MACHINE MODELING
IN
LOOM

by

David Wilczynski
Thomas A. Lipkis
Pacific Software Solutions
1911A Ernest Ave.
Redondo Beach, CA 90278
(310)372-0333

October 20, 1993

I. INTRODUCTION

A. Overview

This project's goal is to build a model of industrial CNC-type machines using LOOM, a knowledge base technology. This preliminary report will show the beginning of the model and try to motivate its creation and the particular choice of LOOM.

There are numerous modeling languages, each with special characteristics. We are using a knowledge representation tool called LOOM. Our model is a network of concepts and relations, connected to one another with links that show a concept's definition and its subsumption (IS-A) relationship to other concepts.

This kind of model is controversial; there is a high start-up cost, while the design of the concepts and relations is as much art as science. Some semantic models are good, some are bad. Many people think that such models are just fancy forms of data-bases. We will try to address all these issues.

B. Loom

The following description of Loom paraphrases the overview in the Loom User Guide Version 1.4. Loom is a high-level programming language and environment intended for use in constructing expert systems and other intelligent application programs. The LOOM language targets a programming methodology that places heavy emphasis on the specification of explicit, declarative domain models. Besides this language LOOM provides: (1) powerful deductive support, including both strict and default reasoning, and automatic consistency checking; (2) multiple programming paradigms that interface smoothly with a declarative model specification; (3) knowledge-base facilities including a full first-order query language; multiple knowledge bases, and dumping and loading of knowledge-base objects (persistent objects).

The knowledge representative framework in LOOM is derived from the language KL-ONE. As a descendent of KL-ONE, LOOM features efficient automatic classification to compute subsumption relationships which define inheritance. LOOM's description language, classifier, and rule language provide inference capabilities not found in current generations of knowledge representation tools. Large knowledge bases are possible because the classifier enforces strict adherence to the taxonomy being created.

LOOM's modeling language is a hybrid consisting of two sublanguages, a definition language and an assertion language. The definition language expresses knowledge about concepts and relations. The assertion language is used to specify constraints on concepts and relations, to assert facts about individuals (instances). So, if the assertions about an instance *I* collectively satisfy the definition of a concept *C*, then *I* is recognized as an instance of *C*. And thus, anything you "know" about *C* is true about *I*. And anything you "can do" to instances of *C,* perhaps in the form of rules*,* you can do to *I*.

Because of the clear separation of rules from the knowledge base of concepts, large and maintainable rule bases can be created. The deep, abstract concept taxonomy makes it possible for the rule-builder to express the rule conditions clearly, with as much precision as required.

C. Automation Intelligence

Automation Intelligence is a company with many products in the CAD/CAM area. Pacific Software is in the process of reworking their post-processing configuration tool. In that tool, information is gathered about the target CNC machine. Information, such as machine type, axis locations, machine travel, and hundreds of other details are provided by the user.

Like most programs, all of the constraints, limits, defaults and relationships are buried within the code and associated text files. For many reasons, the program now needs modernization. As an adjunct to rewriting it, we will also model the machines with which the program deals. In LOOM terms, we are building the definitional taxonomy of machine tool terms that the configuration tool can use.

The hope is to build a knowledge base of machine descriptions that can be used by all of the CAD/CAM tools required by this domain. Perhaps then when two tools each talk about a "tilting-head" spindle, they will mean the same thing.

II. THE MODEL

A. Motivation

The model as shown in the appendix expresses concepts and relations about CNC machines. The driving motivation is to model the milling machines and lathes that Automation Intelligence's configuration tool handles.

In constructing a LOOM model, we try to express "truths" about the domain being modeled. That is, the concepts being defined should make sense to anyone in the field. Yet, in practice having an application in mind seems to make the modeling easier and more focused.

The configuration tool tries to glean from the user hundreds of parameters about the target machine. Some are required for both lathes and milling machines, some depend on the particular type of milling machine (as we will show in the next section), some depend on the value of parameters the user has just specified. However, in the configuration tool, the relationships between all these parameters is opaque. How setting one parameter affects the others is hardly known by anyone--only the code knows and it's not very forthcoming.

B. The Definitions - Concepts and Relations

Consider the following concept definitions:

```
(defconcept Machine)

(defconcept Material-Removing-Machine :is-primitive Machine)

(defconcept Live-Tooling-Machine :is
  (and Material-Removing-Machine
       (at-least 1 has-cutting-spindle)
       (exactly 1 has-primary-cutting-spindle)
))
```

First, **Machine** is defined. It is given no roles--roles are also called relations. Then, a **Material-Removing-Machine** is defined as a kind of **Machine**. Again, no roles have been specified. At this point in the modeling exercise the categories for the taxonomy are being fleshed out.

Next, we define a live tooling machine, that is, a machine that removes material by spinning a cutting tool. Thus, a **Live-Tooling-Machine** is defined to be a **Material-Removing-Machine** that has at least one cutting spindle and exactly 1 primary cutting spindle. Notice that these roles can overlap; the primary cutting spindle will be one of the cutting spindles.

The roles, **has-cutting-spindle** and **has-primary-cutting-spindle**, are relations of **Live-Tooling-Machine** that differentiate it from a **Material-Removing-Machine**. Later in this project lathes will be shown to be **Material-Removing-Machine** that spin their workpieces and apply nonspinning tools to them.

The **:is-primitive** verb tells LOOM that though a **Material-Removing-Machine** is a kind of **Machine** with the specified characteristics, there is some additional property that all **Material-Removing-Machine**'s have (but that **Machines** in general don't) that has not been modeled. Thus LOOM will never infer that any instance of **Machine** is a **Material-Removing-Machine** unless it is explicitly stated. The use of **:is** in the definition of **Live-Tooling-Machine** means that **Live-Tooling-Machine** is fully defined by its stated properties. Thus any instance of **Material-Removing-Machine** that has at least one cutting spindle and exactly one primary cutting spindle will be recognized by LOOM as a **Live-Tooling-Machine**.

The specifier, **:is-primitive**, is used in those cases where a concept is undefinable in terms of its features (such as natural kinds), or we choose not to fully model it because it would be too difficult or is not needed. This is the case for **Live-Tooling-Machine**.

The definition of a relation is just as important as the definition of a concept. Consider:

```
(defrelation has-attachment :is-primitive :range Thing-With-3D-Offset)

(defrelation has-spindle :is
  (and has-attachment
       (:range Spindle-Assembly)))

(defrelation has-cutting-spindle :is
  (and has-spindle
       (:range Cutting-Spindle-Assembly)))
```

Thus, **has-attachment** is that relation whose range is **Thing-With-3D-Offset**, i.e., objects that have a position. Then, **has-spindle** is that kind of **has-attachment** whose range is a **Spindle-Assembly**, which is a kind of **Thing-With-3D-Offset**.

Finally, we have **has-cutting-spindle**, which specializes **has-spindle** by tightening the range from a **Spindle-Assembly** to a **Cutting-Spindle-Assembly**. Now we have the ability to distinguish between machine descriptions based on spindle types. We will be able to ask how many spindles a particular machine has, how many cutting spindles it has, and if the numbers are different, perhaps infer that we have a hybrid Lathe/Milling machine. Or, if we know the machine is precisely a Live-Tooling-Machine, we might infer that any spindles it has are probably cutting spindles. The point is that at least we some information on which to base our reasoning.

Notice that in applying a relation, such as **has-cutting-spindle**, to a concept, we used **:at-least** and **:exactly** to specify the cardinality of the objects filling the roles. These and other relation forming operators give LOOM expressive power that differentiates it from traditional data bases.

Now we come back to the machine descriptions. First, a simple view of the **Milling-Machine** concept.

```
(defconcept Milling-Machine :is-primitive
  (and
    Live-Tooling-Machine
    (exactly 1 home-position)
    (at-least 1 has-machining-table)
    (at-least 1 motion-axis)
))
```

Whether this definition is correct or not is unimportant for now. We are just building up a taxonomy of terms. A model like this will change as we learn more about the domain and its differentiating terms. For the **Milling-Machine** we have added the roles for a home position, a machining table, and a motion-axis. Now, we come to one of Automation Intelligence's machine types:

```
(defconcept Machine-Type-2                    ;; AI machine pg. 2-17
   "4-Axis Machining Center with contouring Rotary Table"
  :is
  (and
    Milling-Machine
   (exactly 3 automated-primary-axis)
   (exactly 1 automated-rotary-axis)
   (the has-machining-table Rotary-Table)
   (same-as
         automated-rotary-axis
         (:compose has-machining-table motion-axis)
))
```

This "4-Axis Machining Center with contouring Rotary Table" is defined as a **Milling-Milling** with a **Rotary-Table** and 4 automated motion axis. 3 are the primary ones (X, Y, and Z), while the fourth is a rotary one (either A, B, or C). Also, the filler of the **automated-rotary-axis** is the same as the filler of **Rotary-Table**'s **motion-axis**

Notice how this definition carefully specializes a **Milling-Machine**. **Machine-Type-2**'s **machining table** is a **Rotary-Table**, which is a kind of **Table-Assembly** (as required by the **has-machining-table** role). A **Milling-Machine** has at least 1 **motion-axis**; **Machine-Type-2** has four motion axes, 3 are of type **automated-primary-axis** and 1 is an **automated-rotary-axis** one. Both are defined in terms of **motion-axis**.

Now that we have this detailed specialization taxonomy of terms, we can start to create instances of them.

C. The Extensions - Instances

Thus far, the model has just created abstract terms. It is as if we had defined a taxonomy of vehicles, cars, GM cars, Chevrolets, and Corvettes. Now we want to talk about that red Corvette in front of our building.

There may or may not be any real-world objects that are described by **Machine-Type-2**. If there are, they are represented separately from the descriptive terms. In LOOM we can create instances of the concepts. For example:

```
(tell (create m2-1 Machine-Type-2))
```

This statement creates an instance named **m2-1**, which is of type **Machine-Type-2**. And everything we know about **Machine-Type-2**, is true of **m2-1**. In fact, we know that it has one rotary axis. We do not know which one it is, only that it has one. In the Automation Intelligence configuration tool, if the user selects **Machine-Type-2**, one of the questions the system asks is which rotary axis does it have. The user types A, B, or C in the appropriate field. Here, in LOOM it is expressed as follows:

(tell (create abx-1 automated-B-axis))
(tell (automated-rotary-axis m2-1 abx-1))

The instance abx-1 is created to be one of the rotary axes and then the system is told that it is **m2-1**'s **automated-rotary-axis**.

All the unfilled roles can be specified or not. It depends on what is needed. If it is not important to know which rotary axis **m2-1** has, don't ask. The point is to have a model on which to base questions. The Automation Intelligence configuration program obviously needs all the specifications. Since it does, it uses defaults extensively. LOOM can express defaults as follows:

```
(defconcept Tilting-Table :is-primitive
 (and
  Table-Assembly
  (the motion-axis (or A-Axis B-Axis C-Axis)))
 :default
 (the motion-axis A-Axis))
```

This concept has a motion-axis role filled by one of the rotary axes. If it is not stated explicitly, then the **motion-axis** filler defaults to the **A-Axis**.

D. Queries and Retrieval

Once instances are created, they can be queried and retrieved. For example:

(ask (Live-Tooling-Machine m2-1))

asks if **m2-1** is a kind of **Live-Tooling-Machine**. This query returns true because **m2-1** is of **Machine-Type-2** which is a kind of **Milling-Machine** which is a **Live-Tooling-Machine**. Here is an example of a retrieval:

```
(retrieve (?x) (and (Milling-Machine ?x)
                    (Rotary-Axis (motion-axis ?x))))
```

This asks for all instances that are **Milling-Machines** and have a **motion-axis** that is a kind of **Rotary-Axis**. All instances which are of type **Machine-Type-2** will obviously be returned because of its definition, but perhaps there are other **Milling-Machines** who have a **motion-axis** that happens to be filled by a **Rotary-Axis**, but not necessarily by definition.

Notice how flexible this retrieval request is, especially when compared to typical data base requests.

III. THE VALUE OF THE MODEL

A. Inheritance and Specialization

This specialization taxonomy is laborious to construct, but there are many payoffs. The subsumption relationships are enforced by the classifier. For example, if you attempt to create an instance of a **Milling-Machine** without a cutting spindle, the system will recognize the inconsistency.

LOOM's use of classification is not typically found in systems that have the notion of classes. In frame systems or object-oriented programming languages classes are created mainly as places to put data structures and methods; subclasses can inherit unspecified data and methods from their supers. There is no notion of classification or recognition like LOOM has. LOOM's subclassing is limited. Roles must be added or tightened to create subclasses.

One way to specialize a concept is to specialize a filler of a role or the role itself, such as how **Cutting-Spindle-Assembly** specializes **Spindle-Assembly**. These fillers and roles are the "bounds" that restrict the values that can be taken on by its instances. In our example with **Machine-Type-2**, had the user told us that the **X-axis** is the **rotary-axis** of the machine, we would know by the definition of the role that the filler must be one of **A-Axis, B-Axis,** or **C-Axis**, and therefore his answer of **X-axis** would be rejected.

B. Knowledge Base vs. Data Base

Knowledge bases and data bases are different not only in form but in how they are used. A data base is a collection of tables, each table is a collection of records, and each record is a collection of fields. Each table is characterized by a schema which defines the fields in the record and perhaps some boundary conditions. Knowledge bases can be loosely defined in these terms as well.

In a LOOM knowledge base, each instance is like a data-base record. An instance's defining concepts are its schema. Its "fields" are the fillers for its roles. There is really no analog to a table, though if you focused on an important concept such as **Machine-Type-2** and listed all its instances, that list would be like a data-base table.

The key point is that every concept is the potential schema for some collection of instances. **Machine-Type-0, Machine-Type-2,** or **Cutting-Spindle-Assembly** can all be instantiated with their roles filled by other instances. But a concept is more than just a pattern used to itemize fields. Records have fixed fields, concepts have role forming operators such as **at-least, at-most,** and **exactly** that a system can reason about. Fields are simply names to a data-base, while concepts and roles form the elaborate taxonomy we have been describing.

For example, a data-base record for an employee might have two fields named primary-phone and backup-phone (or more likely, ph1 and ph2). Because of the names the data base searcher might recognize that information for what it is.

In LOOM, an employee concept might have two roles each filled by a **PhoneNumber** concept. One role might be called **primary-phone** which is a kind of **phone** relation and a kind of **primary** relation, the other role just called **phone**. There might be rules in the

knowledge base that say that in order to get in contact with someone, you should look for roles which are a kind of **communication** role. Retrieving those roles, you might get to the **phone** roles. Then, another heuristic might know about **primary** relations, that they might make a good metric on which to sort. And on and on and on. That kind of reasoning is unavailable to the data base programmer. LOOM's structured roles support this kind of reasoning.

A data base record's field names have no relation to one another other than through the semantics the user can infer from the names themselves. Data bases are organized for storing large amounts of similar data and quickly retrieving information from them. Knowledge bases are organized to strictly define terms and reason about their instances. Neither should be thought of as a substitute for the other. Using LOOM as a data base would be ludicrous. Using it to help define schemas for a data base would be fine.

IV. CONCLUSION

The machine model knowledge base described here is just in its infancy. Yet, even in this state some interesting concepts and relations have been defined. The purpose of this paper is to show that such a model is relevant and has value to a variety of potential users. We only hinted at the value of this knowledge base to Automation Intelligence's configuration program, but other machine applications would benefit from this knowledge base, if only to have common terms.

The knowledge base has high start up costs. It takes hundreds of concepts and relations to model even simple things. But there is significant added value. Many different paradigms can coexist within the LOOM knowledge base infrastructure. Object-oriented programming, production rules, and problem-solvers of all kinds could benefit from LOOM's classified taxonomy.

V. APPENDIX 1 - THE MODEL IN LOOM

```
(make-package "MACHINE-MODEL")
(in-package "MACHINE-MODEL")
(loom:use-loom "MACHINE-MODEL")
(defkb "MACHINE-MODEL-KB" nil :path-name "/u/pss/ai/loom/MachineModel.kb")
(change-kb "MACHINE-MODEL-KB")

(defrelation x-coordinate :range Number)
(defrelation y-coordinate :range Number)
(defrelation z-coordinate :range Number)

(defrelation object)
(defrelation position :is-primitive (:range 3D-Coordinate))
(defrelation has-attachment :is-primitive (:range Thing-With-3D-Offset))
(defrelation attached-to :is (inverse has-attachment))

(defrelation home-position :is-primitive
  (and position (:range Zero-3D-Coordinate)))

(defrelation has-spindle :is
  (and has-attachment
      (:range Spindle-Assembly)))
(defrelation spindle-orientation :domain Spindle-Assembly
            :range (one-of 'Horizontal 'Vertical))
(defrelation max-revs-per-minute :domain Spindle :range Integer)

(defrelation has-cutting-spindle :is
  (and has-spindle
      (:range Cutting-Spindle-Assembly)))
(defrelation has-primary-cutting-spindle :is-primitive has-cutting-spindle)
(defrelation hold-cutter :domain Cutting-Spindle
            :range (:or Cutter-Assembly Cutter))

(defrelation has-machining-table :is
  (and has-attachment
      (:range Table-Assembly)))
(defrelation part-of)
```

```
(defrelation motion-axis :range Motion-Axis)
(defrelation rotary-axis :is
  (and motion-Axis
       (:range Rotary-Axis)))
(defrelation primary-axis :is
  (and motion-Axis
       (:range primary-Axis)))
(defrelation automated-motion-axis :is
  (and motion-axis
       (:range Automated-Axis)))
(defrelation automated-primary-axis :is
    (and primary-axis automated-motion-axis))
(defrelation automated-rotary-axis :is
    (and rotary-axis automated-motion-axis))
(defrelation non-orthogonal-motion-axis :is-primitive motion-axis)
(defrelation name :range String)
(defrelation powered-by :range (one-of 'Automated 'Manual))

(defconcept 3D-Coordinate :is-primitive
  (and
   (exactly 1 x-coordinate )
   (exactly 1 y-coordinate )
   (exactly 1 z-coordinate )
))
(defconcept Zero-3D-Coordinate :is
  (and
   3D-Coordinate
   (the x-coordinate 0)
   (the y-coordinate 0)
   (the z-coordinate 0)
))

(defconcept Axis)
(defconcept X-Axis :is-primitive Axis)
(defconcept Y-Axis :is-primitive Axis)
(defconcept Z-Axis :is-primitive Axis)
(defconcept A-Axis :is-primitive Axis)
(defconcept B-Axis :is-primitive Axis)
(defconcept C-Axis :is-primitive Axis)
```

```
(defconcept Motion-Axis :is-primitive
  (and
   Axis
   (at-least 1 powered-by)))

(defconcept Rotary-Axis :is
  (and Motion-Axis
       (or A-Axis B-Axis C-Axis)))

(defconcept Primary-Axis :is
  (and Motion-Axis
       (or X-Axis Y-Axis Z-Axis)))

(defconcept Automated-Axis :is
  (and Motion-Axis (the powered-by 'Automated)))

(defconcept Manual-Axis :is
  (and Motion-Axis (the powered-by 'Manual)))

(defconcept Automated-X-Axis :is
  (and X-Axis Automated-Axis))
(defconcept Automated-Y-Axis :is
  (and Y-Axis Automated-Axis))
(defconcept Automated-Z-Axis :is
  (and Z-Axis Automated-Axis))
(defconcept Automated-A-Axis :is
  (and A-Axis Automated-Axis))
(defconcept Automated-B-Axis :is
  (and B-Axis Automated-Axis))
(defconcept Automated-C-Axis :is
  (and C-Axis Automated-Axis))

(defconcept Thing-With-3D-Offset :is
  (and
   (exactly 1 position)
   (exactly 1 object)))
```

```
(defconcept Machine)
(defconcept Machine-Attachment :is-primitive
  (and
   (the attached-to Machine)))

(defconcept Material-Removing-Machine :is-primitive
  (and Machine))
(defconcept Forming-Machine :is-primitive
  (and Machine))
(defconcept Cutting-Machine :is-primitive
  (and Machine))

(defconcept Live-Tooling-Machine :is-primitive
  (and Material-Removing-Machine
       (at-least 1 has-cutting-spindle)
       (exactly 1 has-primary-cutting-spindle)
))
(defconcept Dead-Tooling-Machine :is-primitive
  (and Material-Removing-Machine))
(defconcept Erosion-Machine :is-primitive
  (and Material-Removing-Machine))
(defconcept Grinding-Machine :is-primitive
  (and Material-Removing-Machine))

(defconcept Milling-Machine :is-primitive
  (and
   Live-Tooling-Machine
   (exactly 1 home-position)
   (at-least 1 has-machining-table)
   (at-least 1 motion-axis)
))
```

```
(defconcept Table)
(defconcept Table-Assembly :is
  (and
   Thing-With-3D-Offset
   (the object Table)
   (the attached-to Machine)
   (at-most 6 motion-axis)
))
(defconcept Tilting-Table :is-primitive
  (and
   Table-Assembly
   (the motion-axis (or A-Axis B-Axis C-Axis)))
  :default
  (the motion-axis A-Axis))

(defconcept Rotary-Table :is-primitive
  (and
   Table-Assembly
   (the motion-axis (or A-Axis B-Axis C-Axis)))                ;; parallel to spindle
  :default
  (the motion-axis B-Axis))

(defconcept Raw-Material)
(defconcept Workpiece :is-primitive
  (and
   Thing-With-3D-Offset
   (the object Raw-Material)
   (the attached-to Fixture)))

(defconcept Holder)
(defconcept Fixture :is-primitive
  (and
   Holder
   (the attached-to Table-Assembly)))

(defconcept Spindle :is
  (and
   (exactly 1 max-revs-per-minute)
))
(defconcept Cutting-Spindle :is
  (and Spindle
       (exactly 1 hold-cutter) ;; at-least  ??
))

(defconcept Spindle-Assembly :is-primitive
```

```
  (and
   Thing-With-3D-Offset
   (the object Spindle)
   (exactly 1 spindle-orientation)
))
(defconcept Cutting-Spindle-Assembly :is-primitive
  (and Spindle-Assembly
       (the object Cutting-Spindle)))

(defconcept Tilting-Head :is-primitive
  (and
   Cutting-Spindle-Assembly
   (the motion-axis (or A-Axis B-Axis C-Axis)))                ;;parallel to spindle
  :default
  (the motion-axis A-Axis))

(defconcept Nonorthogonal-Head :is-primitive
  (and
   Cutting-Spindle-Assembly
   (the motion-axis  C-Axis) ; main spindle axis
   (the non-orthogonal-motion-axis B-Axis)))

(defconcept Cutter)
(defconcept Cutter-Assembly)

(defconcept Machine-Type-0                                    ;; AI machine pg. 2-13
    "3-Axis Machining Center with or without Positioning Rotary Table"
  :is
  (and
   Milling-Machine
   (exactly 3 automated-motion-axis)
   (filled-by automated-motion-axis
              Automated-X-Axis Automated-Y-Axis Automated-Z-Axis)
   ;; notice that we didn't say how the motion axis worked, e.g., in the
   ;; Z-direction, does the spindle or the table move? IPOST doesn't care.
   ;; The instances will declare how the movement occurs.
))
```

```
(defconcept Machine-Type-2 ;; AI machine pg. 2-17
    "4-Axis Machining Center with with contouring Rotary Table"
  :is
  (and
   Milling-Machine
   (exactly 3 automated-primary-axis)
   (exactly 1 automated-rotary-axis)
   (the has-machining-table Rotary-Table)
))

(tell (create m0-1 Machine-Type-0))
(tell (create m2-1 Machine-Type-2))
(tell (create abx-1 automated-B-axis))
(tellm (automated-rotary-axis m2-1 abx-1))
(ask (Live-Tooling-Machine m2-1))
(retrieve (?x) (and (Milling-Machine ?x)
                    (Rotary-Axis (motion-axis ?x))))

(retrieve (?x ?y)
          (and (MACHINE-TYPE-2 ?x)
               (automated-rotary-axis ?x ?y)
               (Rotary-Axis ?y)
               (Automated-Axis ?y)))
(retrieve (?x) (and (MACHINE-TYPE-2 ?x)
                    (automated-B-axis (automated-rotary-axis ?x))))
```