Overview of Toolkits

Paul Chung
AI Applications Institute
Edinburgh University

OVERVIEW OF TOOLKITS

The toolkits

- ART (V 3.1) - Inference Corporation
- KEE (V 3.05) - Intellicorp
- Knowledge Craft (V 3.1) - Carnegie Group Inc.

The overview

- highlights the facilities that each toolkit provides
- points out the toolkits' similarities and differences

The overview is not an evaluation

- complex systems cannot be compared very well by just a table of options
- they show differences in different application areas

WHY MULTI-PARADIGM SYSTEMS?

- a large programming problem can usually be divided into a number of sub-problems
- a single paradigm may not be suitable for solving all of the sub-problems
- multi-paradigm systems give the programmer the right tool at the right time

Ways of building a multi-paradigm system

- design a new system that supports different paradigms
  - ART and KEE
- provide an interface for different languages
  - LOGLISP, POPLOG and Knowledge Craft
- augment an existing system with a new language construct
  - HOPE with unification

General components

- Schema Representation
- Rule Based Programming
- Object Oriented Programming
- Access Oriented Programming
- Context Manipulation
- Development Environment

SCHEMA REPRESENTATION

- group related information into a structure. Schemata are also called *frames* and *units*.
- a schema has slots and values
- Example
  {car-1
    reg: JB007
    colour: silver
    make: AM
  }
- schemata can be used to represent objects or concepts
- Inheritance and Specialization
  - enables the easy creation of objects that are *almost like* others with a few incremental changes
  - reduces the need to specify redundant information and simplifies updating and modification

RULE BASED PROGRAMMING

- knowledge is represented in rule form:
  If *Premises* then *Conclusions*
- rules are usually used in a forward chaining manner
- conflict resolution
  - refraction
  - recency
  - specificity
- rules may be used in a backward chaining manner
- truth maintenance
  - keeps track of dependencies between premises and conclusions
  - marks inconsistencies

OBJECT ORIENTED PROGRAMMING

- schemata are viewed as objects
- combines the properties of procedures and data
  - data abstraction
  - local states
- methods
- message passing
  - (Send-Message *Object Selector Arg1* ....)
  - a form of indirect procedure call
  - computation is invoked by the passing of *messages* between objects
- inheritance and specialization

ACCESS ORIENTED PROGRAMMING

- *active value* the mechanism underlying access-oriented programming
  - also known as *demon*
- activate computation when data are fetched or stored
- very useful if the value of one variable is dependent on the value of another
- used a lot in fancy graphical front end
- also used for constraint propagation

CONTEXT MANIPULATION

- context is also known as world
- when to use it
  - hypothetical situation
  - time
  - state in solution to a problem
- some operations
  - create new context
  - add and delete information from context
  - merge contexts

DEVELOPMENT ENVIRONMENT

- text editor
- file handler
- knowledge base browser
- interface development facilities
- rule debugger
- LISP debugger

ART - SCHEMA REPRESENTATION

- user defined relation
  - new relation
  - transitivity
  - inverse
  - controlled inheritance
- multiple inheritance
- single value and multi-value slots
- automatic creation of objects

ART - RULE BASED PROGRAMMING

- allows very complex expressions on the LHS, including arbitrary LISP expressions
- forward chaining rules

```
(DEFRULE EXAMPLE-RULE-1
 (LIKE ?X ?Y)
 (SWEET ?Y)
 =>
 (ASSERT (LIKE-SWEET-THINGS ?X)))
```

- backward chaining rules

```
(DEFRULE EXAMPLE-RULE-3
 (GOAL (LIKE-SWEET-THINGS ?X))
 (LIKE ?X ?Y)
 (SWEET ?Y)
 =>
 (ASSERT (LIKE-SWEET-THINGS ?X)))
```

- logical dependencies

```
(DEFRULE EXAMPLE-RULE-3
 (LOGICAL (ON ?X TABLE)
  (ON ?Y ?X)
  (ON ?Z ?Y))
 =>
 (ASSERT (ONE BIG PILE)))
```

- conflict resolution
  - salience then recency

ART - OBJECT ORIENTED PROGRAMMING

- creating a method
  - (defaction *action schema arg1* ....)
- sending a message
  - (invoke *action schema arg1* ....)
  - variations
    - before
    - after
    - whopper

ART - ACCESS ORIENTED PROGRAMMING

- attach an action to a slot
- activation
  - get, put, modify
  - before, after

ART - CONTEXT MANIPULATION

- create, delete and merge contexts
- add and delete information
- believe (select) context
- poison context
  - schema value conflict
  - retraction from assumptive base
  - other user defined rules that look out for contradictions
- inheritance of contexts (context tree)
- if a slot value is changed in a child context, only that slot is copied
- multiple levels of contexts

KEE - SCHEMA REPRESENTATION

- **subclass** and **member** relations only
- multiple inheritance
- two kinds of slots
  - own slot - not inherited
  - member slot - inherited by subclasses and members
- facets
  - define slot usage
  - inheritance (e.g. override, union)
  - type and range of value
  - cardinality
  - others and user defined ones

KEE - RULE BASED PROGRAMMING

- two kinds of rules
  - Standard Rule
    (IF *Premises* THEN DO *Conclusions*)
  - Deduction Rule
    (WHILE *Premises* BELIEVE *Conclusions*)
- related rules are grouped into a rule class
- the same rules are used for forward and backward chaining
  - start forward chaining
    (ASSERT
    '(THE COLOUR OF CAR IS RED)
    'CHOOSE.OWNER.TYPE.RULES)
  - start backward chaining
    (QUERY
    '(THE OWNER OF CAR IS ?WHO)
    'CHOOSE.OWNER.TYPE.RULES)
- conflict resolution
  - depth first or breadth first
  - deduction rule, standard rule, new world rule
  - most general rule or most specific
  - weight (same as salience)

KEE - OBJECT ORIENTED PROGRAMMING

- KEE was originally developed as an object oriented system, so this part of KEE is very well developed
- creating a method
  - assign the slot value class to **'method**
  - assign the slot value to be a lambda expression or a function name
- sending a message
  - (unitmsg *schema slot arg1* ....)

KEE - ACCESS ORIENTED PROGRAMMING

- attach a function to a slot
- activation
  - get
  - put
  - add
  - remove

KEE - CONTEXT MANIPULATION

- create, delete and merge contexts
- add and delete information
- inheritance of contexts
- if a slot value is changed in a child context, only that slot is copied
- create exclusion contexts, these contexts and their children cannot be merged later
- detection of inconsistent context
  - violation of cardinality, type or range
  - user supplied rules that detect inconsistent state
- an inconsistent context remains
  - can get explanation for inconsistency
  - can resurrect context by removing reason for inconsistency

KNOWLEDGE CRAFT - SCHEMA REPRESENTATION

- user defined relations
  - new relation
  - transitivity
  - inverse
  - controlled inheritance
- multiple inheritance
- meta-information is encapsulated in schema
  - meta-schema: a schema representing meta-knowledge about another schema
  - meta-slot: a schema representing meta-knowledge about a slot
  - meta-value: a schema representing meta-knowledge about a value
  - meta information is not inherited
- slot restriction
  - domain
  - range
  - cardinality

KNOWLEDGE CRAFT - RULE BASED PROGRAMMING

- CRL-OPS (for forward chaining)
  - an enhanced version of OPS5 (for example, allows LISP expressions on the LHS of a rule)
  - conflict resolution
    - recency
    - most specific rule
  - CRL-OPS' own working memory is not accessible from outside OPS
- CRL-PROLOG (for backward chaining)
  - LISP syntax rather than Edinburgh Prolog syntax
  - access to LISP via *call* and *bind*
  - simple schemata interface

KNOWLEDGE CRAFT - OBJECT ORIENTED PROGRAMMING

- creating a method
  assign the slot value to be a function name
- sending a message
  - (call-method *schema slot arg1* ....)

KEE - ACCESS ORIENTED PROGRAMMING

- attach a function to a slot
- activation
  - get, put, delete
  - before, after

KNOWLEDGE CRAFT -
CONTEXT MANIPULATION

- create, delete and select contexts
- copy and move schemata
- add and delete information
- inheritance of contexts
- once a slot value is changed in a child context the whole schema is copied
- no consistency check

ARTICLES

Fraser, J.L. (1987) *Overview of KEE.* Airing, No 2. AI Applications Institute, Edinburgh University.

Inder, R. (1987) *The State of the Art.* Airing, No 1 and 2. AI Applications Institute, Edinburgh University.

Kingston, J. (1987) *A Technical Review of Knowledge Craft.* Airing, No 3 and 4. AI Applications Institute, Edinburgh University.

Lurent, J., Ayel, J., Thome, F. and Ziebelin, D. (1986) *Comparative Evaluation of Three Expert System Development Tools: KEE, Knowledge Craft, ART.* The Knowledge Engineering Review, Vol 1, No 4, pp18-29.

Richer, M.H. (1986) *An Evaluation of Expert System Development Tools.* Expert Systems, Vol 3, No 3, pp166-183.

Wall, R.S., Apon, A.W., Beal, J., Gately, M.T. and Oren, L.G. (1985) *An Evaluation of Commercial Expert System Building Tools.* Data & Knowledge Engineering, Vol 1, pp279-304.