Structured Representation of Complex Stochastic Systems

Nir Friedman*
Computer Science Division
387 Soda Hall
U.C. Berkeley
Berkeley, CA 94720
nir@cs.berkeley.edu

Daphne Koller
Computer Science Department
Stanford University
Gates Building, 1A
Stanford, CA 94305-9010
koller@cs.stanford.edu

Avi Pfeffer
Computer Science Department
Stanford University
Gates Building, 1A
Stanford, CA 94305-9010
avi@cs.stanford.edu

Abstract

This paper considers the problem of representing complex systems that evolve stochastically over time. *Dynamic Bayesian networks* provide a compact representation for stochastic processes. Unfortunately, they are often unwieldy since they cannot explicitly model the complex organizational structure of many real life systems: the fact that processes are typically composed of several interacting subprocesses, each of which can, in turn, be further decomposed. We propose a hierarchically structured representation language which extends both dynamic Bayesian networks and the *object-oriented Bayesian network* framework of [9], and show that our language allows us to describe such systems in a natural and modular way. Our language supports a natural representation for certain system characteristics that are hard to capture using more traditional frameworks. For example, it allows us to represent systems where some processes evolve at a different rate than others, or systems where the processes interact only intermittently. We provide a simple inference mechanism for our representation via translation to Bayesian networks, and suggest ways in which the inference algorithm can exploit the additional structure encoded in our representation.

1 Introduction

Consider the problem of representing and reasoning about a complex system such as a computer network, a factory, or a busy highway system. We may be interested in estimating the current status of the system, in predicting its behavior over the near future, or in understanding the cause for a certain sequence of observations. These applications, as well as many others, have some common characteristics. First, they require that we represent the system and its evolution over several time points. Second, there is significant uncertainty over the projected evolution of the system, leading us to prefer stochastic models, which provide an explicit representation for the various possibilities and their likelihoods. Finally, the system represented is usually quite complex. It is composed of several subprocesses that interact with each other; the subprocesses can, in turn, be decomposed into yet finer grained processes. Thus, for example, a computer network is usually composed of several sub-networks, each of which is composed of servers and clients, and so on.

How do we represent such a stochastic dynamic system? A standard approach, in the AI literature, is to use a *dynamic Bayesian network (DBN)* [3]. A DBN is a temporally-extended version of a Bayesian network (BN) [11], and shares many of the same advantages. It provides a natural representation of the different states of the process via a set of time-indexed random variables, clear and coherent probabilistic semantics for our uncertainty, and a compact representation of our probabilistic model using a set of *conditional independence assumptions* (including the Markov assumption).

Unfortunately, despite their appealing properties, DBNs are not ideally suited for the type of task that concerns us here. In many real-life domains, including the ones above, the high-level processes are structured entities, each composed of lower-level processes. These processes interact, thereby probabilistically influencing each other. However, most of the "activity" of a process is internal to it, and therefore encapsulated from the rest of the world. Simon [13] has observed that this type of hierarchical decomposition of complex systems in terms of weakly interacting subsystems is extremely common. For example, in a highway system [6], most of the activity inside a car is local to it; only limited aspects, such as lane occupancy and speed, influence other cars on the highway. DBNs do not support the notion of a process, far less the ability to refer to its interactions with others. Our inability to refer to components of the process also prevents us from declaratively reusing parts of the model that occur more than once. For example, if our computer network uses many servers of the same type, we would like to utilize the same model for all of them.

Our earlier work on the *Object-Oriented Bayesian Network (OOBN)* framework [9] addresses these problems for the case of static models. This framework supports decomposition and modularity by defining the domain in terms of a set of *objects*. An object has attributes, which can be simple variables (such as the speed of a car), or complex attributes which are themselves objects (e.g., the car's engine). Complex objects can depend on other objects via a set of *input* attributes, and can affect other objects via a set of *output* attributes. By defining classes of objects, the OOBN framework also supports reuse of model fragments for multiple objects.

Unfortunately, the basic OOBN framework cannot be di-

*Current address: Institute of Computer Science, The Hebrew University, Givat Ram, Jerusalem 91904, Israel. nir@cs.huji.ac.il.

rectly applied to the task of representing interacting processes. An OOBN object is a black box that does not interact with the rest of the world except via its inputs and outputs. Its probability model is a stochastic mapping from the inputs to the outputs. To avoid a cyclic mapping, the framework requires that objects can be ordered so that the inputs of an object precede it in the ordering. Obviously, this requirement prohibits us from describing interacting objects, where each takes as input some of the other's outputs.

To avoid these shortcomings, we define a new representation, called *dynamic OOBNs (DOOBNs)*, which applies the hierarchical decomposition approach of OOBNs to evolving processes. We allow a process to be hierarchically decomposed into subprocesses, which interact repeatedly over time. We show how we can perform inference in our framework using standard inference tools for BNs and OOBNs.

Our representation allows us to obtain many of the advantages of OOBNs in the temporal setting. We can easily provide modular and reusable specifications of hierarchical, composable Markov models. DOOBNs also allow us to explicitly represent important properties of process-based systems that cannot naturally be described using DBNs. In particular, our framework supports a natural representation for processes that are changing with different *time granularity* and for processes with *sparse interactions*. As we show, the explicit representation of such properties allows us to model the system in terms of a smaller number of variables, thereby potentially making inference more efficient.

2 Object-oriented Bayesian networks

We begin with a brief overview of the OOBN framework [9], on which our current work is based. An OOBN is a hierarchically structured probabilistic model, based on Bayesian networks (BNs). BNs provide a concise representation of a joint probability distribution over a set of variables. A BN is a directed acyclic graph whose nodes are the random variables and whose edges are direct probabilistic dependencies between them. Each variable is associated with a conditional probability table (CPT), encoding the local dependence of a variable on its parents. The complete joint distribution over the set of variables is defined as the product of the conditional probability of each node given its parents [11].

OOBNs extend BNs by allowing structured objects, which are hierarchically composed of other objects. Specifically, an object has a set of attributes. Some of these are simple, and correspond to nodes in a traditional BN. Others are complex, and take other objects as their value. The value of an object is a tuple consisting of the values of all its attributes.

Definition 2.1 : A *simple type* is an enumerated set $\{a_1, \ldots, a_n\}$; a *value* for a simple type τ is any element of τ. A *complex type* is a tuple $\tau = \langle A_1 : \tau_1, \ldots, A_n : \tau_n \rangle$; a value for τ is a tuple $\langle A_1 : v_1, \ldots, A_n : v_n \rangle$ where each v_i is a value for τ_i. The elements A_i are called *attributes* of τ. We require that types are stratified, i.e., non-recursive.

We use the term *object* to denote an entity in the domain. A *simple object* has a simple type. A *complex object* has a complex type τ, and three subsets $\mathcal{I}$, $\mathcal{V}$ and $\mathcal{O}$ of the attributes of τ, where $\mathcal{I}$, $\mathcal{V}$ are a disjoint partition of $\{A_1, \ldots, A_n\}$, and $\mathcal{O} \subseteq \mathcal{V}$. The sets $\mathcal{I}$, $\mathcal{V}$, and $\mathcal{O}$ are called *input* attributes, *value* attributes, and *output* attributes.

The type of an object X is denoted $\mathcal{T}(X)$. An object also has an *output type*, denoted $\mathcal{OT}(X)$. The output type of a simple object τ is simply τ itself. The output type of a complex object X with output attributes $\mathcal{O} = \{A_{j_1}, \ldots, A_{j_k}\}$ is defined as $\mathcal{OT}(X) = \langle A_{j_1} : \mathcal{OT}(\tau_{j_1}), \ldots, A_{j_k} : \mathcal{OT}(\tau_{j_k}) \rangle$. An *attribute chain* for an object X is a (possibly empty) chain $\rho = A_1.A_2.\cdots.A_j$ such that A_1 is an attribute of X, A_2 is an attribute of $X.A_1$, and so on. ∎

Aside from its type, the specification of a complex object includes a *probability model* that describes the conditional probability of value attributes, given the input attributes.

Definition 2.2: Let X be a complex OOBN object with type τ. A probability model $\mathcal{P}$ for X defines the following for each $X.A_i \in \mathcal{V}(X)$.

- If τ_i is simple, $\mathcal{P}$ specifies for A_i a set of *parents* $Par(A_i) = \{\rho_1, \ldots, \rho_k\}$ and a CPT. For each j, ρ_j must be an attribute chain of X such that $X.\rho_j$ is simple. The CPT for A_i maps from joint values of $Par(A_i)$ to distributions over τ_i.
- If τ_i is complex, $\mathcal{P}$ associates with A_i a probability model $\mathcal{P}_i$ and an *input binding* that for each $I \in \mathcal{I}(X_i)$ assigns some object $Y.\rho$ such that $\mathcal{OT}(Y.\rho) = \mathcal{T}(I)$.[1] ∎

The probabilistic model of an object defines the way in which its attributes depend on each other. In order for the probability distribution to be well-defined, it is important that there are no cycles in the dependency graph.

Definition 2.3: The *dependency graph* $\mathcal{G}(X)$ associated with an OOBN object X is a directed graph with a node for each of the attributes of X, and an edge from B to A if

- A is a simple attribute, and $B.\rho \in Par(A)$.
- A is complex, and $B.\rho$ is assigned to one of A's inputs.

An OOBN object X is *well defined* if $\mathcal{G}(X)$ is acyclic. ∎

Given this acyclicity requirement, the probabilistic model for an OOBN object is guaranteed to define a coherent conditional probability distribution.

Theorem 2.4: [9] *The probability model for a well defined OOBN object defines a conditional probability distribution over its value attributes given its input attributes.*

The OOBN framework also allows *classes* of objects to be defined. A class defines the set of attributes for a complex object as well as its probabilistic model. Multiple objects can therefore be defined from the same class, allowing reuse of model components. The encapsulation properties of the OOBN language allow objects from the same class to be used in a variety of contexts. An *OOBN model* is simply a complex OOBN object without input and output attributes. By building on the reusable class definitions, very large structured models can be constructed from a set of modular components.

It is easy to see that an OOBN that has only simple attributes is essentially a Bayesian network. In general, Koller and Pfeffer show how to "unwind" complex OOBN objects into a "flat" Bayesian network that is equivalent to it, in a precise sense. We briefly review this construction.

[1] Input binding results in objects having several names. For example, if $\mathcal{P}$ binds $X.A$ to $X.B.I$, then $X.A$ and $X.B.I$ refer to the same object.

Let X be an OOBN object. We define $\mathcal{S}(X)$ to be the set of simple attributes of X as follows. If X has simple type, then $\mathcal{S}(X) = \{X\}$. If X has complex type, then $\mathcal{S}(X) = \cup\{\mathcal{S}(X.A) : A \in \mathcal{V}(X)\}$. Let $\mathcal{B}$ be an OOBN. It is easy to prove that an assignment of values to $\mathcal{S}(\mathcal{B})$ uniquely determines the value of $\mathcal{B}$. Thus, a distribution over $\mathcal{S}(\mathcal{B})$ determines a distribution over the (complex) value of $\mathcal{B}$.

Definition 2.5: Let $\mathcal{B}$ be an OOBN. The *induced network* $BN(\mathcal{B})$ is a Bayesian network over $\mathcal{S}(\mathcal{B})$, such that the parents of each $X \in \mathcal{S}(\mathcal{B})$ are $Par(X)$ and the conditional probability $P(X \mid Par(X))$ is the CPT of X in $\mathcal{B}$. ▮

Koller and Pfeffer show that the induced network is well-defined and captures the distribution defined by $\mathcal{B}$:

Theorem 2.6: [9] *Let $\mathcal{B}$ be an OOBN, then $P_{\mathcal{B}}(\mathcal{S}(\mathcal{B}))$, the distribution $\mathcal{B}$ defines over simple attributes, is the same as $P_{BN(\mathcal{B})}(\mathcal{S}(\mathcal{B}))$.*

The construction of the induced network allows us to use any standard BN inference algorithm to answer queries about the distribution described by an OOBN. However, OOBNs give us the ability to represent models that are much larger and more complex, thereby raising the problem of inference in these very large networks. Luckily, we can exploit the encapsulation properties of OOBNs to guide the application of inference procedures on the induced BN. That is, the input and output attributes of an object "shield" its internals from rest of the OOBN. More precisely, we can define as follows:

Definition 2.7: The *interface* of an OOBN object X, denoted *Interface*(X), contains all the simple attributes in $\mathcal{I}(X)$ or in $\mathcal{O}(X)$. Note that the interface of a simple object is the object itself. ▮

The interface of X can be used to separate the inference for simple objects within X from the inference in the rest of the model. In particular, we can divide the inference process for $BN(\mathcal{B})$ into a separate computation for each complex object X, one which need only look at the simple variables in $Interface(X) \cup \bigcup_{A \in \mathcal{V}(X)} Interface(A)$. Koller and Pfeffer define a cost function $Cost(X)$ that measures the complexity of this computation for an object X, one which is (in the worst-case) exponential in the number of simple variables in $Interface(X) \cup \bigcup_{A \in \mathcal{V}(X)} Interface(A)$. They show that

Theorem 2.8: [9] *The complexity of the inference in $BN(\mathcal{B})$ is $O(\sum_X Cost(X))$.*

This result is important, since it provides guarantees about the complexity of inference in an OOBNs based on properties of the object classes that are used to construct it.[2]

3 Interacting objects

While OOBNs are useful for describing hierarchical models of objects, they are limited by their inability to describe interacting processes that mutually influence each other. Consider, for example, a pair of computational processes, e.g., a client process and a server process, who go through several rounds of communication. If we were to draw a high-level dependency graph, as above, we would see that the state of each process depends on that of the other.

Introducing cycles into an OOBN may lead to an incoherent probability model, just as it does for BNs. However, a cyclic description of a process at a high level may obscure an acyclic process at a finer level of granularity. In the client-server example, it is clear that there are many possible acyclic models at the finer model: the client sends the server a request, the server in turn sends a result to the client, influencing its next query, and so on.

We could, of course, accommodate such models simply by dropping the acyclicity requirement for the dependency graph of an object. In order to guarantee coherence of our model, we could simply test, on a case-by-case basis, whether an OOBN object defined an acyclic model at the lowest level of granularity, i.e., whether $BN(\mathcal{B})$ is acyclic.

This approach, while straightforward, defeats the purpose of providing modular, composable models of objects. There would be no guarantee that a set of objects, each of which has a coherent model of its own, would compose into a coherent object at a higher level. We therefore take a different approach, that allows an object to provide guarantees about its own internal ordering. Each output of an object X will declare the inputs on which it depends. When X is used in a higher-level object, the only thing required to ensure an acyclic process is that no output of X can influence any of the inputs on which it depends. In other words, the output cannot be used before the inputs on which it depends are supplied.

Definition 3.1: An *interacting object* X is an OOBN object, as before, except that $\mathcal{P}$ associates complex attributes with *interacting* objects, and that $\mathcal{P}(X)$ defines a *dependency model* d_X which is a function from $\mathcal{O}(X)$ to subsets of $\mathcal{I}(X)$. ▮

Intuitively, if X is an interacting object, the value of an output attribute $X.O$ depends only on the inputs specified in $d_X(O)$.

To ground this intuition, consider the interacting OOBN in Figure 1(a), representing a simple client/server system. There are two computations to be processed, and two servers available to process them. An allocator receives requests from each of the computations and decides what job to send to each server. The two servers each have the simple attribute FREE, indicating whether the server is available for computation, and the complex output attribute RESULT, which in turn has the simple attributes COMPLETE and SIZE. The servers also take as input the complex attribute JOB, which has simple attributes PRIORITY and COMPLEXITY. Presumably, the servers will also have various other complex attributes representing their components, such as their processors, memory and hard drive, but they are not shown in this simple model. Note that we can use the same model to describe the two server objects, and similarly for the computation objects. A computation object has the simple output attribute SUCCESS, the complex value attribute RESULT (of the same type as SERVER.RESULT), and a complex attribute QUERY. It also has a simple input RESPONSE, which is a notification from the allocator as to which process, if any, is serving its request.

The model of Figure 1(a) is not a legal OOBN, since there are numerous cycles: for example, between each computation and the allocator, and between each server and the allocator. All the cycles are broken, however, because the QUERY output of the computation objects and the FREE output of the server objects do not depend on any

[2] Note that one of the complex objects is the OOBN $\mathcal{B}$ itself. In particular, if the OOBN corresponds to a standard BN, then $Cost(\mathcal{B})$ is simply the cost of inference in the BN.

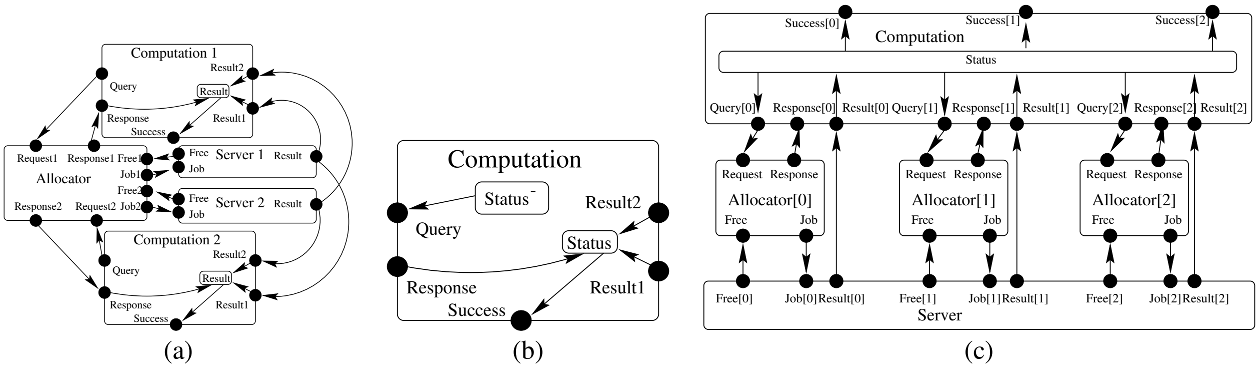

Figure 1: (a) A high-level picture of an interacting OOBN. (b) The DOOBN model for a computation object. (c) The interacting OOBN for 3 time slices of the client-server DOOBN.

of their inputs. The model can express this fact by asserting that their dependency lists are empty. The dependency list for COMPUTATION.SUCCESS, on the other hand, is {RESPONSE, RESULT1, RESULT2}.

Recall that the dependency graph of an OOBN object ensures that the probability model of the object does not contain cycles. Since an OOBN treats an enclosed object as a black box, Definition 2.3 assumes that all outputs of an object are dependent on all of its inputs. We now modify this definition to use the declared dependency model of the contained objects of X to get a more refined representation of the dependencies among attributes. To do so, however, we need to define the graph over input/output attributes of objects rather than over complete objects.

Definition 3.2: The *dependency graph* $\Gamma(X)$ of an interacting object X is a directed graph whose nodes are the input attributes of X, the simple value attributes of X, and the input and output attributes of complex value attributes of X. There is an edge from node ν to node μ in $\Gamma(X)$ if:

1. $\nu.\rho \in \mathit{Par}(\mu)$, for a simple attribute μ.
2. μ is of the form $A.I$ (where we use I to denote input attributes and O to denote output attributes), and some expression of the form $\nu.\rho$ is assigned to $A.I$ by $\mathcal{P}(X)$.
3. μ is of the form $A.I$, ν is of the form $B.O$, and B was assigned to $A.I$.
4. μ is of the form $C.I$, ν is of the form $C.O$, and I is in $d_{X.C}(O)$. ∎

The cases (1) and (2) are similar to the cases we saw in Definition 2.3, except that in case (2) the edge points to $A.I$ rather than to A as a whole. Case (3) deals with situations in which an entire top-level attribute B is assigned to $A.I$; there, any of B's output attributes may be used within A, so we must have an edge between each of them and $A.I$. Case (4) simply reflects the dependencies reported by the complex attributes. This graph allows us to state the basic definitions that force an interacting object to define a coherent probability distribution.

Definition 3.3: An object X is said to *respect* d_X if, whenever there is a causal path in $\Gamma(X)$ from an input attribute $X.I$ to an output attribute $X.O$ or to one of $X.O$'s attributes (if $X.O$ is complex), then $I \in d_X(O)$. An interacting object X is *coherent* if $\Gamma(X)$ is acyclic, X respects d_X, and all complex attributes of X are also coherent. ∎

We can now state the main theorem of this section:

Theorem 3.4: *If X is a coherent interacting object, then X defines a conditional probability distribution over its value attributes given its input attributes.*

Thus, the use of declarative dependency models allows us to extend OOBNs for interacting objects, without requiring other changes to the representation. The main difference is that the semantics of an interactive object is no longer in terms of one stochastic map from inputs to outputs, but rather as a more complex collection of such mappings. In spite of the different semantics, we still perform inference by constructing the induced BN; in fact, the construction of Definition 2.5 applies unchanged.

Theorem 3.5: *Let $\mathcal{B}$ be a coherent interacting OOBN. Then, the network $BN(\mathcal{B})$ is well-defined, and $P_{\mathcal{B}}(\mathcal{S}(\mathcal{B}))$ is the same as $P_{BN(\mathcal{B})}(\mathcal{S}(\mathcal{B}))$.*

As for OOBNs, the ability to construct a ground-level BN provides us with an inference algorithm for the richer representation. More importantly, the encapsulation properties of objects also remain unchanged. Technically, it is easy to show that the value of an object X is independent of the remaining objects in the OOBN (those not enclosed in X) given values for X's inputs and outputs. Thus, as for OOBNs, we can utilize the structure encoded in our representation to provide guidance to the inference algorithm. Using the same definitions for *Interface* and *Cost*, we can thus show:

Theorem 3.6: *The complexity of the inference in $BN(\mathcal{B})$ is $O(\sum_X \mathit{Cost}(X))$.*

4 Dynamic OOBNs

Interacting OOBNs allow us to describe complex interacting objects. Yet, they are still unwieldy for dealing with tasks such as modeling a computer network or a freeway system, which involve *repetitive* interactions. In this section we introduce a compact representation for such domains.

Let us reconsider the example of the previous section, but now under the assumption that the computations are long-lived: they generate a stream of queries and wait for one to be answered before sending the next. In each round, the allocator will attempt to dispatch incoming queries to the servers. The servers themselves may take several rounds to process a query.

To capture such repetitious interaction patterns between objects, we define *dynamic OOBNs* (DOOBNs), which have the same relationship to OOBNs as dynamic Bayesian networks (DBNs) have to BNs. In a DOOBN, we have dynamic objects whose state—the values of its objects—changes over time. We define the dynamics of the object using a standard *Markov assumption*, i.e., that the future state of the process is independent of its past given its future. As usual, this assumption allows us to represent the process via a *transition model* that describes the conditional probability of the current state given the current inputs and the previous state.

The objects in a DOOBN are of two kinds: *persistent* and *transient*. A persistent object exists for the entire duration of the system; its state changes over time, but its state at one time point can influence its state in the future. We often refer to persistent objects as *processes.* A transient object X, on the other hand, has a limited lifetime; every time the system changes, X is replaced by a new object of the same type. In our running example, the computation and server objects have state, and so they are persistent. On the other hand, queries and results, which are communicated between the various units, are transient. They are replaced by new queries and results every round of communication.

Definition 4.1: A *dynamic object* X is declared to be either *transient* or *persistent*. A transient object is exactly an interacting OOBN object. A persistent object is defined in the same manner as an interacting object, except that it also allows parents of simple attributes and inputs of complex attributes to be of the form $A.\rho^-$, where A is a value attribute of X. Additionally, the model $\mathcal{P}$ of a persistent object can associate attributes with either persistent or transient objects. However, input and outputs attributes must be transient.

The dependency graph $\Gamma(X)$ for a dynamic object X is the same as that for an interacting object. The definitions of when X respects d_X and when X is coherent are the same for dynamic objects as for interacting objects. ∎

Intuitively, the attribute $A.\rho^-$ refers to the value of the object that was assigned to $A.\rho$ at the preceding point in time. (Recall that $A.\rho$ must be a transient object.) To make this definition precise, we assume that a process changes state at fixed time intervals, each of which is called a time slice. Thus, $A.\rho^-$ is simply the value of A at the previous time slice. We now get the following as a corollary of Theorem 3.4.

Theorem 4.2: *If X is a coherent dynamic object, then X defines a conditional probability distribution over its value attributes given its input attributes and its previous value.*

Figure 1(b) shows the DOOBN model for a computation object. Since the computation is persistent, it may itself contain persistent attributes, and in fact the STATUS attribute, representing the accumulated result of computation, is persistent. STATUS has, among other things, the simple attribute WORK-TO-GO. The current query depends on the status at the previous time slice, while the status at the current time slice is affected by the current result.

To fully describe an evolving process, we must provide a starting point as well as a transition probability.

Definition 4.3: A *Dynamic OOBN* (DOOBN) is a pair $\mathcal{D} = \langle \mathcal{D}_0, \mathcal{D}_{\rightarrow} \rangle$, where $\mathcal{D}_{\rightarrow}$ is a dynamic object without inputs, and $\mathcal{D}_0$ is an (interacting) OOBN object of the same type as $\mathcal{D}_{\rightarrow}$. For simplicity, we also assume that $\mathcal{D}_0$ and $\mathcal{D}_{\rightarrow}$ have the same dependency model d. ∎

Intuitively, $\mathcal{D}_{\rightarrow}$ describes the transition probability from one time slice to the next, while $\mathcal{D}_0$ is an OOBN (that therefore lacks the A^- attributes) that describes the distribution over the initial state. As a whole the DOOBN model describes a complex Markov process, where the value of the process at each step is of type $\mathcal{T}(\mathcal{D}_{\rightarrow})$.

To reason about such a process, we need to reason about the values of the process at different points in time. To do so, we *unroll* the DOOBN into an interacting OOBN that has many copies of the same objects, one for each time slice. To unroll a DOOBN for N time slices, we unroll each of its attributes. A transient object is duplicated N times; a persistent one is unrolled recursively. The inputs and outputs are then connected appropriately by the enclosing object.

Definition 4.4: Let $\mathcal{D}$ be a DOOBN, let X be a persistent object in $\mathcal{D}$, and let $N > 0$ be the number of non-initial time slices. The unrolled object $\mathcal{U}_N(X)$ is an interacting OOBN object defined as follows:

- If A is a transient attribute of X, then $\mathcal{U}_N(X)$ contains an attribute $A[0]$—a copy of $X.A$ in $\mathcal{D}_0$—and N attributes $A[1], \ldots, A[N]$, each of which is a copy of $X.A$. If $\mathcal{D}_{\rightarrow}$ (or $\mathcal{D}_0$ for $k = 0$) assigns $B.\rho$ to $A.I$ or a parent $B.\rho$ to A, then $\mathcal{U}_N(X)$ assigns $\sigma_k(B.\rho)$ to $A[k].I$ or to $\mathit{Par}(A[k])$ respectively. If A is simple, the CPT of $A[k]$ is the same as that of A.
- If A is a persistent attribute of X, then $\mathcal{U}_N(X)$ contains the single attribute $\mathcal{U}_N(A)$. If $B.\rho$ is assigned to $A.I$ in X, then a *time k snapshot* $\sigma_k(B.\rho)$ is assigned to $\mathcal{U}_N(A).I[k]$.
- The dependency list $d_{\mathcal{U}_N(X)}(O[k]) = \{I[j] \ : \ j < k, I \in \mathcal{I}(X)\} \cup \{I[k] : I \in d_X(O)\}$.

The definition of $\sigma_k(B.\rho)$ is straightforward: $\sigma_k(B.\rho^-) = \sigma_{k-1}(B.\rho)$; if B is transient, $\sigma_k(B.\rho) = B[k].\rho$; otherwise, letting $\rho = O.\rho'$, $\sigma_k(B.\rho) = B.O[k].\rho'$. ∎

At its most basic, the construction generates a copy of each simple attribute for each time slice. These attributes are encapsulated in a way that matches the process structure: persistent attributes are represented by a single long-lived subobject, while transient attributes have a separate object for each time slice. Figure 1(c) shows part of the interacting OOBN for 3 time slices of the DOOBN for our example.

Theorem 4.5: *If $\mathcal{D} = \langle \mathcal{D}_0, \mathcal{D}_{\rightarrow} \rangle$ is a DOOBN where $\mathcal{D}_0$ is a coherent interacting OOBN object and $\mathcal{D}_{\rightarrow}$ is a coherent DOOBN object, then $\mathcal{U}_N(\mathcal{D})$ is a coherent interacting OOBN.*

If we now take the resulting interacting OOBN, and apply the transformation described in Theorem 3.5, we obtain a BN with $N + 1$ time slices, each containing the simple attributes in $\mathcal{D}_{\rightarrow}$; These are connected appropriately both within and between time slices. This transformation is illustrated in Figure 2, which shows two time slices of the BN constructed from the DOOBN for our example. (For simplicity, the model for the size of the result has been omitted.) Recall that the same simple attribute can have a number of names, because values are passed from the outputs of one object to the inputs of another. In the BN, we use the name of the attribute at the time it is created. For example, COMP.STATUS.WORKTOGO depends on COMP.RESULT.COMPLETE, which is the

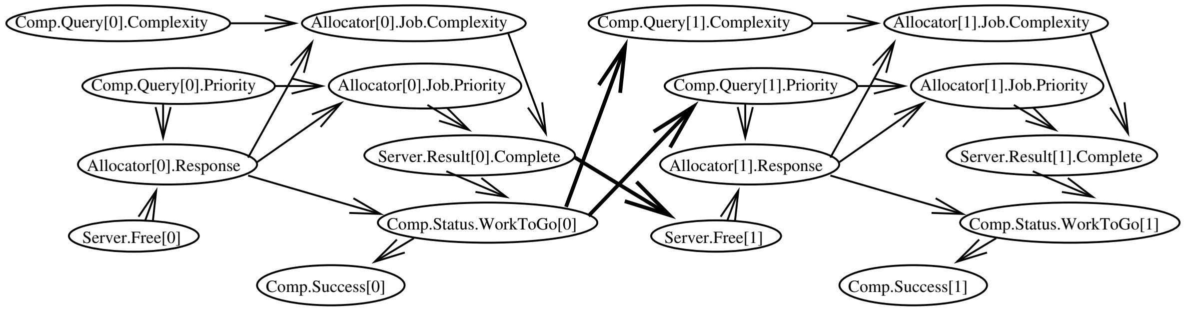

Figure 2: The DBN for 2 time slices of the client-server DOOBN. Edges between the time slices are shown in bold.

same as SERVER.RESULT.COMPLETE. This is represented by an edge from SERVER.RESULT.COMPLETE to COMP.STATUS.WORKTOGO in each time slice.

Since the BN was built from an interacting OOBN, Theorem 3.6 on the cost of inference still applies. Since the number of objects in an N-time slice interacting OOBN is linear in N, one might think that the cost of inference is also linear in N. Unfortunately, this is not the case—the cost of inference in the subobjects grows exponentially with N. The reason is that the interface to a persistent object contains $N + 1$ copies of its inputs and outputs, and the cost of reasoning in an OOBN is exponential in the size of the largest interface.

Of course, there is no requirement that we use the guidance given to us by object boundaries in performing the inference. After all, the resulting BN structure is a familiar one: it is exactly the type of structure that we obtain by unrolling a DBN for N time slices; perhaps we can employ standard DBN inference algorithms. As we mentioned above, the key to most BN algorithms is the separation of the inference problem into two pieces using a set of variables that render the pieces conditionally independent (e.g., the interface of an object in OOBNs). DBN inference algorithms, even the most sophisticated [8], rely on the same basic idea. Largely, DBN inference algorithms focus on *Markovian separators*, which separate the future of the process from its past. Unfortunately, in order to render the future and past independent, our separator must contain some set of variables that block all paths through which influence can flow. In Figure 1(c), for example, a Markovian separator would have to "cut" both the computation process and the server process, rendering their future independent of their past. This separator would have to contain (at least) all the simple variables in these processes whose value at the previous time step is referenced. For complex processes the number of such variables may be quite large. The inference algorithm has to maintain a joint distribution over the separator, rendering the cost exponential in the number of variables in it.[3] Since we are interested in reasoning about fairly complex systems, using Markovian separators is often infeasible.

It would appear that we are faced with two unpalatable alternatives. We can either use a method that is exponential in the number of time slices, or one that is exponential in the size of a time slice. If both are large, neither option will be feasible. This problem also occurs in traditional DBNs, and approximate algorithms for DBN inference is an important area of current research [4, 7, 2]. The work of Boyen and Koller [2] is particularly relevant to our discussion, as it explicitly utilizes a decomposition of a complex process into weakly interacting subprocesses. They utilize this decomposition to approximate the joint distribution over a Markovian separator by assuming that the states of the subprocesses are independent. They then provide bounds on the error incurred by approximation using the degree to which the different subprocesses interact. In their current work, the decomposition of the process into subprocesses is given as input to their algorithm, presumably by some user. Our DOOBN representation makes this structure explicit, allowing their algorithm to take advantage of the process structure automatically.

5 Time granularity

By making the notion of a process explicit, we obtain the ability to refer directly to its properties and distinguish them from the properties of other processes in the system. In particular, our framework gives us the tools to avoid one of the main deficiencies of the DBN representation. In a standard DBN model, we begin by picking some fixed granularity of time, and then generating multiple instances of each state variable in our domain, one instance for each time slice. For example, in the traffic surveillance application of [6], accurate tracking requires that the locations of vehicles be modeled at the rate of 30 time slices a second. All other variables in the system are therefore modeled at the same level of granularity. However, most of these variables, e.g., the weather or the alertness of the driver, evolve much more slowly than the car's location. Clearly, had the presence of the location variables not forced them to do so, it would not have occurred to the designers of [6] to model these other variables so finely.

What we want is a framework that allows us to model different parts of the system at different levels of time granularity. One could imagine trying to extend the DBN framework directly in order to achieve this goal. For example, we could annotate each variable with a time granularity, using

[3] One might think that, in highly structured processes, it would be possible to use conditional independence to decompose the joint over the separator. Unfortunately, this is not the case. Except for the first few time slices, all of the variables in a Markovian separator are correlated. Intuitively, for almost any two variables, there is an influence path leading back through earlier time slices, that renders them dependent.

that number to dictate the frequency with which it is modeled. Aside from leading to unwieldy notation, such a solution suffers from a serious knowledge engineering problem: DBNs provide no notion of a high-level process consisting of several variables; thus, if we decide to change the time granularity at which one of our processes is modeled we have to manually change the annotation on all of the relevant variables.

We now describe how to represent such situations by making a small extension to the basic DOOBN model. We simply add an optional additional granularity argument, $\delta(X)$, to each persistent attribute X in the DOOBN, that denotes how frequently we should model the process represented by X. If X is not labeled with a granularity value, it simply inherits its granularity value from its enclosing object.

Let us analyze the behavior of a process containing processes at different granularities by examining our transformation from Definition 4.4. The number of times that a transient object is duplicated depends on $\delta(X)$, where X is the enclosing process. We start with one copy at time 0; each instance lasts for exactly δ seconds, at which point a new instance is generated from the appropriate transition probability model. Thus, the kth "time slice" of the object maintains its value over the half open interval $[k\delta, (k+1)\delta)$.

In this context, it no longer makes sense to index time-specific instances of an object A with a natural number k representing its time slice. Instead, we use $A[t_1, t_2]$ to index an object by the interval over which its holds. We use the notation $A[t]$ as a shorthand reference to the (unique) time indexed object $A[t_1, t_2]$ for which $t \in [t_1, t_2)$.

The probability distribution over the values of a transient object depends on values of other objects, both from the current time and from the previous time. To understand the nature of this dependence, consider the (imaginary) process by which a new value for A is chosen at time t_1. A parent B of A represents a dependence of $A[t_1, t_2]$ on a contemporaneous variable. Thus, the appropriate parent for $A[t_1, t_2]$ would be $B[t_1]$—the value of B at the time A's value is selected. On the other hand, a parent of the form B^- represents a dependence of A on the value of B at a preceding moment in time. We choose to interpret B^- as referring to the closest preceding moment, i.e., $B[t_1 - \epsilon]$ for arbitrarily small ϵ. Intuitively, A looks at the value of $B.\rho$ immediately before t_1, and then changes its value at t_1. Clearly, this interpretation is not the only legitimate one. In particular if some of A's inputs evolve much more quickly than A, we may wish to have A depending on an their earlier values, or perhaps on some appropriate aggregate value. Space limitations prevent us from discussing these ideas, but the extensions are straightforward and introduce no new subtleties. We now formalize this intuition by extending Definition 4.4.

Definition 5.1: Let X be a persistent DOOBN object, and let $T > 0$ be some time point. Let δ be the time granularity of X and $N = \lfloor T/\delta \rfloor$. The unrolled object $\mathcal{U}_T(X)$ is simply $\mathcal{U}_N(X)$, with the following changes:

- Any attribute in $\mathcal{U}_N(X)$ of the form $A[k]$ is renamed to $A[k\delta, (k+1)\delta]$.
- The function σ_k is replaced by the function $\sigma_t(B.\rho)$, defined as follows: if B is transient, $\sigma_t(B.\rho) = B[t].\rho$; otherwise, letting $\rho = O.\rho'$, $\sigma_t(B.\rho) = B.O[t].\rho'$. $\sigma_t(B.\rho^-)$ is defined to be $\sigma_{t-\epsilon}(B.\rho)$ for an ϵ so small that $\sigma_{t-\epsilon}(B.\rho) = \sigma_{t-\epsilon'}(B.\rho)$ for any $\epsilon' \in (0, \epsilon)$. ∎

Theorem 5.2: *If $\mathcal{D} = \langle \mathcal{D}_0, \mathcal{D}_\rightarrow \rangle$ is a DOOBN with time granularities where $\mathcal{D}_0$ is a coherent interacting OOBN object and $\mathcal{D}_\rightarrow$ is a coherent DOOBN object, then $\mathcal{U}_T(\mathcal{D})$ is a coherent interacting OOBN.*

The proof is based on defining the set of time points at which some object in the model determines its value. Although this set of time points no longer has regular structure, it is still finite. The same techniques as in the previous section can be used to show that the probability model over the variables at these time points is acyclic, and that the conditional distribution is well-defined given the values at the previous time point.

If we now take the resulting interacting OOBN, and apply the transformation into a BN described in Theorem 3.5, the result is still a BN, but it is no longer a standard DBN. Variables that evolve more slowly have fewer copies than their more volatile counterparts. The dependencies between variables are resolved automatically by our construction process.

While modeling processes at their appropriate granularity is a worthwhile goal on its own, it can also provide some help with the inference task. As we saw in the previous section, using the OOBN structure to perform inference in a DOOBN is exponential in the number of time slices. However, if some processes evolve much more slowly than others, their interfaces will be fairly small. Therefore, it may make sense to use interface-based separators, at least for slowly changing processes. The resulting fragments might involve fewer processes, and thus would allow for efficient use of Markovian separators. A combined approach that uses both kinds of separators suggests itself. This topic deserves a full investigation, which is beyond the scope of this paper.

6 Sparsely interacting processes

Up to now, we have assumed that the interaction between processes is regular; at each point in time, they receive inputs from other processes, change their state, and export their outputs. Clearly, this type of regular communication is not an aspect of all systems. In our client/server example, it is reasonable to assume that a server output representing the size of the query result influences the client state only if the client has issued a query to the server at some previous time point, and if the server has just indicated that the processing is complete. If queries are only issued rarely and their processing takes a variable amount of time, the interaction model is far from regular. As another example, consider modeling cars traveling on a highway. While the behavior of one car is definitely influenced by attributes of others, this influence might be intermittent; e.g., a driver might examine the status of cars on the left only when he tries to change lanes.

Such situations are represented very naturally in the *transition system* framework of Manna and Pnueli [10]. There, each process can take one of several possible transitions, which affect its state. However, each transition is associated with a *guard*, which may or may not be true in a given state. A transition is said to be *enabled* in a state if its guard is true at that state; at a given state, a process can only take a transition if it is enabled. In our client/server example, we may have an additional *processing-complete* variable in the server and a *query-issued* variable in the client. The transition which updates the client's state based on the query results would be enabled only if both of these variables are true.

We can extend this idea to our probabilistic framework by using the techniques of Boutilier *et al.* [1]. They define a notion of *context specific independence (CSI)*, which corresponds to situations in which the variables on which some variable depends are different in different situations (or contexts). They also show how these situations can be represented explicitly within the language, by using structured CPT models such as trees. In our example, the state of the client object will only depend on the server's other output variables if both of these variables are true.

Boutilier *et al.* also propose ways in which the CSI structure can be exploited to speed up BN inference. We believe that these ideas can be extended to DOOBN models, allowing our inference algorithm to take advantage of sparse interaction between the processes. We conclude this section by sketching one possible mechanism by which computational advantage can be gained.

Recall that applying the OOBN inference idea to models generated from DOOBNs led us to consider the use of process interfaces to decompose the BN computation. When interactions between the processes are sparse, the actual "information" content of the interface is much smaller. For example, assume that our client is unlikely to issue very many queries to the server, perhaps because it intersperses queries with other (lengthy) tasks. We can define an *active interaction* to be a state in which both the *query-issued* and *processing-complete* variables are true, so that the client state is influenced by the server. At a time slice where there is no active interaction, none of the other server outputs affect the client; otherwise, the client state depends on some set of ℓ variables describing the query results. If we know that there are exactly k "active interactions" in N time slices, there are only $\binom{N}{k}$ interaction patterns, and a total of $\binom{N}{k}2^{k\ell}$ possible interactions between the server and the client. Using a process of *global conditioning* [12], we can do a case analysis on the different possible interactions, thereby separating the client and server processes. The cost of this separation is $\binom{N}{k}2^{k\ell}$, which (for small k) is exponentially smaller than the $2^{N\ell}$ required without using the sparsity. We can also extend these ideas to situations where we have no specified bound on the number of interactions. If we only know that "runs" where there are many queries are highly improbable, most of the probability mass would be on those runs where the interaction pattern is sparse. The bounded conditioning algorithm of [5] can then be used to restrict attention only to runs with sparse interaction.

7 Conclusion

In this paper, we have presented a new language for representing complex dynamic systems with uncertainty. The language supports structured hierarchical representations of systems in terms of interacting subprocesses, and allows large models to be constructed from modular, reusable components. Our language can express important aspects of complex systems, such as the different rates at which various processes evolve, and the sparse interactions that take place between processes. Our framework also allows additional features to be incorporated easily; for example, we may be able to represent asynchronous stochastic systems by extending our time granularity framework to allow for stochastic models of when processes *wake up*.

We provide an inference algorithm for answering queries over models in our language by converting them to Bayesian networks. Unfortunately, as with standard DBNs, inference in DOOBNs can be extremely costly. However, as our discussion above illustrates, the types of structure that can be expressed in our language can potentially lead to improved inference algorithms. We believe that approximations are crucial for inference in large complex processes, and we hypothesize that the encapsulation structure of our representation can guide approximation methods such as these of [4, 2]. We plan to examine these issues in future work.

Acknowledgments Part of this work was done while Nir Friedman was at Stanford. This work was supported by ARO under the MURI program "Integrated Approach to Intelligent Systems", grant number DAAH04-96-1-0341, by ONR contract N66001-97-C-8554 under DARPA's HPKB program, by DARPA contract DACA76-93-C-0025 under subcontract to Information Extraction and Transport, Inc., and through the generosity of the Sloan Foundation and the Powell Foundation.

References

[1] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proc. UAI*, 1996.

[2] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. Submitted to UAI '98, 1998.

[3] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Comp. Int.*, 5(3), 1989.

[4] Z. Ghahramani and M. I. Jordan. Factorial hidden Markov models. *Machine Learning*, 29, 1997.

[5] E.J. Horvitz, H.J. Suermondt, and G.F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proc. UAI*, 1989.

[6] T. Huang, D. Koller, J. Malik, G. Ogasawara, B. Rao, S.J. Russell, and J. Weber. Automatic symbolic traffic scene analysis using belief networks. In *AAAI*, 1994.

[7] K. Kanazawa, D. Koller, and S.J. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *Proc. UAI*, 1995.

[8] U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proc. UAI*, 1992.

[9] D. Koller and A. Pfeffer. Object-oriented Bayesian networks. In *Proc. UAI*, 1997.

[10] A. Manna and A. Pnueli. *Temporal Verification of Reactive Systems*. Springer Verlag, 1995.

[11] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[12] R. Shachter, S. Andersen, and P. Szolovits. Global conditioning for probabilisitic inference in belief networks. In *Proc. UAI*, pp. 514–522, 1994.

[13] H. Simon. *The Sciences of the Artificial*. MIT Press, 1981.