An Object-Based Methodology For Knowledge Representation

Robert L. Kelsey

Los Alamos National Laboratory

New Mexico State University

rob@lanl.gov

Roger T. Hartley

New Mexico State University

rth@cs.nmsu.edu

Robert B. Webster

Los Alamos National Laboratory

robw@lanl.gov

Abstract

An object-based methodology for knowledge representation is presented. The constructs and notation to the methodology are described and illustrated with examples. The "blocks world," a classic artificial intelligence problem, is used to illustrate some of the features of the methodology including perspectives and events. Representing knowledge with perspectives can enrich the detail of the knowledge and facilitate potential lines of reasoning. Events allow example uses of the knowledge to be represented along with the contained knowledge. Other features include the extensibility and maintainability of knowledge represented in the methodology.

DISCLAIMER

1 Introduction

An object-based methodology for knowledge representation attempts to represent and describe knowledge in terms of objects. The object is a conceptual building block for constructing the representation for the knowledge of interest. This methodology is not unlike object-oriented analysis (OOA) and design (OOD) [1] [4]. They all share many of the same concepts and use similar terminology. They all decompose knowledge into manageable units called objects.

An object paradigm for a knowledge representation methodology is appealing because it is natural to define and decompose things in terms of objects. This is especially true of declarative knowledge which is descriptive knowledge comprising "knowing that" rather than "knowing how" [3]. Another reason this methodology is appealing is because objects are modular and this increases the potential for dynamic and maintainable representations.

This methodology is an analysis and design methodology. It is meant to be used to design domains of knowledge, perhaps to be put into existing knowledge representation systems. Systems are difficult to use because they often have no methodology for modeling the knowledge.

Other features of this methodology include the use of perspectives and events. Representation of knowledge using multiple perspectives is an important part of the methodology. Perspectives allow for multiple views of the existing knowledge and put the knowledge in additional contexts. This may increase the detail of the knowledge. Events allow the actual use of knowledge to be represented along with the knowledge itself. An event describes an interaction or operation between two objects by showing the objects before and after an interaction has occurred.

The following sections discuss the constructs of the methodology and illustrate the constructs and notation with examples such as the blocks world. The blocks world is a good example because it makes use of all of the methodology's constructs. The increased use of object-oriented notions has made some of the object-oriented terminology confusing and ambiguous. The terminology used in this methodology is not always defined in the same way as some common object-oriented terms. Thus, terminology is defined as it is used and attempts have been made to make the terminology precise.

2 Constructs of the Methodology

2.1 Class

The basic construct in the methodology is the class. A class is a template or prototype that describes an object or a collection of objects. A member of a class is called an object or instance of a class [1] [4]. The act of creating an object is called instantiation (from a class). There are two types of classes in the methodology. They are agent classes and non-agent classes. An agent

class typically describes an object that acts on other objects or initiates actions. A non-agent class describes an object which is essentially inanimate and does not usually act on other objects.

An example of an agent class is the class of soccer players. An example of a non-agent class is the class of soccer balls. A soccer player kicking a soccer ball is an example of an agent object initiating an action (kicking) on a non-agent object. Classification into agents or non-agents is not a requirement of the methodology, but it is helpful for determining behavior in classes of objects and interactions between classes of objects. Schank [8] and others have made this distinction between agent and non-agent, calling agent objects actors, and found the distinction to be useful.

2.2 Attributes, Methods, and Perspectives

There are three parts to a class which are necessary to adequately describe the objects which belong to a class. The three parts are attributes, methods, and perspectives. Attributes are the characteristics and properties that describe a class of objects. The attributes of an object can help determine whether the object is of one class or another class. Upon instantiation the object is assigned a unique name identifying it and its attributes are assigned values.

Consider a class of chairs. The class chair may have attributes such as number of legs, style (early American, art deco, modern, et cetera), and type (office, dining room, living room, outdoor, et cetera). A particular chair identified as Sam's chair has four legs and an early American wood finish. Sam's chair is for Sam's dining room. Consider another class, a class of desks. The class desk may have attributes such as number of legs, number of drawers, style (early American, art deco, modern, et cetera), and type (office, computer, child, et cetera). Sam's desk has four legs, three drawers, is of modern style, and built especially for computer use. Although chairs are not usually mistaken for desks (nor desks for chairs), the attributes and associated attribute values make the difference in objects specific. Figure 1 shows an example of two class definitions, chair and desk. The name of a class follows the keyword *class*. The names of attributes of a class follow the keyword *attributes*. If an attribute is limited to specific values, those values are shown in square brackets following the attribute name.

class chair
attributes
number_of_legs
style [early American, art deco, modern]
type [office, dining room, living room, outdoor]

class desk
attributes
number_of_legs
number_of_drawers
style [early American, art deco, modern]
type [office, computer, child]

Figure 1: Class definitions for chair and desk.

Methods determine the behavior of objects of classes. Methods are the operations that are allowed on an object of a class. Objects interact with each other through methods. By limiting the methods available in a class, interaction between objects can be constrained. Methods are defined in a class,

but are not available for use until an object of the class has been instantiated. An example method is the operation kick, in the class soccer balls. Suppose an object named Sam's soccer ball is instantiated from the class soccer balls. Further suppose that Sam is an agent object instantiated from the class soccer players. The method or operation kick is how the agent object Sam interacts with the object Sam's soccer ball. Kick is an allowed operation on the object Sam's soccer ball. Figure 2 shows the class definitions for soccer ball and soccer player. Notice that soccer ball has no attributes and one method. The name of a method follows the keyword *method*.

```
class  soccer_ball
 methods
  kick
```

```
class  soccer_player
 attributes
  position
  expertise_level
  jersey_number
```

Figure 2: Class definitions for soccer ball and soccer player.

Perspectives are a means of expressing points of view. A perspective defines how or in what way an object of a class may be viewed. It may be thought of as one user's view or opinion of a domain of knowledge. Some attributes and methods may be important to one perspective while not important to another perspective. Thus, a particular perspective is defined by grouping the attributes and methods in a class that are pertinent to the perspective. Similar to a method, a perspective on an object is defined in that object's class and limits or constrains the available views by other objects. In this way, a perspective can be used to further focus and limit the knowledge in a domain.

As an example, consider an object hunk of quartz crystal of the class crystals. There are several attributes including weight, type, color, and crystal size. Among the perspectives available are the geologist's point of view and rocks as a paperweight point of view. The geologist perspective may be interested in all of the attributes while the paperweight perspective is probably only interested in the weight attribute. Figure 3 shows the class definition for crystal and which attributes are grouped to which perspectives. With each attribute listed in the class definition is the keyword *perspective* followed by the perspective name or names (separated by commas) to which the attribute belongs. Methods can be grouped in perspectives as well and the notation is similar.

```
class  crystals
 attributes
  weight        perspective  paperweight, geologist
  type          perspective  geologist
  color         perspective  geologist
  crystal_size  perspective  geologist
```

Figure 3: Class definition for crystal showing groupings of perspectives.

Like methods, perspectives on an object may be defined in both agent and non-agent classes. An agent object acts on other objects (agent and non-agent) by invoking the object's methods. The

situation is similar for perspectives. An agent object can have a perspective or point of view about other objects (agent and non-agent). Thus, while the allowable perspective is defined in one object of a class, the use or observation of the perspective takes place in an agent object.

2.3 Creating a Domain

Up to this point discussion has focused on the defining of classes and the parts necessary to define classes. There has been a little discussion about the difference between classes and objects, but not about how objects are created or instantiated. Instantiation leads to defining a knowledge domain. The defining of classes designates what is available to a domain. A class can be thought of as an available resource. To create a domain of knowledge, the class pertinent to that domain must be instantiated for use. This means that classes of interest are identified and instantiation causing objects (of those classes) to be made available for use in the domain. Figure 4 shows a domain definition named soccer practice which includes objects Sam's soccer ball and Sam (a soccer player).

domain soccer_practice
soccer_ball Sam's_soccer_ball
soccer_player Sam

Figure 4: Domain definition for soccer practice.

2.4 Using the Knowledge

A domain of objects can represent knowledge and how the knowledge may be used, but it does not necessarily represent use or show use. The representation of knowledge alone can be valuable, but adding examples of actual use to the representation will increase the value. Another construct is needed to represent examples of actual use in the methodology. This construct is the event.

An event has a unique name and is essentially composed of four parts. The four parts are the *before state*, *method*, *after state* or *result state*, and *agent*. A state is defined with the values of the attributes of an object at a particular moment in time. More than one object may participate in an event, thus a state can contain the attribute values of multiple objects. A method is the single operation that causes the change in state from before to after. An agent is the agent-object that called or instigated the method. To keep things simple, only the objects with attribute values that change from before state to after state are listed within the before and after states.

A single event represents a single operation use or example of the knowledge contained in the domain. A sequence of events may be utilized to represent plans for reaching some goal or the steps involved for some kind of instruction. A sequence of events may be represented in a list in chronological order. Figure 5 shows the declaration of a single event. The event name follows the keyword *event*. Following the keywords *before state* and *after state* are the object name, attribute

name, and attribute value for those that change during state change. Following the keyword *method* is the object name and method name that caused the change in state. Following the keyword *agent* is the name of the instigating agent object.

```
event  event_one
 before state
  computer.power_switch = off
 method
  computer->turn_on
 after state
  computer.power_switch = on
 agent
  computer_operator
```

Figure 5: Event declaration for event one.

2.5 Inheritance and Aggregation

The concepts of inheritance and aggregation can be useful when creating classes and adding to current classes. Inheritance is the generalization/specialization abstraction mechanism [2]. Generalization takes one or more classes and generalizes them into one class. The is-a relation is often used to express generalization (and inheritance). For example, the class of modes of transportation is a generalization of the classes, trains, automobiles, and planes. The reverse relationship to generalization is specialization.

Aggregation/decomposition or aggregation/disaggregation is another abstraction mechanism [2]. Aggregation is the idea of the sum of the parts being the whole. The part-of relation is often used to express the aggregation abstraction. For example, rooms, walls, and doors are parts of a house. The reverse relationship to aggregation is decomposition or disaggregation.

The methodology supports the ability to do inheritance and aggregation/disaggregation. For inheritance, when a new class is defined, it may be defined as a sub-class of some existing class. All the attributes, methods, and perspectives of the existing class are inherited by the sub-class. Figure 6 shows the definition of a class transportation vehicle and the definition of a class (sub-class) car inheriting from transportation vehicle. The keyword *of* is used to show inheritance.

For aggregation/disaggregation, the attributes in a class may have names which are instances of another class. This means the parts of a whole are described by the attributes in a class. The class is the whole while the attributes are the parts. Since an attribute can be an instance of a class, each part can have parts as well. Figure 7 shows the class definitions for house, wall, door, and room. The classes wall, door, and room have instances in the attributes of the class house.

class transportation_vehicle
attributes
passenger_quantity
cost

class car *of* transportation_vehicle
attributes
transmission_type [standard, automatic]
door_quantity
miles_per_gallon

Figure 6: Example of inheritance.

class house
attributes
door front_door
wall front_wall
room den

class wall
attributes
area
material

class door
attributes
material

class room
attributes
area

Figure 7: Example of aggregation/disaggregation.

3 Classic AI Example

One of the classic problems in artificial intelligence (AI) is the blocks world [6] [7] [9]. The blocks world comes from a set of problems with limited and focused domains of knowledge known as microworlds [7]. These problems appear to need intelligence to solve them. The blocks world problem involves a number of blocks on a table and a robot arm that can pick up a single block at a time. Tasks in the blocks world involve the robot arm manipulating the blocks into a particular arrangement such as a stack of blocks. The blocks world has often been used to discuss planning and goal-oriented AI approaches.

The blocks world problem has many different versions but they all do basically the same thing. Sometimes the constraints differ like the number of blocks that may be stacked atop one another. Knowledge about the blocks world is typically represented using predicate calculus. The blocks are represented by constants. Conditions or state information about the blocks are represented as predicates for example, on and clear. Then there are actions, for example move, which are also represented by predicates. Although it is a simple knowledge domain, the blocks world contains enough parts and pieces to exercise this object-based methodology. In the following sections the representation of the blocks world is described using the object-based methodology.

3.1 Class Definitions

There are three classes which must be defined for the blocks world. They are the class table, class block, and class robot arm. The class table is simple because it contains no attributes or methods. While it is true a table can have many attributes similar to those of the desk and chair previously described, in the blocks world the table is merely a physical object where the blocks may be located.

Any knowledge beyond this is outside the scope of the problem. Figure 8 shows the definition of the class table with the perspective named stacking for the blocks world problem.

```
class  table                    class  block
  perspective  stacking           attributes
                                    ID                          perspective  stacking
                                    location [table, an ID]     perspective  stacking
                                    covered [NULL, an ID]       perspective  stacking

                                  methods
                                    pickup                      perspective  stacking
                                    putdown (locale)            perspective  stacking
```

Figure 8: Definition of the classes table and block.

The class block is more interesting than the class table. The class block has three attributes which are an identification (ID), a location, and a covered (or not) indicator. Although each instance of a class block has a unique name, the ID attribute is available for the methods. The value of a location attribute may be the table or the ID of another block (to indicate stacking of a block) or the robot arm. The value of the covered attribute may be null or the ID of another block. Null indicates there is nothing atop (covering) this block and an ID indicates which block is atop this block. All the attributes of class block belong to the perspective named general. Figure 8 shows the definition of the class block.

The methods of the class block are the operations that may be performed on an instance of the class. There are two methods, pickup and putdown. The pickup method allows this block (a particular instance) to be picked up by an agent object (in this case the robot arm). The putdown method allows the block to be put down at some location, hence an argument in the method is necessary and may have the value table or an ID. Both these methods belong to the perspective named general. The actual content of a method, that is how a method does what it does, can be represented in a number of ways. It may be represented with rules, pseudocode, or even a program. The representation of the method content depends on the end application of the knowledge. Figure 9 shows a pseudocode representation of the content of the methods pickup and putdown.

The class robot arm is the only agent class in the blocks world. Table and block are non-agent classes. The class robot arm has only one attribute which is named holding. The holding attribute indicates whether the robot arm is holding a block or not, and if so, the ID of which block it is holding. There are no methods for the class robot arm. Since robot arm is an agent, it has a perspective of the rest of the domain. Figure 10 shows the definition of the class robot arm. The name of the perspective follows the keyword *perspective*. In this case the perspective name is stacking since the typical blocks world problem is about stacking the blocks. A brief definition describes the perspective and follows the keyword *definition*. After the keyword *classes* follows a list of the classes pertinent to this perspective. Objects of these classes are within the perspective (or view) of the robot arm agent object instantiated from the class robot arm. The class definitions (of the listed classes) can then be inspected to see which attributes and methods are grouped to

```
method   pickup
  IF robot_arm.holding is NULL THEN
    IF covered is NULL THEN
      location := robot_arm
      robot_arm.holding := ID
    ELSE
      error "block covered"
  ELSE
    error "arm is holding"
```

```
method   putdown
  IF robot_arm.holding !NULL THEN
    IF locale = an ID THEN
      IF locale.location = an ID THEN
        error "stack too high"
      ELSE
        location := locale
        robot_arm.holding := NULL
    ELSE
      location := locale
      robot_arm.holding := NULL
  ELSE
    error "not holding block"
```

Figure 9: Examples of method contents.

the perspective named stacking (see Figure 8 for the class block definition).

```
class  robot_arm
 attributes
   holding [NULL, an ID]

 perspective   stacking
  definition
     the typical view of the blocks world
  classes
   table
   block
```

Figure 10: Definition of the class robot arm.

It can be argued that the methods pickup and putdown belong to the robot arm and not the block. It makes sense that the robot arm actually exercises the actions of picking up and putting down blocks. So, why does the class block have these methods? After trying both ways (with more than the blocks world example) a decision was made to think of methods as operations on an object and therefore contain them within the operated-on object. Also, this seems to lead to a cleaner way of constraining allowable interactions between objects. The truth is methods can be represented in either place, but it is important to pick and use one or the other and not both.

3.2 Domain Definition and Events

Now that the available resources are defined, a particular domain for the blocks world can be created. In this domain there is the table, robot arm, and four blocks. Figure 11 shows the definition of the domain blocks world one. The table object is named table1. The robot arm object is named robot arm1 and the keyword *with* indicates the perspective that this particular instance of robot arm is

using. In this case the perspective is stacking. The four block objects are named blockA, blockB, blockC, and blockD.

```
domain blocks_world_one
 table  table1
 robot_arm  robot_arm1  with  stacking
 block  blockA
 block  blockB
 block  blockC
 block  blockD
```

Figure 11: Definition of the domain blocks world one.

Some example events for the blocks world are shown in Figure 12. The first event, event one, illustrates that blockA is picked up by the robot arm. The second event, event two, illustrates that blockA is putdown on top of blockB by the robot arm. What is interesting about the event construct is that it can be used to draw inferences. Given two of either the *before state*, *method*, or *after state*, the third may be inferred with the help of class definitions and method contents. For example, suppose the method pickup had not been identified in event one of Figure 12. By looking at the change in values of blockA's location and the robot arm's holding indicator and the method contents, it can be inferred that the pickup operation was executed. It may also be possible to make inferences by including the agent, that is, given any two of the four event parts, infer the other two.

```
event  event_one
 before state
  blockA.location = table1
  robot_arm1.holding = NULL
 method
  blockA->pickup
 after state
  blockA.location = robot_arm1
  robot_arm1.holding = blockA
 agent
  robot_arm1
```

```
event event_two
 before state
  blockA.location = robot_arm1
  robot_arm1.holding = blockA
  blockB.covered = NULL
 method
  blockA->putdown(blockB)
 after state
  blockA.location = blockB
  robot_arm1.holding = NULL
  blockB.covered = blockA
 agent
  robot_arm1
```

Figure 12: Two events taking place in blocks world one.

3.3 Another Perspective

Suppose that the blocks world is viewed from another perspective. Suppose that it is not an exercise in stacking blocks but an exercise in putting blocks next to one another, as if they are alphabet blocks for teaching a child to spell. This perspective requires a different group of attributes and methods for the class block. Figure 13 shows the class block with the newly added perspective

named spelling. Note that knowledge to actually spell is not represented. The attributes grouped in the perspective spelling are ID, rightside, and leftside. Rightside indicates which block (if any) is on the right and leftside indicates which block (if any) is on the left, hence the possible values of null or an ID.

```
class  block
 attributes
  ID                                  perspective  stacking, spelling
  location [table, an ID]             perspective  stacking
  covered [NULL, an ID]               perspective  stacking
  rightside [NULL, an ID]             perspective  spelling
  leftside [NULL, an ID]              perspective  spelling
 methods
  pickup                              perspective  stacking, spelling
  putdown (locale)                    perspective  stacking
  putbeside (NULL | an ID, side)      perspective  spelling
```

Figure 13: The class block with the two perspectives stacking and spelling.

The methods grouped in the perspective spelling are pickup and putbeside. The method putbeside can have an argument of null or two arguments identifying a block and a block's side. A null argument essentially means the block is put down on the table away from other blocks. The two argument pair, a block ID and the left or right side, identifies which block on the table to put the block in the robot arm beside. Note that a method content for putbeside is not shown. The attribute ID and method pickup are shared with the perspective named stacking and are therefore the same.

With the new perspective the class robot arm also changes. The perspective named spelling must be added to the class robot arm by listing a definition and the classes associated with the perspective. Figure 14 shows the class robot arm with the added perspective. Adding this new perspective to the current knowledge also illustrates how simple it is to extend the knowledge domain using this methodology. A domain definition will be the same as shown in Figure 11 except that the instance of robot arm will with the perspective named spelling.

```
class  robot_arm
 attributes
  holding [NULL, an ID]

 perspective  stacking                    perspective  spelling
  definition                               definition
   the typical view of the blocks world     setting blocks side by side for spelling
  classes                                  classes
   table                                    table
   block                                    block
```

Figure 14: The class robot arm with the two perspectives stacking and spelling.

The representation of multiple perspectives may allow for additional reasoning over the domain. Different perspectives on the same object and the same perspective on different objects may initiate some useful comparisons and contrasts. Gentner [5] describes domain comparisons including literal similarity, analogy, and abstraction which are derived from the mapping of objects, object attributes, and relations between objects. Consider the two different perspectives presented for the blocks world. These are two different views of the same set of objects. Are they really so different? The general perspective sees stacks of blocks and the spelling perspective sees lines of blocks side by side. If a stack of blocks is laid down on the table then it will look like a line of blocks side by side. This is an important observation and a type of observation or inference that may be facilitated by perspectives.

4 Conclusions

This object-based methodology is a structured, yet flexible means of representing knowledge. Only the knowledge that is pertinent and necessary to a problem or application area need be represented. Should the needs change, the knowledge is easily extended by changing and/or extending the classes and/or adding new classes to the domain. The notion of containing knowledge in classes allows knowledge to be shared and re-used. Class libraries of common knowledge may be created and distributed. The use of perspectives also adds to the extensibility and detail of the knowledge. Perspective may also facilitate new lines of reasoning over the domain of knowledge. The use of events allows for representing knowledge use along side of knowledge containment and facilitates another line of reasoning.

Initial work shows a large potential for this methodology. More research must be done to further understand and implement the potential reasoning mechanisms in this methodology. Other reasoning based on inheritance and aggregation/disaggregation has not been mentioned, but holds potential as well. This methodology needs to be more rigorously exercised to identify the types of knowledge it may represent and the limits or bounds on that knowledge. Also, comparisons to other knowledge representation schemes and systems must be made.

References

[1] Grady Booch. *Object-Oriented Analysis and Design with Applications.* Addison-Wesley Publishing Company, Menlo Park, CA, second edition, 1994.

[2] Alexander Borgida, John Mylopoulos, and Harry K. T. Wong. *On Conceptual Modelling*, chapter 4, Generalization/Specialization as a Basis for Software Specification, pages 84–114. Springer-Verlag, New York, NY, 1984.

[3] Ronald J. Brachman and Hector J. Levesque. *Readings In Knowledge Representation.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1985.

[4] Peter Coad and Edward Yourdon. *Object-Oriented Design.* Yourdon Press, Englewood Cliffs, NJ, 1991.

[5] Dedre Gentner. *Readings in Cognitive Science A Perspective from Psychology and Artificial Intelligence*, chapter 3.2, Structure-Mapping: A Theoretical Framework for Analogy, pages 303–310. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[6] Elaine Rich and Kevin Knight. *Artificial Intelligence.* McGraw-Hill, Inc., New York, NY, second edition, 1991.

[7] Stuart J. Russell and Peter Norvig. *Artificial Intelligence A Modern Approach.* Prentice Hall, Englewood Cliffs, NJ, 1995.

[8] Roger C. Schank and Robert P. Abelson. *Scripts Plans Goals and Understanding An Inquiry Into Human Knowledge Structures.* Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ, 1977.

[9] Patrick Henry Winston. *Artificial Intelligence.* Addison-Wesley Publishing Company, Reading, MA, second edition, 1984.

Report Number (14) LA-UR--97-1598
CONF-971161

Publ. Date (11) 199707
Sponsor Code (18) DOE/DP, XF
UC Category (19) UC-705, DOE/ER

DOE