LA-UR-99-1041

Approved for public release; distribution is unlimited.

Title: Using Perspective to Model Complex Processes

Author(s): Kelsey, Robert L.-XCM

Bisset, Keith R.- TSA-5

RECEIVED
SEP 01 1999
OSTI

Submitted to: SPIE AeroSense
April 4-5, 1999
Orlando, Florida

Los Alamos
NATIONAL LABORATORY

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Using perspective to model complex processes

R. L. Kelsey[a] and K. R. Bisset[b]

Los Alamos National Laboratory,

[a]XCM MS-F645, Los Alamos, NM 87545

[b]TSA-5 MS-F602, Los Alamos, NM 87545

ABSTRACT

The notion of perspective, when supported in an object-based knowledge representation, can facilitate better abstractions of reality for modeling and simulation. The object modeling of complex physical and chemical processes is made more difficult in part due to the poor abstractions of state and phase changes available in these models. The notion of perspective can be used to create different views to represent the different states of matter in a process. These techniques can lead to a more understandable model.

Additionally, the ability to record the progress of a process from start to finish is problematic. It is desirable to have a historic record of the entire process, not just the end result of the process. A historic record should facilitate backtracking and re-start of a process at different points in time. The same representation structures and techniques can be used to create a sequence of process markers to represent a historic record. By using perspective, the sequence of markers can have multiple and varying views tailored for a particular user's context of interest.

Keywords: abstraction, perspective, object-based knowledge representation

1. INTRODUCTION

A poor representation of a problem can make the problem difficult to understand and thus difficult to solve. Due to a poor representation there may be missing details and diminished reality, especially in the case of modeling real-world processes. A correct representation of a problem can contribute to more successful problem solving.[1] In a correct representation, the form and function of entities in the problem may need to be more unconventional or novel rather than typical.[1]

An object-based representation and its elements can facilitate the creation of more correct representations. The elements of an object-base representation are not unlike those of object-oriented modeling which include abstraction, encapsulation, modularity, and hierarchy.[2] Object-oriented programming can model systems which "correspond more closely to the 'real-world' "[3] and humans tend to perceive the world as being made of objects rather than actions.[3]

Abstraction is a means of dealing with complexity.[2] Abstraction is used in object-oriented analysis and design, simulation, and other areas. The notion of perspective (when supported in an object-based representation) can be used to create better abstractions. Creating multiple perspectives can provide additional points of view which need not be consistent with one another. This can lend focus and detail to the parts which compose an area of interest. Perspectives help simplify the complexity in complex processes and facilitate novel representations of problems. Consider the service branch problem as an example. The air force holds a perspective which identifies each and every airplane, while what is happening and contained on the ground (the army) may be encapsulated as a single ground force. The army would hold a perspective almost exactly opposite to the air force.

Anderson[1] presents and discusses a number of problems that illustrate the importance of a correct representation. These problems include the mutilated checkerboard, the 27 apples, and the two-string problem. Gardner[4] discusses a puzzle called fifteen and a word game that can both be abstracted to the game of tick-tack-toe with the help of a three-by-three magic square. These are simple examples of making a problem easier to solve through abstraction of the use of another perspective.

(Send correspondence to R.L.K)
R.L.K.: E-mail: rob@lanl.gov
K.R.B..: E-mail: kbisset@lanl.gov

When modeling and simulating complex physical and chemical processes, two problems in particular exist. One is the problem of representing material elements before, during, and after a change in state or phase. Some representations regard each change in state/phase as a new or different object. This is not only a poor representation of reality but a difficult representation to manage.

The other problem deals with recording the progress of a process, especially those modeled in computer simulation. A historic record of a simulation from start to finish can result in huge amounts of data. Much of the data may even be irrelevant or redundant when removed from the context of the whole. Dump data such as this is difficult if not impossible to query for individual pieces of information due to lack of structure.

The notion of perspective can be used to create more appropriate representations for these problems. These representations are more realistic and may contain more contextual detail. The following sections describe the object-based knowledge representation methodology (created and used by the authors) and its use. Examples are discussed to illustrate the use of perspective in the state/phase change problem and the process record problem.

2. BACKGROUND

2.1. Decomposition

In software design and development, decomposition refers to the breaking up of the whole into smaller parts. The goal being that smaller parts are more easy to understand and deal with, as opposed to the whole. Two types of decomposition are algorithmic and object-oriented.[2] Algorithmic decomposition is also known as top-down structural decomposition. This type of decomposition breaks the problem up into tasks which contribute to an overall process. In this case, the parts of the whole are algorithms which represent the tasks. In object-oriented decomposition the parts of the whole are objects as opposed to tasks. The objects represent tangible entities in the problem, ones being operated on by the tasks.

Whether one type of decomposition is better than the other depends on the problem and application. What is important to realize is that decomposition is not just for software design and development. It is also useful in knowledge representation. Decomposition occurs during domain analysis of the knowledge of interest and during acquisition or population of the knowledge into the representation. Since the knowledge representation used in this work is object-based, an object-based decomposition is also used.

2.2. Object-based knowledge representation

2.2.1. Introduction

A methodology is used to create and use a knowledge representation for the problems of state/phase change and process recording. This knowledge representation work is different from mainstream knowledge representation work where the use of ontologies continues. Ontologies are taxonomic or hierarchical classifications for knowledge. Although some ontologies are created for specific knowledge domains, many claim the ability to represent large portions of knowledge or all knowledge. As a result, ontologies can be large and difficult to understand and use.

What the methodology offers is a framework and guidelines for creating a knowledge representation specific to and for an application domain of knowledge. This means that the knowledge of interest is more closely coupled with the representation which helps make the population and maintenance of the representation easier. Representations are created with a set of primitive and meta-level constructs. The meta-level constructs are object-based and have defined relationships between them. This provides for the creation of structured, yet modular, representations.

The following sections describe in more detail some of the aspects of the object-based knowledge representation methodology including the meta-level constructs and their implementation. More in depth details of the methodology and knowledge representation in general can be found in Kelsey et al.[5,6] Some other applications of the methodology are described in Kelsey et al.[7,8]

2.2.2. Object-based primitives

The methodology contains a set of object-based primitives referred to as meta-level constructs. These are the basic building blocks for creating a representation. The most fundamental of these is the class. A class represents a collection of like objects. It can be considered a template for creating an instance or object. A class is defined and described by its attributes and methods (also meta-level constructs). An attribute is a characteristic or property of an object of the class. A method is an operation that can take place on an object of the class.

Also contained within a class is the perspective meta-level construct. A perspective is not a component for describing a class, but defines a point of view through which an object of the class views other objects in the knowledge domain. A perspective defines how those other objects are seen and perceived and can be used to further focus and limit the knowledge in a domain. In this manner perspective establishes a relationship between meta-level constructs and will be discussed as such in a later section.

A domain meta-level construct defines a group of objects that are instantiated from the available defined classes. A domain contains instances (objects) of the classes. An instance is a meta-level construct and represents an actual and unique object as defined by a class. Where a class represents a template, an instance represents the actual entity. A domain containing instances represents a body of knowledge for use in an application.

An event meta-level construct defines an actual use of the knowledge represented in the domain. This is not to be confused with a triggering mechanism or triggering entity. An event describes an occurrence between two objects. An event contains a before state, a method, an after state, and an agent. A before state is a list of attributes and their associated values (of the two objects) that change during the event being represented. Non-changing attributes are not listed and the associated values in the before state are values before the occurrence. A method is the method or operation causing the occurrence. The after state lists the same attributes in the before state but with their associated values after the occurrence. The agent identifies the object who called the method to instigate the occurrence.

Figure 1 shows a portion of the meta-level constructs and the structure they create. At the top of the structure is the kr meta-level construct. A kr contains classes and a domain. A class contains attributes, methods, and perspectives. A class can be instantiated to create an instance that is contained by a domain.

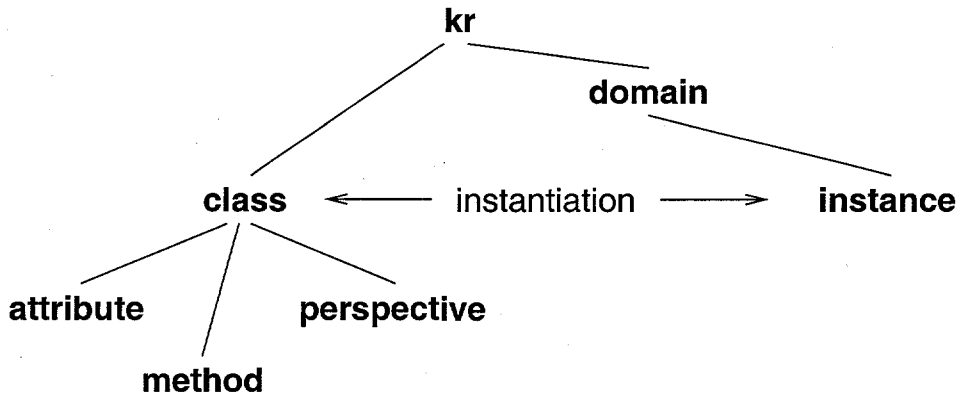

Figure 1. Some of the meta-level constructs and the structure they create.

2.2.3. Relationships

There are a number of supported relationships between the meta-level constructs. The relationships are inheritance, aggregation, perspective, and method-based. These relationships create structure between the entities of knowledge in a representation and in the overall representation. Perspective will be discussed in further detail in a later section. The inheritance relationship expresses a parent/child relationship. Using inheritance, a class can be defined to be a child of another (parent) class. The child class inherits all the attributes and methods of the parent class.

The aggregation relationship expresses a whole/part relationship. Using aggregation, a class can be defined that is made of components and each component is an instance of a class. This is accomplished by allowing an attribute of a class to contain an instance. In this manner, aggregate classes of other classes are created.

Method-based relationships are relationships specific to the knowledge being represented. These relationships occur between objects of classes that interact with each other through methods defined in each of the classes. Methods are the operations and behavior specific to objects of a class and this is the reason why these relationships are called method-based and specific to the knowledge being represented.

2.2.4. Perspective

The perspective relationship expresses a personal point of view of a knowledge domain. An object of a class possesses a view of the rest of the classes/objects in the domain. An object can have more than one defined perspective and change between them as necessary. A perspective can be general or limited and focused or anywhere in-between. It is only dependent on what classes are defined in the domain and the application of interest.

A perspective is created by identifying the classes pertinent to the perspective. That is, what classes in the domain are seen within this perspective? Additionally, the pertinent attributes and methods of each included class (in the perspective) must be identified. A perspective includes, as well as excludes, classes and their associated attributes and methods. As a result, a perspective can be thought of as a way of partitioning the knowledge domain.

A perspective can be further focused and limited through the inclusion/exclusion of instances (of classes) in the domain. Additionally, attributes of each instance can be included/excluded depending on the value associated with each attribute. This allows a perspective that creates an increasingly specific and narrow partition of the knowledge domain.

2.2.5. SGML Implementation

The meta-level constructs and their use is implemented in the Standard Generalized Markup Language[9] (SGML). This standard describes technology for facilitating text interchange in documents,[10] but the technology has many potential uses. An SGML document contains three components.[11] The first component is the SGML declaration which determines the formal syntax and any optional features to be used within documents. These are the rules for designing a document type definition (DTD) and for validating the conformance of a DTD and an associated document.

The second component is the document type definition (DTD) which is a rule set for a group of documents. The rules in a DTD describe the notation for making the content pieces of a document and how those pieces relate to one another. A DTD is like a grammar. The third component is an actual document containing content and markup.

Implementation of the meta-level constructs is in the form of a DTD. Each of the meta-level constructs and their associated parts and how each relates to one another is defined in the DTD. The DTD also defines the corresponding markup language to be used for each of the meta-level constructs. Examples of the DTD created for the object-based knowledge representation methodology are available in Kelsey et al.[5,6] An actual representation of knowledge corresponds to a document, the third component.

Once a knowledge representation has been created and populated with knowledge, it can be used. Use of the knowledge is obtained through the creation of post-parser routines. Typical utility uses are to translate and/or convert knowledge between different formats and systems. SGML is particularly well suited for this type of use.

There are a number of reasons for using SGML to implement a knowledge representation scheme. There is growing familiarity with markup languages and their use which means a user may be able to understand more quickly how to create and populate a knowledge representation of this type. Another benefit is that SGML is portable. Hypertext markup language (HTML), a language used for World Wide Web (WWW) documents, is an application of SGML and illustrates its portability. Perhaps the most important benefit is that the DTD for knowledge representation provides a formal definition of the structure and relationships of and between the meta-level constructs of the knowledge representation. Syntax and structure is defined and can be checked in a formal manner.

3. COMPLEX PROCESSES

3.1. Phase/state changes

3.1.1. Introduction

Modeling complex physical and chemical processes where changes in the state of matter occur can be difficult. The similar problem of systems with multiple phases of matter is also difficult. The difficulty of these problems lies in how to represent entities that change form and in so doing, end up unlike their original self.

Poor abstraction and decomposition in these types of situations can lead to representations that are difficult to understand and use. However, there is always a tradeoff. What is simple for a user to understand may be complicated for a machine to process. In the case of the phase/state change problem, a better model of reality seems appropriate. Representing the reality of the situation should be easier for a user to grasp and may lead to a more accurate model for computer simulation. The following sections describe and discuss a simple example with changing states of matter.

3.1.2. An example

Consider water as a simple example. Water can exist in three different states: ice, liquid water, and steam. These are the solid, liquid, and gas states of matter. In the solid state, water has a shape and volume. In the liquid state, water has a volume, but there is no definite shape. The shape in a liquid state depends on the container holding the water. In the gas state, water has neither a volume or a shape (these characteristics actually apply to all matter in these states, not just water[12]). A gas can be compressed to fit into a fixed and smaller volume.

Water changes state from solid to liquid to gas due to an increase in temperature, given a fixed pressure. Pressure actually affects the change from liquid to gas (boiling point) much more so than solid to liquid (melting point).[12] This is because liquids and solids are far less (if at all) compressible relative to gases.

Two different ways of representing water and its states without the notion of perspective are discussed. Both ways use object decomposition. One way uses the concept of inheritance and the other way uses the concept of aggregation. In the first way there is a class water which contains attributes that are common to all states of water, such as temperature. Additionally, there is a water-the-gas, water-the-liquid, and water-the-solid class, all which inherit from the class water. The attributes in each of these are specific to the state of water being represented. In the class water-the-liquid there are the attributes volume and density. In the class water-the-solid there are the attributes volume, density, and shape. Note that these representations and those that follow assume a fixed pressure and not all attributes are identified. There could be additional attributes depending on the application. Much is absent here in the interest of simplicity. Figure 2 shows a representation of water and its states using inheritance.

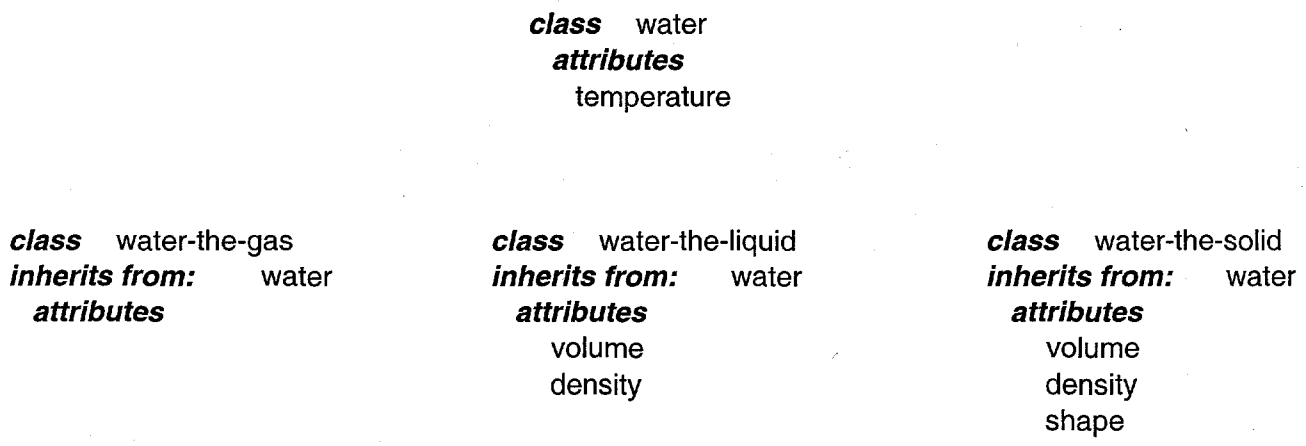

Figure 2. Representation of water and its states using inheritance.

The other representation uses aggregation and state-specific classes. This means there are classes water-the-gas, water-the-liquid, and water-the-solid, but they do not inherit from the class water. Instead, the class water contains an attribute state and an attribute state-specific (in addition to the attribute temperature). The attribute state identifies the current state and the attribute state-specific points to the state object (of the state class) as specified by the attribute state. In this manner, the state specific attributes are contained within the class water through aggregation. Figure 3 shows a representation of water and its states using aggregation.

Using perspective, the representation of water and its states becomes simplified. There is only one class which is the class water and it contains all the attributes that are necessary to water and its different states. By defining a perspective specific to each state, the class water can be viewed as if it represents the state of interest (and nothing more). Figure 4 shows the representation of water and its states using perspective and the SGML implemented meta-level constructs. It is important to note that the perspectives defined in Figure 4 are shown outside of a

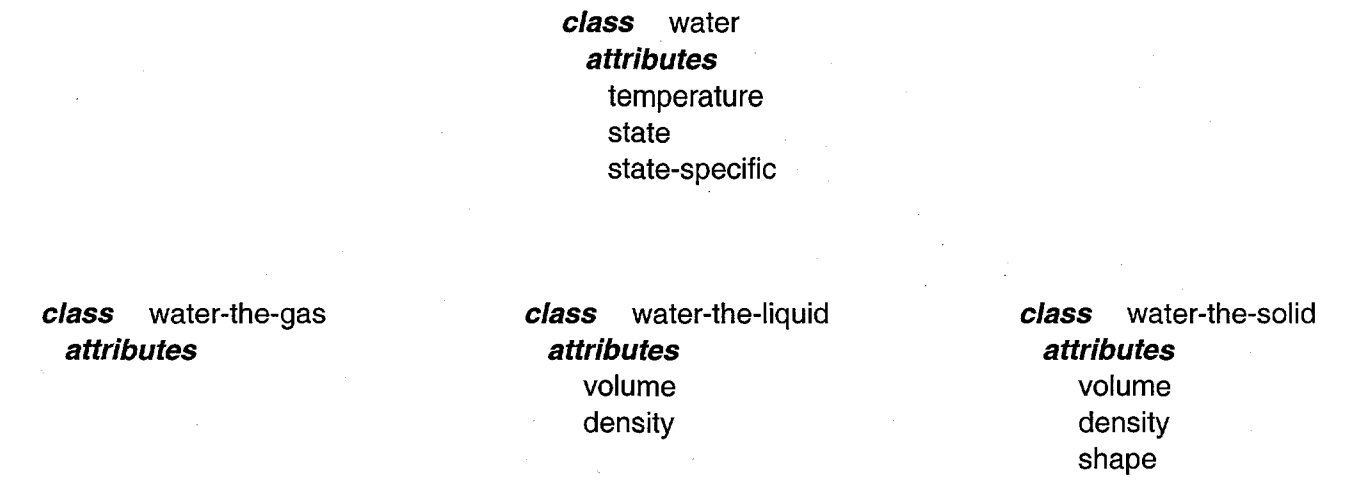

Figure 3. Representation of water and its states using aggregation.

class for brevity. Perspectives are actually contained in the class that possesses the perspective. In this case, these perspectives could be defined in some kind of class user that might be viewing and using the class water in terms of different states.

```
<class name="water">
  <attribute name="temperature"></attribute>
  <attribute name="density"></attribute>
  <attribute name="volume"></attribute>
  <attribute name="shape"></attribute>
</class>
```

```
<perspective name="gas-state">
  <def>gas state view of water</def>
  <iclass name="water">
    <atts>temperature</atts>
  </iclass>
</perspective>
```

```
<perspective name="liquid-state">
  <def>liquid state view of water</def>
  <iclass name="water">
    <atts>temperature density volume</atts>
  </iclass>
</perspective>
```

```
<perspective name="solid-state">
  <def>solid state view of matter</def>
  <iclass name="water">
    <atts>temperature density volume shape</atts>
  </iclass>
</perspective>
```

Figure 4. Representation of water and its states using perspective.

In Figure 4 the <class name= > tag is used to define the class water and the <attribute name= > tag is used to define the attributes of the class water. The <perspective name= > tag is used to define a perspective where the <def> tag shows the definition of this perspective. An <iclass> tag lists the classes to be included in this perspective and the <atts> tag lists the associated attributes to be included of the included class. Within the perspective gas-state is included the class water and its associated attribute temperature. Within the perspective liquid-state is included the class water and its associated attributes temperature, density, and volume. Within the perspective solid-state is included the class water and its associated attributes temperature, density, volume, and shape.

There is more maintenance necessary in supporting the representations that use either inheritance or aggregation. In the inheritance example, there is a class defined for each state. This means that as the state of water changes, an instance of these classes must be destroyed and an instance must be created. In the aggregation example, it is not clear whether instances of each of the state classes must be destroyed because only one is pointed to at any particular time. These issues do not apply to the representation using perspective. Each perspective consolidates the pertinent information for that state of water. There are no additional contrived classes necessary to model the different states.

3.2. Process record

3.2.1. Introduction

Process records are a record of every change that has taken place in a process, such as those modeled in a simulation. This discussion deals only with the form of the knowledge generated by the process and how it can be used. It does not deal with the mechanism for extracting that knowledge from the process.

A process can be thought of as a sequence of changes to the attributes of a collection of objects (e.g., the simulation entities of a simulation). Each change is recorded as an event. A process record enables a user to more easily analyze data, to restart a process at any point, and to branch processes. Combined with perspectives, a process record allows data to be presented with a narrow focus, containing only the pertinent information. This ability aids in the analysis of large amounts of complex data.

Normal simulation dumps provide a snapshot of the state of simulation at a particular point in time. A simulation can only be restarted from these points. In addition, important information about the changes that take place in between snapshots may be left out of the dumps. These drawbacks can be reduced or eliminated by increasing the frequency of the snapshots, with a corresponding increase in amount of data generated. Large amounts of data are hard to analyze without complex filters and the intelligent presentation of data. It is hard to extract the relevant pieces of information from this data, or to even define what "relevant" is for a particular user with a particular purpose.

Instead of simulation dumps, the use of events is presented. Events provide information not only about what happened (the change taking place), but also about what caused the change (agent instigating the change), and the mechanism of the change (method causing the change). A sequence of these events is a continuous record of the changes that take place during a process.

If a sequence of events records all of the changes that have taken place in a process, the state of a process can be recreated at any point. This allows a process to be re-started and the execution continued from any point. Additionally, events can be inserted into or removed from the process record and the contents of events in the process record can be changed. This allows the process to be branched at particular points. This can be used for "what if" scenarios and experimenting with different branches of the execution of a process. It also removes the need for rerunning the beginning parts of the simulation, leading to a decrease in execution time for successive simulations. During testing, it can also be used to explore branches of the simulation that are hard to reach through normal execution and reduce time needed to create data sets which exercise particular branches. Because the old and new values of changed attributes are given in each event, changes can be tracked in both the forward and backward direction, starting from any event in the process record.

3.2.2. An example

Consider a simple simulation of a bouncing ball. It is assumed that the collisions are completely elastic, and that there is no air resistance. There are three entities in this simulation: the bouncing ball, the floor, and gravity. Figure 5 shows the class definitions of each of these entities. The bouncing ball has two attributes: position and direction. Position may have the value air or ground indicating the top or bottom of the bounce, respectively. Direction (of motion) may either be up or down. In addition, the bouncing ball object contains two methods: push and pull. The floor object calls the push method of the ball at the bottom of the bounce and the gravity object calls the pull method of the ball at the top of the bounce.

```
<class name="bouncing-ball">
  <attribute name="position"></attribute>
  <attribute name="direction"></attribute>
  <method name="push"></method>
  <method name="pull"></method>
</class>
```

```
<class name="floor">
</class>
```

```
<class name="gravity">
</class>
```

Figure 5. Representation of the bouncing ball, the floor, and gravity.

If a simulation dump happens every two seconds and the ball hits the ground every second, then the ball appears not to move, since for each simulation dump the ball is in the same position. This is clearly incorrect. If the ball hits

the ground every minute and the simulation lasts for 29 seconds, then the ball appears not to move. This is correct since the simulation was not run long enough to see movement. For a simulation with many complex objects of differing time scales, it may be difficult to determine which of the above cases is present. Through the use of events, the proper case can be determined since all the information is contained in the process record. Figure 6 shows the events for one bounce of the ball.

```
<event name="event1">
  <befstate>
    ball.position = air
  </befstate>
  <method>
    gravity->pull
  </method>
  <aftstate>
    ball.position = ground
  </aftstate>
  <agent>
    gravity
  </agent>
</event>
```

```
<event name="event2">
  <befstate>
    ball.direction = down
  </befstate>
  <method>
    floor->push
  </method>
  <aftstate>
    ball.direction = up
  </aftstate>
  <agent>
    floor
  </agent>
</event>
```

```
<event name="event3">
  <befstate>
    ball.position = ground
  </befstate>
  <method>
    floor->push
  </method>
  <aftstate>
    ball.position = air
  </aftstate>
  <agent>
    floor
  </agent>
</event>
```

```
<event name="event4">
  <befstate>
    ball.direction = up
  </befstate>
  <method>
    gravity->pull
  </method>
  <aftstate>
    ball.direction = down
  </aftstate>
  <agent>
    gravity
  </agent>
</event>
```

```
<event name="event5">
  <befstate>
    ball.position = air
  </befstate>
  <method>
    gravity->pull
  </method>
  <aftstate>
    ball.position = ground
  </aftstate>
  <agent>
    gravity
  </agent>
</event>
```

Figure 6. *Five events recording one bounce of the ball.*

Perspectives can be combined with events to present meaningful views of the data and to remove irrelevant data, cutting down on information overload. A perspective would only present events which contain attributes, methods, or classes of interest. For example, a perspective that includes the class ball would see all five events. A perspective that only includes the attribute position of the class ball would see **event1**, **event3**, and **event5**. A perspective that only includes the class gravity would see **event1**, **event4**, and **event5**.

4. CONCLUSIONS

The problem of changing states of matter demonstrates some intriguing issues for knowledge representation. In the simple example discussed, water and its states were represented using multiple classes and by an opposing manner using multiple perspectives. Whether creating a class for each state is correct or not depends on one's definition of state, but a physical object water is always water, irregardless of which state it exists. A representation using multiple perspectives (one for each state) recognizes that water is always water, but consolidates the pertinent details of each state into a separate partition. In this manner, water takes the form of the current state as it would in reality. The perspective masks the details that belong to another state. A perspective creates not just a view, but a partitioned access and partitioned use of a domain of knowledge.

The event meta-level construct is used to create a continuous record of a process. Each change within a process is recorded with an event along with why the changed occurred (method causing the change) and the reason it occurred (what agent instigated the change). A typical simulation dump outputs the values of all its variables at a specified time interval. A number of these dumps still does not contain the continuous record possible with a sequence of events because some information may be lost in between dump intervals. Also, a dump will contain a potentially large amount of irrelevant information, such as variable values that have not changed. With the use of perspectives, events can be consolidated into more focused categorizations which could make them more useful for analysis of process records. There is also the potential for more flexible re-starts of a process including the ability to experiment by inserting user-defined events into a process sequence. Although the overhead of recording every change in a process can be extreme, the value gained for analysis and experimentation may outweigh the cost.

REFERENCES

1. J. R. Anderson, *Cognitive Psychology and Its Implications*, W.H. Freeman and Company, New York, NY, fourth ed., 1995.
2. G. Booch, *Object-Oriented Analysis and Design with Applications*, Addison-Wesley Publishing Company, Menlo Park, CA, second ed., 1994.
3. P. Norvig, *Paradigms of Artificial Intelligence Programming: Case Studies in Common LISP*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

4. M. Gardner, *Aha!*, Scientific American, Inc. / W.H. Freeman and Company, New York, NY, 1978.
5. R. L. Kelsey, R. T. Hartley, and R. B. Webster, "An object-based methodology for knowledge representation in SGML," in *Proceedings of the Ninth IEEE International Conference on Tools With Artificial Intelligence*, pp. 304–311, IEEE Computer Society, (Los Alamitos, CA), 1997.
6. R. L. Kelsey, *Object-Based Knowledge Representation Implemented in SGML.* PhD thesis, New Mexico State University, Las Cruces, NM, July 1998.
7. R. L. Kelsey, R. B. Webster, and R. T. Hartley, "Using multiple perspectives to suppress information and complexity," in *Digitization of the Battlespace III*, vol. 3393, pp. 72–85, Society of Photo-Optical Instrumentation Engineers, Society of Photo-Optical Instrumentation Engineers, (Bellingham, WA), 1998.
8. R. L. Kelsey and R. B. Webster, "Adapting perspectives to facilitate knowledge assimilation," in *Applications and Science of Computational Intelligence II*, vol. 3722, Society of Photo-Optical Instrumentation Engineers, Society of Photo-Optical Instrumentation Engineers, (Bellingham, WA), 1999.
9. International Organization for Standardization, Geneva, *ISO 8879:1986 Information processing - Text and office systems - Standard Generalized Markup Language (SGML)*, October 1986.
10. E. van Herwijnen, *Practical SGML*, Kluwer Academic Publishers, Boston, MA, second ed., 1994.
11. L. Alschuler, *ABCD...SGML A User's Guide To Structured Information*, International Thomson Computer Press, Boston, MA, 1995.
12. D. D. Ebbing, *General Chemistry*, Houghton Mifflin Company, Boston, MA, 1984.