Representation for Dynamic Situation Modeling

Bruce D'Ambrosio, Masami Takikawa, and Daniel Upper
Information Extraction and Transport, Inc.
1600 SW Western Blvd., Suite 300
Corvallis, OR 97333
{dambrosi,takikawa,upper}@iet.com

Abstract

In dynamic situation modeling we do not know at design time all the entities in a situation, their types, or the relationships in which they participate. We present a status report on JPF, a Frame-based probabilistic representation and reasoning system with embedded support for hypotheses about the structure of a situation, including existence, type, and association (relational) hypotheses. We describe the external representation and its mapping to a Bayesian network. We close with a summary of our experience in using this system on two large DARPA projects.

1 Introduction

1.1 Task

Our focus task is dynamic situation modeling. Given an input stream of reports from multiple sensors of multiple types, the task is to infer the entities present, their states, relationships, and activities. For example, given an input stream of reports from network intrusion detection systems and intelligence sources, are we under cyber attack and if so by whom, with what intent? This task is essentially constructive. It is not possible to pre-specify, at a sufficient level of detail, a single Bayesian network that encompasses all possible situations one might encounter. For example, in the cyber attack domain, one may be simultaneously under multiple attacks form several sources, with varying objectives. An unknown number of cyber agents (e.g. viruses, trojans) may be present on one or more hosts in our system, organized into teams to pursue any of a number of possible objectives.

The task is related to many other "understanding" tasks. We briefly explore the relationship to three: diagnosis, image understanding, and plan recognition. It is different from each in significant ways. Diagnosis typically assumes known entities and structure. Here we know only the entity types that might be present, and how they might be related. For example, we model the domain fact that an exfiltration (removal of data) attack on a classified site is most likely to be carried out by a highly skilled attacker. We can use this fact whenever we suspect an attack on a classified site.

Image understanding, like situation assessment, focuses on hypothesizing entities and relationships among them. However, for image understanding the relationships are typically static geometric ones, whereas in situation assessment the relationships are primarily functional. Using the example above, the *agent* of an attack is in a functional relationship to the attack itself.

Finally, the above example may seem, and is, very similar to plan recognition. However, while plan recognition usually starts from known entities, we get only indirect evidence about the possible existence of entities in a situation. We can lack prior knowledge, not only about the attacks under way, but also about the agents who execute those attacks, both human and cyber, as well as the resources utilized, such as hosts and network infrastructure.

1.2 Research Program

We have developed JPF, a probabilistic frame-based representation language, for use in these dynamic situation modeling tasks. JPF provides three major facilities, the first of which is a language for constructing situation-independent models of a domain. These models contain information about types of entities (for example, "hosts with classified information are usually hard to attack"), but do not refer to specific entities at all.

The second major facility JPF provides is a set of structural uncertainty primitives. Taken together, a domain model and these structural uncertainty primitives constitute a language for constructing situation models in that domain. Finally, JPF produces the situation model in Bayes net form.

Our initial hypotheses were that: (1) the domain modeling language would have the expressivity needed to model interesting domains; (2) the situation modeling language, comprised of the domain model and the structural uncertainty primitives, would permit reasonably parsimonious construction of dynamic situation models; and (3), the resulting Bayes net would be computationally tractible for moderately large situations.

1.3 Representational Requirements

The task of situation assessment as we have described is constructive, and therefore our representation must be compositional. We use a two-level representation, in which entities and relationships in a situation are described as instances of generic types. Much of the knowledge about types, and significant elements of reasoning about situations, is taxonomic. This suggests that our generic types should exist in a type hierarchy. Finally, we usually know more than just the type of an entity - associated with an entity type is a set of attributes. The particular set of attributes, and expectations about values those attributes can take, are both important elements of domain knowledge.

Domain knowledge of this sort has traditionally been captured in object-oriented and, more generally, frame-based representational systems, and so we have adopted the general syntax of frame-based systems. So, for example, we have frames for various types of cyber attack, different types of attacking agent, and so on. However, while useful, standard frame-based representations are not sufficient. Situation assessment is a task in hypothetical reasoning. In particular, our representation must be able to support four types of hypotheses about the structure of constructed situations, existence, type, association, and identity hypotheses.

Existence hypotheses are necessary because we often have indicative, but not definitive, evidence about the presence of an entity or activity. For example, a software upgrade event *may* introduce a malicious agent along with the upgraded software. An attempt to contact an invalid IP number on our network *may* be part of a network mapping attack. However, until further confirmatory evidence causes acceptance of the existence of the agent or attack, it remains hypothetical.

Type hypotheses are necessary because initial information, in addition to being uncertain, is partial. A status report may imply no more than that we may be under attack. Yet, as mentioned earlier, much of the information available to confirm or reject the attack hypothesis is under specific attack types. Because the taxonomy is large, it is computationally intractable to consider all possible subtypes of an abstract type like “attack” simultaneously. Rather, there is considerable expertise in how one explores the space of possible types for an entity or activity.

Association hypotheses are necessary because entities and activities do not exist in isolation. When we hear of an attack, we surmise possible attackers, attack-targets, objectives, and so on. Note these are not typical random variables because their domains are not, in general, pre-enumerated. Rather, a type is specified in advance and association hypotheses draw specific instances of that type to add to a domain.

Identity hypotheses are necessary because it is possible for multiple existence hypotheses to refer to a single thing-in-the-world. For example, we may only later realize that two separate attack hypotheses, each created from separate evidence, in fact both denote the same actual attack. We have not yet implemented a mechanism for identity hypotheses and will not discuss them further.

In summary, our representational requirements include the ability to model entity and relationship types, their taxonomic relationships and their attributes. In addition, we require the ability to form structural hypotheses about the existence, type, and association of instances of these entity types in a particular situation.

2 Domain Description

In this section, we describe the modeling language. The modeling language is based loosely on frames, a popular knowledge representation aproach, and is augmented with various ways to construct structural hypotheses. The semantics of the modeling language is understood by the Bayes net it creates for the instantiated frames.

2.1 Frames

The fundamental modeling unit is the frame. A frame defines general properties hold among a class of objects, called *frame instances*. Frames contain *slots*, roughly, attributes. Each *slot* can have a number of *facets* defined on it.

Some of these facet names are reserved words, and their values define the probability model over instances of frame definitions.

Frames exist in a lattice (i.e., multiple inheritance), and inherit all slots (and facets defined on them) defined in parent frames.

2.1.1 Frame Example

Figure 1 shows a part of the taxonomy of attacks in cyber battlefields we have developed for DARPA Information Assurance Cyber Command and Control (IA/CC2) program. Simplified version of Attack and CyberAttack frame definitions are shown in Figure 2.

The Attack frame defines two slots called target and targetStatus. These slots are also available in CyberAttack because it is a subframe of Attack as specified by an “isa”

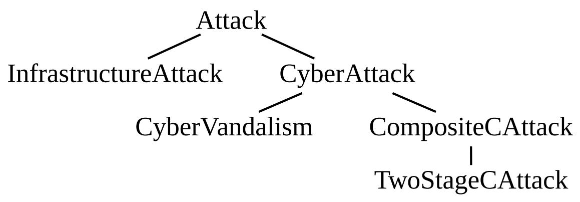

Figure 1: Attack taxonomy for IA/CC2.

```
frame Attack isa Activity
  slot target
    facet domain = Agent
    facet distribution = UniformDist
  slot targetStatus
    facet domain = [Inopera-
tive, Operative]
    facet parents = [target.status]
    facet distribution =
      function x,y {
        if x==y then 1 else 0 end
      }
end;
frame CyberAttack isa Attack
  slot confidentiality
    facet domain = [True, False]
    facet distribution = [0.5, 0.5]
  slot integrity
    facet domain = [True, False]
    facet parents = [confidentiality]
    facet distribution =
      function conf {
        if conf==True then [0.6, 0.4]
                      else [0.5, 0.5] end
      }
end;
```

Figure 2: Attack and CyberAttack definitions.

clause. CyberAttack adds two slots, called confidentiality and integrity, which represent the objectives of the cyber attack, that is, what the attack tries to compromise.

The "domain" facet defines the domain of a slot. It can be a finite set as in the targetStatus, confidentiality, and integrity slots, or can be a frame as in the target slot, in which case, the slot can contain any instance of that frame. We call a slot with a frame domain a *reference slot*.

The "distribution" facet defines a probability distribution over the slot domain. The distribution can be specified in various ways. The distribution for the confidentiality slot is specified by a vector of probability numbers, and the distributions for the targetStatus and integrity slots are written as functions.

If there is a "parents" facet, then the distribution facet defines a conditional probability table (CPT) conditioned by the specified parent slots. For example, the integrity slot in the CyberAttack frame is conditioned by the confidentiality slot in the same frame. Its distribution says that if the cyberAttack attacks the confidentiality of the target, then it is more likely that it also attacks the integrity of the target.

2.1.2 Mapping To Bayes Nets

We perform inference by mapping a situation represented by a set of frame instances into a Bayes net.

When a frame is instantiated, one Bayes node is created for each uncertain slot defined in the frame. Figure 3 shows the Bayes net fragment for an instance of CyberAttack, *A*.

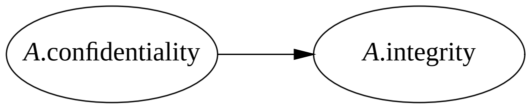

Figure 3: Bayes net fragment for a CyberAttack instance.

2.2 Structural Hypotheses

The fundamental structural hypothesis is that something exists in the world. We term this an *existence* hypothesis.

Given that an entity exists, the next question we need to answer to define its representation is its *type*, that is, its location in the frame type lattice. Type hypotheses are hypotheses about the type of an entity. We currently support only subtype hypotheses, that is, hypotheses that an entity currently modeled at one level of the type lattice may be modeled as an immediate subtype.[1]

Finally, much of interest in a domain concerns the relationships among entities. *Association* hypotheses are hypotheses about the value of a slot whose domain is drawn from a class of frame instances. Association hypotheses come in two flavors: one-to-one (e.g., organization behind a particular attack) or one-to-many (e.g., the set of missions this web server is supporting). A slot with one-to-one association is called a *reference uncertainty slot*, and a slot with one-to-many association is called a *set-valued slot*.

Path expressions are used to define parents by referencing through associations. An example path expression appears in the parent facet in the targetStatus slot in Figure 2, which refers to the status slots of the Agent instance stored in the reference uncertainty slot called "target".

[1]We assume that each things-in-the-world is correctly modeled as a single leaf type, we just don't know which one.

2.2.1 Implementing Existence

The implementation of basic existence is pretty straightforward. Each hypothetical frame instance has an "exists" node with domain {"Context.In", "Context.Out"}. All nodes are conditioned by the exists node, and Context.Out is added to the domain of all nodes so that if exists node is Context.Out all conditioned node also becomes Context.Out. Context.Out is normally invisible to the user.

For example, if the existence of a CyberAttack instance, *A*, is hypothetical, an exists node is created, and it conditions all nodes found in the frame. (See Figure 4.)

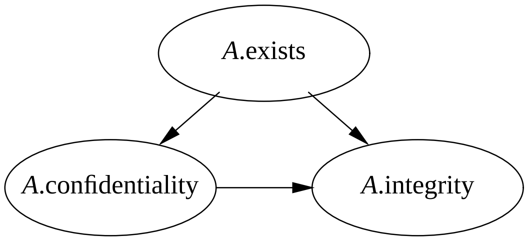

Figure 4: Bayes net fragment for a hypothetical CyberAttack instance.

2.2.2 Implementing Subtype

Subtype hypotheses are modeled by adding a *subtype* node to the Bayes net for the parent frame instance, which conditions the exists nodes in the child instance. The conditional probability table (CPT) of the child exists node is defined to ensure that the child instance exists if and only if the parent instance exists and the subtype node refers to the particular child.

There is an important issue introduced by subtyping: What do priors mean if they can be redefined at any level? We take the prior specified in any frame other than a leaf frame to be the *default prior* for child frames. The actual prior for a non-leaf frame-instance is the expectation over the priors of its instantiated (through subtype hypotheses) children.

The CPTs of nodes for slots in the parent encode the expectation over the corresponding nodes in the children. For example, let frame *F* have two children, C_1 and C_2. Then the system generates the following CPT for the slot *S* of *P*:

$$P(F.S|F.\text{subtype}, C_1.S, C_2.S) = \begin{cases} 1 \text{ if } F.\text{subtype} = C_i \text{ and } F.S = C_i.S \text{ for } 1 \le i \le 2 \\ 0 \text{ otherwise.} \end{cases}$$

This CPT can be readily generalized for cases with *n* children.

Figure 5 shows the Bayes net fragment for this example subtype relation, including exists nodes.

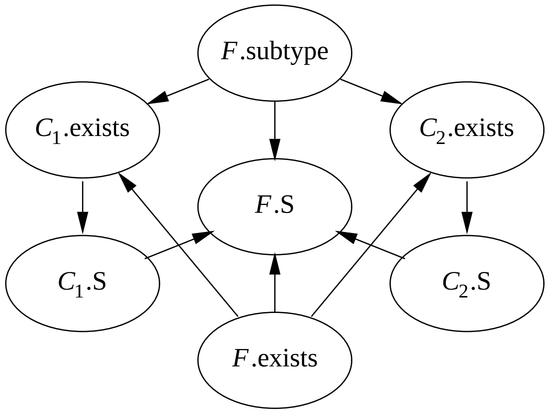

Figure 5: Bayes net fragment for the subtype example.

2.2.3 Implementing One-to-One Association

We use a simple "switch" or "multiplexor" representation for the one-to-one association. For example, consider the TwoStageCAttack frame defined in Figure 6.

```
frame TwoStageCAttack isa CyberAttack
  slot 1st
    facet domain = CyberAttack
    facet distribution = UniformDist
  slot 2nd
    facet domain = CyberAttack
    facet distribution = UniformDist
  slot integrity
    facet domain = [True, False]
    facet parents = [1st.integrity,
                     2nd.integrity]
    facet distribu-
tion = T.S.IntegrityDist
end;
```

Figure 6: A part of definition of TwoStageCAttack frame.

In this frame, there are two reference uncertainty slots called "1st" and "2nd", both of which refer to a CyberAttack. The third slot, "integrity" is conditioned by the integrity of two CyberAttacks. Let *f*=T.S.IntegrityDist be the function that specifies the CPT, that is, the function that returns a probability number given the integrity value of the first attack, the second attack, and the two-stage attack.

Suppose that *T* is an instance of TwoStageCAttack, and its first stage can be one of CyberAttack instances $\{A_1, \ldots, A_m\}$, and its second stage can be one of $\{B_1, \ldots, B_n\}$. Then, the system generates the following CPT for the integrity (*I*) of *T*:

$$
\begin{aligned}
P(T.I|T.1\text{st}, T.2\text{nd}, A_1.I, \ldots, A_m.I, B_1.I, \ldots, B_n.I) = \\
f(A_i.I, B_j.I, T.I) \text{ if } T.1\text{st} = A_i \text{ and } T.2\text{nd} = B_j \\
\text{for } 1 \le i \le m \text{ and } 1 \le j \le n
\end{aligned}
$$

The above representation requires the CPT of size exponential in the number of references. In order to avoid this exponential explosion, we factor the above CPT multiplicatively using Local Expression Language [D'Ambrosio, 1995].

Following [Takikawa and D'Ambrosio, 1999], we use the following notation for generalized distributions:

$$
\begin{aligned}
G(\ & X_1, \ldots, X_m | Y_1, \ldots, Y_n, \\
& \langle f(X_1, \ldots, X_m, Y_1, \ldots, Y_n) \rangle),
\end{aligned}
$$

where X_i is a conditioned variable, Y_j is a conditioning variable, and f is a density function specifying actual numerical probabilities.

Using Local Expression Language, the above CPT can be factored as follows:

$$
\begin{aligned}
& P(T.I|T.1\text{st}, T.2\text{nd}, A_1.I, \ldots, A_m.I, B_1.I, \ldots, B_n.I) = \\
& \prod_{i=1}^{m} \prod_{j=1}^{n} G(T.I| \ T.1\text{st}, T.2\text{nd}, A_i.I, B_j.I \\
& \qquad \langle f_{ij}(T.I, T.1\text{st}, T.2\text{nd}, A_i.I, B_j.I) \rangle)
\end{aligned}
$$

where

$$
\begin{aligned}
& f_{ij}(T.I, T.1\text{st}, T.2\text{nd}, A_i.I, B_j.I) = \\
& \begin{cases} f(A_i.I, B_j.I, T_I) \text{ if } T.1\text{st} = A_i \text{ and } T.2\text{nd} = B_j \\ 1 \text{ otherwise.} \end{cases}
\end{aligned}
$$

In this representation, each generalized distribution contains only one reference for each parent, so its size is fixed no matter how many references there are in the reference uncertainty slot, avoiding the exponential explosion.

Note that the same optimization is applicable to the subtype hypotheses. Also note that any inference algorithm can be easily extended to handle this multiplicative factorization of multiplexors.

2.3 Roles

It is often important to place expectations on slot-fillers. For example, the attacker in a smurf attack is usually moderately sophisticated technologically. We represent most of these as *constraints*, that is, as soft observations on derived values. For example, the above might be represented as shown in Figure 7.

In this frame, the attacker slot is a reference uncertainty slot, which refers to a possible attacker Agent. The attackerLevel slot represents the attacker's technology level. It is inferred from the techLevel slot of the attacker through the distribution function which encodes the identity matrix.

```
frame SmurfAttack isa CyberAttack
  slot attacker
    facet domain = Agent
    facet distribution = UniformDist
  slot attackerLevel
    facet domain = [Super, Good, Bad]
    facet par-
ents = [attacker.techLevel]
    facet distribution =
      function x,y {
        if x==y then 1 else 0 end
      }
    facet observation = [0.2, 0.6, 0.2]
end;
```

Figure 7: The definition of SmurfAttack frame.

The most important part is the observation facet of the attackerLevel slot, which puts a soft observation, represented as a likelihood vector, to the attackerLevel node of SmurfAttack instances. In this case, the soft observation favors "Good" technology level.

The Bayes net for an example SmurfAttack instance (S) with two attacker association hypotheses (A_1 and A_2) is shown in Figure 8.

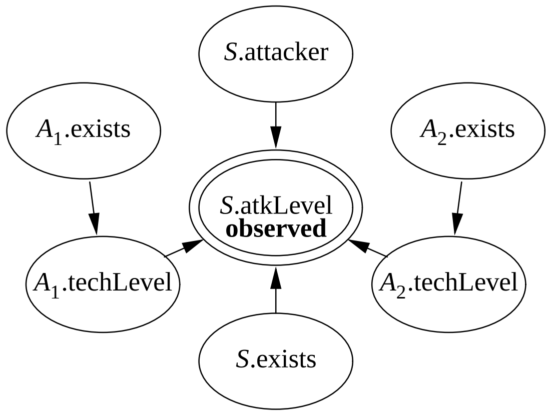

Figure 8: Bayes net fragment for the SmurfAttack example.

This representation provides compositionality, is mediated by the existences of the participating objects and their participation in associations, and avoids any possibliity of introducing directed loops in the Bayes net.

2.4 One-to-Many

We use a "SetMember" frame to build a set. SetMember frames represent a linked list of members whose membership is conditioned by a membership slot, and are customized for each use. For example, Figure 9 shows the

AttackerSetMember frame that is customized to represent a set of attackers.

```
frame AttackerSetMember
  slot member
    facet domain = Attacker
  slot next
    facet domain = AttackerSetMember
  slot membership
    facet domain = [True, False]
    facet distribution = [.5, .5]

  slot levelConstraint
    facet domain = [Valid, Invalid]
    facet parents =
      [membership, member.techLevel]
    facet distribution =
      function membership, level {
        if member-
ship==True && level!=Bad
        then Valid
        else InValid end
      }
    facet observation = [.9, .1]

  slot superCount
    facet parents =
      [membership, member.techLevel,
       next.superCount]
    facet distribution =
      function member-
ship, level, nextCount {
        if member-
ship==True && level==Super
        then nextCount+1
        else nextCount end
      }
end;
frame EmptyAttackerSetMember
        isa AttackerSetMember
  slot superCount = 0
end;
```

Figure 9: Frames for attacker sets.

The first three slots are essential: The “member” slot points to an actual member; the “next” slot points to the remaining set; and the “membership” slot represents a hypothesis that this member belongs to this set.

This representation permits expectations to be placed both on the set itself (e.g., cardinality constraints) and on set members. It is easy to specify, provides a place to gather role restrictions for a ont-to-many slot, and did not require any extension to the underlying implementation. It does, however, suffer the major disadvantage of fixing the order in which the members of the set are represented in the Bayes net.

The “levelConstraint” slot is an example of a constraint placed on all members. It states that attacker’s technology level should not be bad. This constraint is conditioned by the membership slot so it is effective only if this attacker is indeed a member of this set. Note that the constraint is soft, that is, a small amount (10%) of leakage is allowed.

A constraint on the set itself, such as “The number of super attackers should be at least one,” can be placed using a slot that summarizes the condition (the number of super attackers) and putting a constraint on that condition. The “superCount” slot in AttackerSetMember is an example of a summary slot that counts the number of super attackers.

3 Experience To Date

We have used JPF primarily as part of an automated system for constructing situation models, which is less mature than JPF and will be described here only very briefly to provide background. Our dynamic situation modeling engine is, loosely, a blackboard system which uses the current set of JPF frame instances as its blackboard. Changes to the frame world – the current set of frame instances – trigger domain-specific modules responsible for performing model construction actions, typically the creation of structural hypotheses. For example, a module that watches for reports of pings to invalid addresses might hypothesize: (1) that an existing attack might well account for this report, and so create an association hypothesis between the attack and the report; or (2) that a new attack is needed to explain the report, and so create an existence hypothesis for the new attack and an association hypothesis linking the attack and report; or (3) both possibilities. In general the modules implement coarse local decision policies about structural hypothesis creation. The set of construction suggestions generated by all triggered modules is then globally filtered, taking into account both competing hypotheses and the current model complexity.

We are using this system on two DARPA projects involving dynamic situation modeling. A characterization of our work to date on the two projects is shown in Table 1.

In this table, *Observations* includes both hard and soft observations, *Computation* information is for computation of all marginals, and *Largest Tbl* is specified in the number of entries. As can be seen from this information, the networks stay sparse and computationally tractable for moderately large situations. Exploitation of local structure as described earlier is essential for this result. The larger network is intractable when the type and association expressions are flattened into simple conditional distributions (the largest table: $2.3 * 10^{15}$ entries).

JspiScript JPF includes its own scripting language jspiS-

Measure	**DDB**	**CC2**
Frames	138	45
slots	359	168
facets	137	168
Typical Run		
Frame Instances	192	111
Existence Hypotheses	85	29
Type Hypotheses	80	0
Association Hypotheses	170	219
Nodes	1016	840
Local Expressions	2258	1438
Observations	228	217
Computation		
time in seconds	44.9	22.7
# mults	$22 * 10^6$	$2.8 * 10^6$
Largest Tbl	373,248	972

Table 1: Example model characteristics.

cript, a complete programming language with additional features for defining and manipulating distributions, local expressions, Bayes nets, frames, and frame instances. Frame definitions (including all examples in this paper) are written in jspiScript, and its general-purpose features can be used as a macro facility. JspiScript is used to write test suites as well as some domain-specific modules for the dynamic situation modeling engine. We use it heavily in interactive mode for testing and debugging. Finally, a frame definition may contain, in place of a CPT, a jspiScript function defining the CPT as has been seen in the examples. We have found the scripting language to be an essential component of large-scale knowledge engineering.

Issues One fact which the current version of JPF does not represent well is coreference. Suppose we have agents A_1 and A_2, and one of them (but we do not know which) is the source of a CyberAttack C_1. As described above, we model this by adding A_1 and A_2 to the (previously empty) domain of C_1.source. We then learn of a second CyberAttack, C_2, and we determine that C_2 has the same source as C_1. This fact is lost if we simply add A and B to the domain of C_2.source.

We have not yet determined how we will extend JPF to handle this kind of information. One possibility is to use a reference uncertainty slot as a placeholder for the frame instance to which it refers, so that we could put C_1.source in the domain of C_2.source.

Another possibility we are exploring is to define a new kind of frame instance, the "reference instance". A reference instance would have the same slots as a normal instance of the same frame, plus an extra reference uncertainty slot. The domain of this extra slot would contain referent instances, and the CPTs for the other slots would encode expectations over the corresponding slots in these referent instances. (Reference instances are connected to their referent instances in much the same way that parent instances are connected to their child instances.) In our example, we would create a reference agent instance *Href*, with referent instances H_1 and H_2, and we would put *Href* alone in the domains of both C_1.source and C_2.source.

A second issue we have begun to explore is *identity*. As mentioned earlier, it is not unusual to create two existence hypotheses for what later is discovered (or hypothesized) to be the same object. It is a simple matter of programming to make the categorical assertion that this is the case. It is more problematic to construct a hypothesis that this is so, and we are currently exploring alternate Bayes net mappings for identity hypotheses.

4 Related Work

Mahoney and Laskey [1998] have written about constructing situation models. Our work is consistent with theirs, but emphasizes the engineering aspects of automatically constructing and evaluating large-scale situation representations. Pfeffer *et al.* [1999] report on a frame-based system very similar in syntax to ours. In fact, much of our syntax is unashamedly stolen from SPOOK. JPF differs from SPOOK in its emphasis on providing facilities for hypothesizing about and managing the structure of a situation representation.

Earlier work on knowledge-based model construction includes work by Breese [1987], Goldman and Charniak [1990], and Wellman [1990]. JPF is most similar in spirit to Goldman and Charniak's work. As with SPOOK, a key difference is our focus on structural hypotheses. Breese focused on the control of situation construction, and used backward-chaining rule-based methods to drive construction. This paper has not focused on structural hypothesis management, but, as alluded to earlier, we use blackboard-style control with explicit decision-theoretic modeling of the cost/benefit of each hypothesis management action. Finally, Wellman, like us, focused on the need for hierarchical refinement during the problem solving process. However, his work was performed in a qualitative probabilistic framework and assumed a known structure, whereas we use traditional discrete domains and must hypothesize structure.

5 Conclusions

We have described JPF, a probabilistic frame-based representation language for dynamic situation modeling. Our initial hypotheses were that such a system: (1) would have the expressivity needed to model interesting domains; (2) would provide the primitives needed to effectively control the potential combinatorial explosion inherent in dynamic situation modeling; and (3) would be computationally tractable for moderately large situations.

Our experience to date provides affirmative answers to the first two questions. First, while expressivity is partially a subjective measure, we have been able to successfully perform initial modeling in two large-scale domains. Second, JPF has served as the representation and reasoning layer on which we have constructed an engine for dynamic situation modeling currently in use on those same two projects (separate paper currently under development).

Our experience with respect to the third hypothesis is somewhat more qualified. While we have reached our initial goal of modeling situations involving thousands of nodes, it is clear that new approaches will be required to reach our goal of hundreds of thousands of nodes. We believe a more continual, incremental approach to inference [Horvitz99], incorporating notions of locality, granularity, and precision, will be needed to build a truly scalable system.

Acknowledgements

The ideas in this paper have benefited from discussion with Suzanne Mahoney, Ed Wright, and Rui Yang of IET. Funding for domain modeling using JPF is provided in part under the DARPA Cyber Command and Control program contract # F30602-99-C-0148.

References

[Breese, 1987] J. Breese. *Knowledge Representation and Inference in Intelligent Decision Systems*. PhD thesis, Department of Engineering-Economic Systems, Stanford University, 1987.

[D'Ambrosio, 1995] B. D'Ambrosio. Local expression languages for probabilistic dependence. *International Journal of Approximate Reasoning*, 13:61–81, 1995.

[Goldman and Charniak, 1990] R. Goldman and E. Charniak. Dynamic construction of belief networks. In *Proceedings of the Sixth Workshop on Uncertainty in AI*, pages 90–97, 1990.

[Mahoney and Laskey, 1998] S. Mahoney and K. Laskey. Constructring situation specific networks. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 370–378, 1998.

[Pfeffer *et al.*, 1999] A. Pfeffer, D. Koller, B. Milch, and K. T. Takusagawa. SPOOK: A system for probabilistic object-oriented knowledge representation. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 541–549, 1999.

[Takikawa and D'Ambrosio, 1999] M. Takikawa and B. D'Ambrosio. Multiplicative factorization of noisy-max. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 622–630, 1999.

[Wellman, 1990] M. Wellman. *Formulation of Tradeoffs in Planning Under Uncertainty*. Pitman and Morgan Kaufmann, 1990.