Integrating Multiple Representations for Incremental, Causal Simulation

Reid G. Simmons
School of Computer Science
Carnegie Mellon University

Abstract

Many engineering problems require the ability to answer "what if" questions about the effects of complex physical events. To address such problems, we have implemented MIDAS, a system that can incrementally simulate events and produce causal explanations describing their effects. MIDAS incorporates an explicit causal model of time, change, and persistence. It integrates multiple specialized representations using a Truth Maintenance System that records belief justifications and enables incremental assertion and retraction of beliefs. MIDAS has been tested in several complex domains, including geology and semiconductor fabrication.

1 Introduction

Simulation of real-world processes is important for many tasks, including monitoring, diagnosis, planning, interpretation, and design. In particular, simulation is needed to predict the effects of events and to verify problem solutions.

This paper describes MIDAS, an incremental, causal simulator that combines multiple representations for efficient simulation of complex physical events. MIDAS (Model-Based, Incremental, Discrete Action Simulator) integrates representations for arithmetic quantities (both qualitative and quantitative), sets of objects, time, temporal objects, logic, relations, and events, all integrated using a Truth Maintenance System (TMS). The simulator has been tested in several domains, including geologic interpretation [12], semiconductor fabrication diagnosis [14], and mobile robot planning.

By a "causal" simulator, we mean that it incorporates an underlying model of how physical events affect the state of the world. The causal model used in MIDAS incorporates explicit representations of time, change, and persistence. The simulator is capable of producing explanations, based on its causal model, that correspond intuitively to our notion of cause and effect. For example, the explanation for "why is this semiconductor layer so thick?" details which events affect the thickness of the layer, the magnitude of the changes, and why the events are believed to have occurred.

By "incremental," we mean that at any time one can add new events, reorder events, and change their parameter values. The incremental nature of the simulator, combined with the causal explanations, enable it to efficiently answer "what if" questions. For example, a user might want to know "what would happen if the duration of event Etch4 were increased slightly?" or "what if the duration increased enough to completely destroy the top layer of the semiconductor wafer?"

Several factors motivate MIDAS' use of multiple representations. First, multiple representations can provide more expressive power, since knowledge about various aspects of a domain (e.g., qualitative and quantitative [10], or definitional and assertional [1]) are often difficult to capture using a single representation. Thus by using multiple representations one can model more complex domains, or can model a given domain with more fidelity. Efficiency is another prime argument for using multiple representations, since inference strategies can be tuned to particular representations and the information required from them. For example, by representing relationships between quantities as a digraph, efficient graph-search techniques can be employed for inferring transitive relationships.

A difficulty in using multiple representations is that they must be integrated somehow. A principled integration involves finding a common framework and common language for the various representations [8]. In MIDAS, the common framework is a justification-based Truth Maintenance System, which is used by all
the specialized representations to record and incrementally modify inferences. The common language used to integrate the representations is a subset of first-order predicate calculus. The logical notation, incorporated in a term data base, is essentially used to pass information between the more efficient specialized representations.

The underlying causal model employed in MIDAS is introduced in Section 2. Section sec:integration describes the framework used to integrate the multiple representations, and Section 4 describes the specialized representations themselves. An example using MIDAS to perform incremental simulation is presented in Section 5.

2 Underlying Causal Model

The MIDAS simulator incorporates an explicit causal model of how events affect the state of the world over time. There are several advantages to having such a model. For one, real-world events may be easier to encode using a causal representation language. Another advantage is that underlying assumptions (such as closed-world persistence assumptions) are explicitly represented, leading to more informative explanations. More importantly, many advanced reasoning techniques, such as model-based reasoning [2], debugging [13], and explanation-based learning [7], need a causal theory and causal explanations of the domain in order to produce accurate results.

The causal model used in MIDAS is based on notions of time, change to objects, and the persistence of attributes over time. MIDAS utilizes a point-based model of time, with intervals defined by their start and end points. While point-based and interval-based temporal representations are essentially equivalent [5], we have found that point-based representations are somewhat more concise and natural for representing the discrete event models used by MIDAS.

To model change, MIDAS uses a concept we call temporal objects. A temporal object consists of a time interval and a set of attributes. The time interval represents the temporal extent of the object (its creation and destruction times). Unlike most frame representations, the attributes of a temporal object do not represent single values, but rather histories of values. Each attribute history contains a set of episodes that encode how the value of the attribute changes over time.

Temporal references are used to refer to the values of attributes at particular points in time. Written as object.attribute@time (or as ( @(attribute object) time) in the prefix notation used by MIDAS), the value of a temporal reference is the value of the last episode in the attribute history prior to time:

\[
\exists (ep : \text{episode}) \left[ \begin{array}{l}
\text{[(ep } \in \text{ obj.attr}) \land (\text{ep.end } \leq t) \land (\text{ep.value } = v)}
\land \neg \exists (\text{ep1 : episode}) \\
\text{[(ep } \neq \text{ ep1}) \land (\text{ep1 } \in \text{ obj.attr}) \land (\text{ep1.end } \leq t) \land (\text{ep1.end } > \text{ep.end})]
\end{array} \right]
\]

The last conjunct in the formula indicates that no other episode affects the attribute between the end of episode ep and time t. MIDAS encodes this persistence of attributes [9] using explicit closed-world assumptions that represent the belief that no other change occurs to the attribute. The persistence assumptions are non-monotonic, so that the value of the temporal reference is updated when new episodes are added to the history.

A characterizing feature of MIDAS' causal model is that the complete evolution of the simulation is encoded in a single, temporally extended state. This has several advantages over the more common representation of change as a sequence, or set, of states. For one, inter-temporal comparisons can be stated directly in the causal language (e.g., "(if (Fl.orientation@tl > F1.orientation@t2) then ...)"), increasing its expressive power. With respect to implementation, a history-based representation of change is typically more compact and efficient, since only causally relevant state changes need to be encoded [16]. Finally, the use of persistence assumptions addresses the frame problem [9] by providing a general mechanism for inferring which attributes do not change over time, and for incrementally updating such beliefs as new changes occur.

3 Integration Framework

The multiple representations used in MIDAS are integrated using a term data base and a Truth Maintenance System (TMS). The term data base provides a logical language and methods for evaluating expressions based on the values of sub-expressions. The TMS provides mechanisms for incrementally asserting and retracting assumptions about the state of the world, justifying beliefs, and producing explanations based on the justifications. Every specialized representation uses the TMS to record the inferences made, and to retract inferences when the underlying assumptions change.

The TMS used is justification-based: each node in
the TMS represents one belief, whose truth in the current context depends on the truth of its supporting nodes. A node is believed when all its supporting nodes are believed, and belief is retracted when any supporting node is disbelieved. This is in contrast to an assumption-based TMS (ATMS), which can represent all contexts at once, but is much less space efficient [3]. To handle multiple contexts, the TMS provides explicit contexts (sets of assumptions), where switching contexts is relatively inexpensive.

MIDAS groups nodes representing similar beliefs into cells, which are based on deKleer's notion of class nodes [3]. For example, a cell is used to represent the relationship between two quantities, A and B. The nodes associated with the cell represent the actual relationships between the quantities (e.g., A < B, A ≥ B). The value of a cell is determined by which node of the cell is currently believed. If more than one node associated with a cell is believed, and the node values are inconsistent, the TMS signals a contradiction (e.g., believing A > B and A = B is contradictory, but believing A > B and A ≥ B is not).

The TMS provides the important capability of adding rules to nodes and cells. Rules on cells fire whenever a new node is added to the cell; rules on nodes fire when the node is believed. The rules typically add new nodes and/or justifications to the TMS. The rule semantics closely follow those described in [3].

The term data base complements the TMS by providing a common language for integrating the representations. A term consists of a logical statement and a value cell. Associated with each relation symbol is an evaluation method that sets up rules on the value cells of the sub-terms. The rules fire whenever values for all the sub-terms are known, and act to either a) infer a value for the term, or b) maintain consistency between representations. For example, associated with the symbol “+” is a rule that determines the sum of two quantities. Thus, evaluating the term (+ A 2) computes a value for the term whenever values for A and 2 are known. If the value for A changes, the TMS automatically retracts the current value, and a new rule firing produces a new value for the sum.

Term data base rules are also used to maintain consistency between representations. For example, suppose the term (if (A > B) (E E S1)) is assumed to be true. MIDAS sets up rules so that if the arithmetic reasoning subsystem ever determines that A is greater than B, then the logical expression (E E S1) is inferred to be true, and consequently the set reasoning system adds element E to set S1. Similarly, if S1 were found to be an empty set, then (E E S1) would be believed to be false, which would justify the belief that A is not greater than B.

4 Representations for Causal Simulation

Together, the TMS and term data base are the glue that binds the representations used in MIDAS. MIDAS uses specialized representations for arithmetic, time, sets, objects, logic, relations, and discrete events. The representations each include specialized data structures and algorithms tuned for their particular task. The representations also build on one another where certain functionality is needed (Figure 1). For example, the set reasoning subsystem uses the arithmetic subsystem to reason about the cardinality of sets. Similarly, the logic subsystem uses set reasoning to handle limited forms of quantification. At the highest level, the representation of events utilizes all the other specialized representations to provide a general causal simulation capability.

4.1 Arithmetic Quantities

MIDAS represents real-valued quantities using an arithmetic reasoning system called the Quantity Lattice [11]. The Quantity Lattice efficiently integrates qualitative and quantitative arithmetic information. Qualitative information is represented as a directed graph of ordinal relationships (e.g., <, >, =) between quantities. Quantitative information is represented by associating a real-valued interval with each quantity, where the actual value of the quantity is taken to lie
somewhere between the upper and lower bounds of the interval. As the intervals associated with quantities change, constraint propagation is used to update the intervals of related quantities. For example, if we assert that \( A > B \) and that \( A \) has the interval \([1, \infty)\), then the Quantity Lattice will infer that the interval of \( B \) must lie in the range \((1, \infty)\).

The Quantity Lattice infers ordinal relationships between quantities by 1) performing graph-search using the transitivity of relationships, and 2) reasoning about the intersection of the intervals. For example, from \( A \leq B, B \leq C, \) and \( C \leq A, \) the system can infer that \( A = B = C. \) Similarly, if the upper bound of one interval is less than the lower bound of a second interval, then the first quantity must be less than the second.

The Quantity Lattice also computes arithmetic functions using both qualitative and quantitative information. The quantitative information is used to perform interval arithmetic on quantities. Qualitatively, the system utilizes relationships between arguments of the arithmetic functions to infer new relationships. For example, one rule used by the Quantity Lattice encodes that the relationship between \( X \) and \( Y \) is the same as the relationship between \((X - Y)\) and zero (e.g., if \( X \) is greater than \( Y \), then \((X - Y)\) is greater than zero). This is used in MIDAS, for instance, to infer that etching away a positive amount of material from a semiconductor layer decreases its thickness.

### 4.2 Time

Time points are implemented directly as real-valued quantities using the Quantity Lattice. The special time points \( \text{beginning-of-time} \) and \( \text{end-of-time} \) are predefined, and a constraint is enforced that for every other time point \( t, \) \( \text{beginning-of-time} \leq t \leq \text{end-of-time}. \) In addition, the simulator uses the special time point \( \text{now} \) to denote the end point of a simulation.

Time intervals are defined by their start and end time points \( (\text{I.start} \) and \( \text{I.end}, \) respectively), with the constraint that \( \text{I.end} > \text{I.start}. \) The duration of an interval \( (\text{I.end} - \text{I.start}) \) is also maintained using the Quantity Lattice. By using the Quantity Lattice to encode time points and intervals, MIDAS is able to represent and reason about both metric time and partial orders.

### 4.3 Sets

The SERF system\(^{[15]} \) provides MIDAS with propositional set reasoning. Just as the Quantity Lattice combines qualitative and quantitative reasoning, SERF combines reasoning about relationships between sets with reasoning about set membership. Assertions can be made that one set is a subset/superset of another, two sets are disjoint/overlap, or two sets are total (i.e., their union is the universal set). Similarly, one can assert that an object is a member of a set, with membership assertions being propagated to related sets. For instance, if one asserts that \( Sl \subset S2 \) and \( A \in S2, \) then SERF infers that \( A \in Sl, \) justified by the two premises.

As with the Quantity Lattice, relationships between sets are inferred both by graph-search using transitivity, and by reasoning about the intersection of set members. For example, if \( Sl \) and \( S2 \) have some members in common, they must overlap; if \( Sl \) has a member in common with the complement of \( S2, \) then \( Sl \) cannot be a subset of \( S2. \)

SERF supports the standard set functions of union, intersection, complement, and difference. These functions integrate both the relationship and membership aspects of SERF. For example, \( A \cup B \) is asserted to be a superset of both \( A \) and \( B. \) In addition, all members of \( A \) and \( B \) are constrained to be members of \( A \cup B, \) and all members of \( A \cup B \) which are known not to be members of \( B \) are inferred to be members of \( A \) (and vice versa).

SERF uses the Quantity Lattice to represent and reason about the cardinality of sets, and includes mechanisms for reasoning about the closure of sets. By asserting that a set is closed, one indicates that all its members are known. This enables SERF to make certain inferences that might not otherwise be warranted. For example, if \( Sl \) is a closed set and all the members of \( Sl \) are also members of \( S2, \) then SERF can infer that \( Sl \subset S2. \)

### 4.4 Temporal Objects

Temporal objects, such as geologic formations or semiconductor layers, are represented as frames, with slots for the start of the object (its creation time), the end of the object (which defaults to \( \text{end-of-time}, \) indicating continued existence), and slots for each of the object’s attributes. The objects form a type hierarchy with value inheritance.

As described in Section 2, each of the attribute slots represent histories of values over time. A history consists of an initial value and a set of episodes that represent all subsequent changes to the attribute. Each episode encodes the time interval during which the attribute changes, the cause of the change, and the resulting value at the end of the episode. Figure 2 shows two
histories for the geologic formation $F_1$. Three episodes are indicated: two caused by tilt events affecting the formation's orientation, and one intervening deposition that placed the formation $F_2$ above $F_1$. While for illustrative purposes Figure 2 depicts the episodes as a temporally ordered list, they are actually represented as an unordered set. This is a more general representation since it can handle partially ordered events and the incremental reordering of events.

To determine the value of a temporal reference (e.g., $F_1$.orientation@Deposition1.end), the Quantity Lattice is used to search for the last episode in the history whose end point is not greater than the given time. Once the relevant episode is found, the value of the temporal reference is taken to be the value associated with the episode, justified by the beliefs 1) that the episode occurs, 2) that it occurs before the time point, and 3) the closed-world persistence assumption that no other episode occurs between them. If the persistence assumption is violated when new episodes are added then the support, and hence the value, of the temporal reference is recalculated.

4.5 Logic

A natural deduction method, based on RUP [6], is used to perform propositional deduction over the standard Boolean connectives (and, or, not, if, iff). Deduction is performed efficiently in a forward-chaining manner, with incremental assertion and retraction of facts supported by the TMS. Since the common integration language of MIDAS is logic-based (Section 3), the logic subsystem is used quite heavily to connect inferences between different representations.

To supplement propositional deduction, we have implemented a limited form of first-order quantification. MIDAS can represent both quantified and existential statements with typed variables. Using SERF, objects are grouped into sets (and subsets) according to their types (and subtypes). Based on the set membership, universally quantified statements are expanded to their equivalent conjunctive form. When new objects of the variable's type are created, a new conjunct is added. For example, if $F_1$ and $F_2$ are objects of type "formation", then the statement:

$$(\text{forall } f: \text{formation}) ( > (\text{@} (\text{thickness } f) \text{ now}) 0))$$

expands into:

$$(\text{and} ( > (\text{@} (\text{thickness } F_1) \text{ now}) 0)$$

$$> (\text{@} (\text{thickness } F_2) \text{ now}) 0)).$$

If the set of objects is declared to be closed, then the universal statement itself is implied by the conjuncts, together with the closed-world assumption that no other objects of the given type exist. Existentially quantified statements are handled similarly, using disjunction instead of conjunction.

4.6 Relations

MIDAS provides a general mechanism for defining domain-specific predicates and functions. Functions, in particular, can be defined at various levels of specificity. At the base level, one can just describe the domain and range of the function, in which case MIDAS assigns an arbitrary value whenever the function is evaluated.

At a higher level of specificity, one can declare functions to be monotonically increasing or decreasing with respect to their arguments. These declarations can be parameterized so, for instance, one can declare a func-
tion to be monotonically increasing in its first argument if its second argument is positive, and decreasing if the second argument is negative. Given these declarations, MIDAS will qualitatively interpolate function values based on the argument values. For example, if foo is increasing in its first argument, then the relationship between (foo A B) and (foo C B) is the same as the relationship between A and C. Note that this qualitative interpolation generalizes some of the arithmetic reasoning performed by the Quantity Lattice (see [11]).

At the highest level of specificity, one can provide an explicit definition of the function or predicate in terms of other functions and predicates (in fact, any LISP code can be used to define relations). In the semiconductor domain, for instance, the following defines the amount of time, starting at time T, an etchant E needs to etch through, and completely destroy, a layer L:

(def-function ETCH-DESTROY-TIME ((L layer) (E etchant) (T time)))

:DEFINITION
(and (if (= L THE-ENVIRONMENT)
  (= (etch-destroy-time L E T) 0.0))
  (if (not (= L THE-ENVIRONMENT))
  (= (etch-destroy-time L E T)
  (+ (etch-destroy-time (# (above L) T) E T)
  (/ (# (thickness L) T)
  (etch-rate E (# (material L) T))))))

4.7 Discrete Causal Events

At the top of the representational hierarchy are causal events (Figure 1). MIDAS uses discrete events, where the changes resulting from an event depend on the state of the world at the start of the event, but no information can be inferred about the what happens during the event. While continuous process representations are more expressive [4], they are also more expensive to simulate and have not proven necessary for the domains we have explored.

Events all have temporal extent (the interval during which they occur), and are specified by their parameters, preconditions, and effects. The parameters are described by a typed list of objects, and the preconditions are logical statements that must be true in order for the event to occur.

The body of an event definition consists of a number of change statements, of the form:

(change <type> <history> <magnitude> <event>.)

When a change statement becomes true, a new episode is added to the history, with extent equal to that of the event. As described in Section 4.4, when a new episode is added the temporal references for the history are recalculated and any invalid persistence statements are retracted.

The attribute's value at the end of the event depends on the type and magnitude of the change statement. If type is "=", then the value equals the magnitude. For example, "(change = (above F1) F2 Deposition1)" indicates that the formation F2 is deposited on top of F1. If type is "+" or "-", the change is relative to the value of the history at the start of the event. For example, "(change + (orientation F1) 5 Tilt1)" means that the orientation of F1 increased by 5 degrees as a result of event Tilt1.

Besides individual change statements, the body of an event may contain conditional statements of the form (if <antecedent> <consequent>), universally quantified statements, or conjunctions of them. In addition, events can specify the creation or destruction objects. As an example of MIDAS' event representation language, Figure 3 illustrates our semiconductor "etch" operation (see also [14]).

5 Incremental Simulation

The simulation algorithm itself is very simple: logical forms are evaluated via the term data base, and the inferences made by the rule firings are recorded by the TMS. Typically, the inferences add new terms and nodes, which induces further rule firings and inference.

The user specifies a particular simulation by asserting a) that certain events occur, b) that the events are sequenced in certain ways, and c) that the events have given parameter bindings. For example, one could assert that Etch22 occurs, with parameter bindings Etchant=Nitric-Acid, and Duration=100. Due to the incremental nature of the simulator, these assertions can be made in any order. For example, one could first assert that event Etch22 occurs with parameters Duration=100, and then state that Oxidation20 occurs before Etch22 (i.e., Oxidation20.end < Etch22.start). Since full retraction is supported, one could then retract the assertion that the duration of Etch22 is 100, and assert instead that it is greater than 150, but less than the duration of Etch4.

To be more concrete, Figure 4 illustrates the results of simulating a 48 step isolation-oxide semiconductor process to create a pair of resistors. Using the TMS justifications recorded by the simulator, MIDAS indicates that Etch4 is one of the events affecting the thickness of layer C and, in particular, the thickness depends on
(def-process ETCH
   :PARAMETERS ((etchant etchant) (duration positive-real))
   :EFFECTS
   (forall ($1 : layer)
    (if (exists-at $1 (start etch))
        (and (if (>= (duration etch) (etch-destroy-time $1 (etchant etch) (start etch)))
               (destroyed $1 (end-of etch)))
            (if (and (>= (duration etch) (etch-destroy-time ($1 above $1) (start etch)))
               (etchant etch) (start etch)))
        (if (<= (duration etch) (etch-destroy-time $1 (etchant etch) (start etch))))
            (if (not (= $1 (surface (layer-description $1)) (start etch))))
                (if (and (> (etch-rate (etchant etch) ($1 above $1) (material $1) (start etch)) 0.0))
                    (change - (top-position $1)
                        (* (- (duration etch) (etch-destroy-time ($1 above $1) (start etch)))
                            (etch-rate (etchant etch) ($1 (material $1) (start etch))))
                        (etch-rate (etchant etch) ($1 (material $1) (start etch))))
                        (change - (thickness $1)
                            (* (- (duration etch) (etch-destroy-time ($1 above $1) (start etch)))
                                (etch-rate (etchant etch) ($1 (material $1) (start-etch))))
                                (etch-rate (etchant etch) ($1 (material $1) (start etch))))))))))
   (etchant etch) (start etch))

Figure 3: Semiconductor “Etch” Event

Figure 4: Simulation of Semiconductor Resistors
the bindings for the duration and etchant parameters of the event. To determine the results of changing parameters, we can retract the belief that the duration of Etch4 is Duration4, and assert instead that it equals Duration5, where Duration5 is assumed to be greater than Duration4, but still less than the time needed to etch totally through layer C. The result of this incremental simulation indicates that more of layer C gets etched, and therefore the overall thickness of the layer is less. This incremental change occurs fairly quickly, in about 10% of the time needed to simulate the initial sequence from Etch4 onwards.

If we assume instead that a dramatic change occurs:

\[
(\text{> Duration5} \rightarrow \text{Etch-Destroy-Time C (Etchant Etch4) (Start Etch4)})
\]

that is, the event lasts long enough to destroy layer C completely, then the incremental effects are more pronounced, and much more of the original simulation has to be updated. Even in this case, however, the additional time is still substantially less than what would be needed to simulate from scratch.

6 Conclusions

We have briefly described the MIDAS incremental, causal simulator. MIDAS uses a TMS and term database to integrate specialized representations, including representations for quantities, sets, time, temporal objects, and discrete events. The underlying TMS provides a common framework for recording, retracting, explaining inferences made by the specialized representations. The term database provides a common language for connecting the various representations. The use of multiple representations, in turn, increases the expressive power and efficiency of the simulator, making it feasible to simulate fairly complex and long sequences of events.

MIDAS has been tested in several domains, including geology, semiconductor manufacturing, and mobile robotics. It is built on earlier work of the author, where the simulation capability was augmented with methods for diagnosis [14] and plan debugging [13]. We are currently in the midst of reimplementing those capabilities to integrate smoothly with the incremental simulation capabilities of MIDAS.

Acknowledgements

Many thanks go to Richard Goodwin for assisting in the implementation and testing of MIDAS, and to John Mohammed for his work with earlier versions of the simulator. This work was sponsored by NASA under contract NAGW-1175.

References


