:Person,Y:Person,Z:Person) male(X) & siblings(X,Y) &
    marriedTo(Y,Z) => brotherInLawOf(X,Z)
```

The family situation also uses operations and axioms. The operations in this case have only a domain and no range. Such an operation is just a constant. The situation is as follows:

```
op John: Person
    axiom John_is_male is male(John)
op Paul: Person
    axiom Paul_is_male is male(Paul)
op Lidia: Person
    axiom Lidia_is_female is female(Lidia)
op Peter: Person
    axiom Peter_is_male is male(Peter)

axiom Peter_is_fatherOf_Paul is fatherOf(Peter,Paul)
axiom Peter_is_fatherOf_Lidia is fatherOf(Peter,Lidia)
axiom Lidia_is_marriedTo_John is marriedTo(Lidia,John)
```

Having represented the family ontology and situation in Slang, one can then use the SNARK theorem prover to prove theorems.

## ACKNOWLEDGEMENTS

This research was partially supported by AFRL/IF under contract F30602-02-C-0039.

## REFERENCES

[Bar81] J. Barwise. Scenes and other situations. *J. Philosophy*, 77:369–397, 1981.

[Bar89] J. Barwise. *The Situation In Logic*, volume 17. CSLI/SRI International, Menlo Park, CA, 1989.

[BJR00] G. Booch, I. Jacobson, and J. Rumbaugh. *OMG Unified Modeling Language Specification*, March 2000. Available at `www.omg.org/technology/documents/formal/unified_modeling_language.htm`.

[BKK+02] K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, J. Letkowski, and P. Emery. Extending the Unified Modeling Language for ontology development. *Software and System Modeling*, 1(2):142–156, 2002.

[Bun77] M. Bunge. *Treatise on Basic Philosophy. III: Ontology: The Furniture of the World*. Reidel, Dordrecht, Netherlands, 1977.

[BZSF96] E. Bass, J. Zenyuh, R. Small, and S. Fortin. A context-based approach to training situation awareness. In *Proc. Third Annual Symposium on Human Interaction with Complex Systems*, pages 89–95, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[DAM01] DAML. DARPA Agent Markup Language website, 2001. `www.daml.org`.

[DUE02] DUET. DAML UML enhanced tool (DUET), 2002. `grcinet.grci.com/maria/www/CodipSite/Tools/Tools.html`.

[EG00] M. Endsley and D. Garland. *Situation Awareness, Analysis and Measurement*. Lawrence Erlbaum, Mahwah, NJ, 2000.

[KBF00] M.M. Kokar, M. Bedworth, and K. Frankel. A reference model for data fusion systems. In *Sensor Fusion: Architectures, Algorithms, and Applications IV*, pages 191–202, Orlando, FL, 2000.

[KK01] M.M. Kokar and Z. Korona. A formal approach to the design of feature-based multi-sensor recognition systems. *Int. J. Information Fusion*, 2(2):77–89, 2001.

[KTW99] M.M. Kokar, J. Tomasik, and J. Weyman. A formal approach to information fusion. In *Proc, of the Second Int. Conf. Information Fusion*, volume 1, pages 133–140, 1999.

[KTW01] M.M. Kokar, J. Tomasik, and J. Weyman. Data vs. decision fusion in the category theory framework. In *Proc. of FUSION 2001 - 4th International Conference on Information Fusion*, volume 1, pages TuA3–15–TuA3–20, 2001.

[KW02a] M.M. Kokar and J. Wang. An example of using ontologies and symbolic information in automatic target recognition. In *Sensor Fusion: Architectures, Algorithms, and Applications VI*, pages 40–50, Orlando, FL, 2002.

[KW02b] M.M. Kokar and J. Wang. Using ontologies for recognition: An example. In *Proc. 5th Int. Conf. Information Fusion*, pages 1324–1343, 2002.

[Nei02] M. Neighbors. XML to DAML translator, 2002. `www.davinciNetBook.com:8080/daml/xmltodaml/presentation/sld001.htm`.

[SBW99] A. Steinberg, C. Bowman, and F. White. Revisions to the JDL data fusion model. In *SPIE Conf. Sensor Fusion: Architectures, Algorithms and Applications III*, volume 3719, pages 430–441, April 1999.

[SNA02] SNARK. SRI's new automated reasoning kit, 2002. `www.ai.sri.com/~stickel/snark.html`.

[SWC02] M.E. Stickel, R.J. Waldinger, and V.K. Chaudhri. A Guide to SNARK, 2002. `www.ai.sri.com/snark/tutorial/tutorial.html`.

[von97] L. vonMises. *Human Action: A Treatise on Economics*. Fox & Wilkes, January 1997. Originally published in 1949.

[W+98] R. Waldinger et al. *Specware™ Language Manual: Specware™ 2.0.3*, March 1998.