A Class Library Implementation of a Principled Open Architecture Knowledge Representation Server with Plug-in Data Types

Brian R Gaines
Knowledge Science Institute
University of Calgary
Calgary, Alberta, Canada T2N 1N4.
gaines@cpsc.ucalgary.ca

Abstract

A knowledge representation server is described which provides a fast, memory-efficient and principled system component. Modeling the server through intensional algebraic semantics leads naturally to an open-architecture class library into which new data types may be plugged in as required without change to the basic deductive engine. It is shown that the operation of an existing knowledge representation system, CLASSIC, may be implemented through one data type supporting sets with upper and lower set and cardinality bounds. The architecture developed is cleanly layered by complexity of inference so that fast propagation of constraints is separated from potentially slow model-checking search. Client programs may obtain estimates of the complexity of a request, and may control the resources allocated to its complete solution.

1 Introduction

One of the basic technologies that may be factored out of virtually all artificial intelligence systems is some form of knowledge representation service. Theoretical foundations have been developed for such technologies in recent years [Brachman and Levesque, 1984; Ait-Kaci, 1986; Nebel, 1990]. These foundations make it possible to develop general-purpose knowledge representation services that are well-principled, space and time efficient, and embeddable as sub-systems in a wide variety of applications.

Since complexity analyses shows that knowledge representation systems with even minimal features are able to represent intractable problems [Brachman and Levesque, 1984; Nebel, 1988; Schmidt-SchauB, 1989], it has been suggested that the representation and deduction capabilities of knowledge representation services should be limited [Levesque and Brachman, 1987). The reasons for and against this have been surveyed by Doyle and Patil [1991] who conclude that, while there are sound arguments for such limitations, the capabilities of the resultant systems will often fail to satisfy reasonable application requirements. One impact of this is that system designers may add functionality that provides the missing capabilities in ways that arc less principled than those of a general server. A second is that problems may have to be represented in unnatural ways that are conducive to poor performance. A third is that the effect of the limitations on deductive capabilities may not be apparent to users, leading to errors.

This motivates an *open-architecture* server design to which functionality may be added in a principled fashion, implemented as a *class library* with well defined interfaces to new classes for additional data types. The inherent intractability of some representable problems, contrasted with adequate practical performance of existing knowledge representation systems [Vilain, 1991], suggests that services need to be *layered.* Some deductions can be carried out in low-polynomial time and space whereas others may involve a search whose lower bound is at least exponential in the size of the problem. An architecture which separates these deductions into separate layers of service is desirable in applications, particularly if the computationally expensive ones can be carried out asynchronously in the background, provide useful partial results, and can be terminated before completion if appropriate. To allow users to estimate the cost of using a service, it is important to instrument the complexity of knowledge structures to provide meta-services that estimate the tractability of a problem.

This paper describes the design and implementation of an open architecture knowledge representation server, KRS, as a class library in C++. The server was originally modeled on CLASSIC [Borgida, Brachman, McGuiness and Resnick, 1989], and designed to be the kernel of a new family of interactive knowledge acquisition tools, and its architecture and applications as an artificial intelligence tool have already been described [Gaines, 1991a], as have the associated rule system [Gaines, 1991b], visual knowledge representation language [Gaines, 1991c], and its application to organizational modeling and problem solving [Gaines, 199Id]. This paper focuses on the underlying principles and the way in which they determine an architecture for term subsumption knowledge representation systems.

2 Theoretical and Practical Framework

The most widely used framework for the formal analysis of knowledge representation has been the standard model and proof theory of first order logic. However, there are alternative algebraic models for first and higher order logics that are becoming increasingly used in programming language semantics because they represent abstract data types simply and naturally. Ait-Kaci [1984, 1986] has given a lattice-theoretic model of knowledge base languages with operational semantics through term rewriting that resolves many of the issues of complexity and deduction algorithms for term subsumption knowledge representation.

The merits of algebraic, type-theoretic semantics for knowledge representation are that they provide formal models and complexity analyses that relate closely to the features and issues of existing knowledge representation systems. For example, they provide simple accounts of common constraints, such as cardinality, extensional inclusion and numeric ranges. Clearly, these are questions of naturality rather than logical power, since accounts of set theory and arithmetic can be developed in first order logic, and algebraic feature constraint logics may be translated into first-order formulae [Smolka, 1992]. However, natural models giving minimal formulations of representational requirements are valuable in both the software engineering of knowledge representation servers, and the effective presentation of the services offered to those using them.

The model for term subsumption knowledge representation used in the development of KRS is that the computational units arc: for the TBox, *concepts,* identified with collections of *constraints* indexed by roles; and for the ABox, *individuals* represented as unique identifiers, each of which has associated conceptual variable whose value in any particular state of the knowledge base is a concept. This has the normal semantics for KL-ONE systems that concepts are precisely a composition of constraints—changing the constraints changes the identity of the concept, whereas individuals are precisely identities of objects-changing the associated constraints changes the *state* of the individual, not its identity. In Zalta's [1988J terminology, abstract objects (concepts) *encode* properties whereas concrete objects (individuals) *exemplify* them.

Thus, the computational structure of both theory and implementation is modeled as an algebra of constraints, but the nature of the constraints is left undefined. Any data type can be used in representation and deduction provided it can be modeled as a constraint algebra. In KRS the data types and their operations arc implemented as separate classes, and 'plugging in' new data types involves adding a new class with four associated operations (composition, subsumption test, input and output). The kernel deduction systems for constraint propagation, for rule, inverse role and coreference application, and for model checking search, remain unchanged.

3 Constraint Algebras

The semantics of constraints may be developed directly from informal requirements to a formal model. The key notions are that the *composition* of two constraints should be a well-defined constraint (binary function), that it makes no difference to compose a constraint with itself (idempotency), that grouping of multiple constraints in resolving them to binary compositions makes no difference (associativity), and that the order of application of constraints makes no difference (commutativity). Without these requirements, one would have the semantics of general *operators* rather than constraints. Together they imply that composition generates a *semi-lattice* in which it is the *join* operation. There is a natural order relation of *subsumption* of constraints, defined as one constraint subsumes another if their composition equals the second constraint. The semi-lattice can then be extended to be a full lattice by defining a dual, order-inverting *meet* operation (which, as a side effect, adds the lattice *adsorption* identities [Gratzer, 19711). A unique lowest element, or *zero,* can be defined in the lattice corresponding to the composition of incompatible constraints. A unique greatest element, or *unit,* can be defined corresponding to a universally applicable constraint.

Thus, the lattice structures common to all knowledge representation systems arise out of the basic primitive of a constraint. Any mathematical or logical formulation is a representation of the properties of this primitive, and any representation schema with reasonably normal semantics will have a constraint algebra interpretation.

Formally, a *constraint algebra* is defined as a pair of binary operations upon a set, L, $\sqcap$ (composition) and $\sqcup$ (minimum common generalization), unique greatest and least elements (universal and incoherent constraints), defined ideal sub-lattices (types), defined order and incompatibility relations, $\rightarrow$ (subsumption) and — (disjoint), and defined atomic elements (values), satisfying the following:

Idempotent $\quad x \in L \Rightarrow x \sqcap x = x,\ x \sqcup x = x$ (L1)

Commutative $\quad x, y \in L \Rightarrow x \sqcap y = y \sqcap x,$
$x \sqcup y = y \sqcup x$ (L2)

Associative $\quad x, y, z \in L \Rightarrow x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z,$
$x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$ (L3)

Absorptive $\quad x, y \in L \Rightarrow x \sqcap (x \sqcup y) = x,$
$x \sqcup (x \sqcap y) = x$ (L4)

Zero 0 $\quad x \sqcap 0 = 0,\ x \sqcup 0 = x$ (L5)

Unit 1 $\quad x \sqcap 1 = x,\ x \sqcup 1 = 1$ (L6)

Ideal J $\quad x \in J, y \in L \Rightarrow x \sqcap y \in J$ (L7)

Atom x $\quad x \in L \Rightarrow a \sqcap x = x$ or $a \sqcap x = 0$ (L8)

Order $\rightarrow$ $\quad x \rightarrow y \Leftrightarrow x \sqcap y = x,\ x \sqcup y = y$ (L9)

Incompatible — $\quad x — y \Leftrightarrow x \sqcap y = 0$ (L10)

The semantic notions of knowledge representation can now be modeled formally by taking any bounded lattice as a constraint algebra in which the atoms arc *values,* other elements are *constraints* upon those values, ideals are *types* of values, the order relation is one of constraint *subsumption,* the incompatibility relation as one of *disjoint* constraints, the unit element as a *universal* constraint and the zero element as an *incoherent* constraint. The next section defines types for record structures as constraint algebras giving a complete account of the semantics of term subsumption knowledge representation schema.

4 Constraint Algebras for the T-box and A-box

To develop the semantics for concepts in the T-box and worlds of individuals in the A-box, take any constraint algebra, L, and consider an indefinite product of such algebras, $X = L \times L \times L \ldots$, indexed by a set of projections, Π, such that $\pi \in \Pi,\ x \in X \Rightarrow \pi x \in L$. The *arity* of an element of X is defined to be the number of non-1 projections. There is a natural constraint lattice formed by X under the definitions:

$$x, y \in X \Rightarrow \pi(x \sqcap y) = \pi x \sqcap \pi y, \pi(x \sqcup y) = \pi x \sqcup \pi y \quad (X1)$$

$$\pi 1 = 1 \quad (X2)$$

$$\pi O = O \quad (X3)$$

$$\pi x = O \Rightarrow x = O \quad (X4)$$

That is, the lattice operations operate on a component by component basis, and any projection becoming zero implies that the product is itself zero. The order relation of (L5) may also be computed on a component by component basis:

$$x, y \in X, x \rightarrow y \Leftrightarrow \forall \pi \in \Pi\ \pi y \rightarrow \pi y \quad (X5)$$

The base algebra L may be treated as a sublattice of X by defining a specific $\underline{\pi} \in \Pi$ and the mapping, $\mu: A \rightarrow X$:

$$a \in A \Rightarrow \pi\mu a = a \text{ if } \pi = \underline{\pi}, \text{ else } \pi\mu a = 1 \quad (X6)$$

(X1) through (X6) construct one constraint lattice as the indefinite product of another such that the original lattice may be mapped to a sublattice of the product. This construction may be iterated indefinitely to construct a well-founded sequence of constraint lattices of increasing order corresponding to the well-founded construction of sets in classical set theory. The *order* of an element of one of the lattices is defined to be the position in this sequence of its initial definition. The *complexity* of an element is defined recursively as its arity plus the arities of each of its non-unit projections—this corresponds to the number of non-unit modes in its expansion or graph.

4.1 T-box Representation

If the projections are defined to be *roles* and the lattice elements *concepts,* this construction is adequate to account for representation and deduction in the T-box of term subsumption systems. Concept subsumption is the order relation of (X5) and individual classification is the same relation if the A-Box is taken simply to associate an individual with a concept asserted of it.

Some issues relating to the complexity of subsumption and classification are now apparent. The subsumption of (X5) for elements of a given order in a system that stores only non-unit projections and caches precomputcd lower order subsumptions involves only n cache accesses where n is the minimum of the arities of the elements being compared. This model of subsumption computation gives an upper bound that makes it appear highly tractable. However, there are a number of reasons why this result is misleading.

First, the subsumption relation in the base lattice will generally not be cached because the space complexity will be high even if the time complexity is not, and the time complexity could also be high in an open architecture system in which arbitrary lattices can be plugged in (including the product lattices just defined). In practice, the base lattices currently used in systems such as CLASSIC are such that subsumption computation is tractable as will be shown in the next section.

Second, the composition of concepts may generate large numbers of additional (anonymous) concepts such that the caching assumed may itself become intractable in space and time. Nebel [1990] has given a simple construction of a series of related concepts where the number of anonymous concepts grows exponentially with the number of originally defined concepts. In KRS the number of anonymous concepts generated by a given concept definition is reported as an auxiliary complexity measure.

Third, it is unrealistic to implement assertions about individuals in a caching scheme. Individual classification may involve a complete expansion of the concepts testing each node at the base lattice level. The number of tests is then the complexity defined above which can also grow exponentially with the number of concepts defined.

Fourth, the subsumption relation defined in (X5) is intensional. It docs not take into account the constraints that may be propagated in the A-box to ensure that the concepts associated with individuals cannot become incoherent. Extensional subsumption taking into account such constraint propagation is consistent with, but a stronger partial order, than intensional subsumption. This is how intractability may be shown in CLASSIC by representing satisfiability of a restricted CNF formula [Lenzerini and Schaerf, 1991].

4.2 A-box Representation

If the projections are defined to be *individuals* and the lattice elements *worlds,* the construction, (XI) through (X5), is adequate to account for representation and some aspects of deduction in the A-box of term subsumption systems. World subsumption through the order relation of (X5) corresponds to monotonic reasoning, that the subsumed world can be reached from the other by a series of assertions about individuals. However, there are additional modes of reasoning in the A-box that make its analysis more complex.

First, CLASSIC-like systems implement rules as pairs of concepts such that if an individual is classified as falling under the first then the second is asserted of it.

Second, individuals may participate in base lattice constructions, for example through extension and cardinality constraints on sets of individuals, and hence assertions of a concept as applying to one individual may imply that other concepts apply to other individuals. In particular, cardinality constraints may imply the existence of (Skolem) individuals not previously created.

Third, there may be assertions about individuals that, while they do not lead to incoherence directly, do so indirectly through the previous two mechanisms. Taking into account these implicit constraints may involve a search over possible worlds that may grow exponentially with the number of individuals involved.

The first two modes of reasoning do not in themselves lead to intractability but, taken together with the third, they allow intractable problems to be represented in the A-box that go beyond those already discussed for the T-box.

5 Base Algebras for CLASSIC-Like Systems

Specific base lattices for CLASSIC-like systems may be constructed from a four element constraint algebra as shown on the left of Figure 1 in which Ⓞ is the zero element, Ⓘ the unit element, and Ⓐ and Ⓝ are complementary elements distinct from them. The semantics of this lattice is that of role constraints:

I n d e t e r m i Ⓘ e a role possibly existing.

O v e r d e t e r m Ⓞ t e a role existing but unfillable.

A p p l i c a Ⓐ ɜ a role definitely existing.

Nonapplicable Ⓝ a role definitely not existing.

This lattice supports no data types in itself but it can be extended to support any arbitrary combination of data types by inserting their constraint lattices between Ⓐ and Ⓞ such that their unit elements represents the relevant types and their zero elements are mapped to Ⓞ. Figure 1 center shows two mutually incompatible constraint lattices inserted in this way. Figure 1 right shows two compatible lattices inserted such that one is a sublattice of the other, corresponding to type coercion.

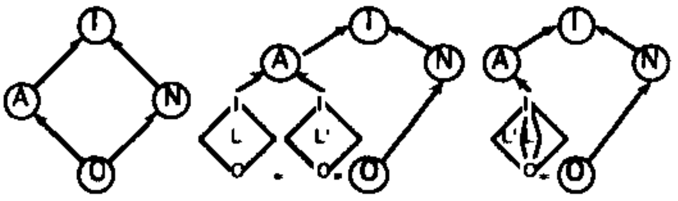

Figure 1 The basic constraint algebra and insertion of data types

In CLASSIC-like systems there is a distinguished data type based on sets of individuals. The algebra of subsets of a set with set union as meet and set intersection as join is a constraint lattice. However, its semantics do not support the distinction between FILLS and ONE-OF constraints, the CLOSE constraint, or cardinality constraints. The following constructions do so by combining the constraints of the algebra of subsets with those of the dual lattice.

Consider an arbitrary set, S, its powerset, PS, and the Cartesian product on its powerset, $P = \mathcal{P}S \times \mathcal{P}S$. There is a natural constraint lattice formed by P under the definitions:

$$x = (l,u),\ y = (l',u') \in P \Rightarrow x \sqcap y = (l \cup l', u \cap u'),\quad x \sqcup y = (l \cap l', u \cup u') \qquad (P1)$$

$$x = (\{\},S) \Leftrightarrow x = 1 \qquad (P2)$$

$$x = (l,u),\ l \not\subset u \Leftrightarrow x = 0 \qquad (P3)$$

The semantics of this lattice are those of constraints upon the subsets of S with the first projection of the product being a lower bound corresponding to FILL, the second projection being an upper bound corresponding to ONE-OF, unsatisfiable constraints in which the lower bound is not contained in the upper bound mapping to zero, and the lattice operations corresponding to complementary pairs of intersections and unions of lower bounds and upper bounds. The atoms of this lattice are elements in which the lower and upper bounds coincide. They provide the semantics for CLOSE in corresponding to a set constrained to consist precisely of its specified members.

The SAME-AS constraint corresponds to an equality assertion between nodes of a concept structure. It is simply implemented in the A-box by indirect addressing. The semantics of **NOT** in **KRS** are represented by having a complement as the second projection, that is S-u rather than u. Complements behave well under intersection and union with sets and other complements, generating either sets or complements, and causing no problems in implementation. In particular, this allows a very simple implementation of disjoint primitives as shown later.

Cardinality constraints may be added to this structure to give an extended constraint lattice based on the product $C = \mathcal{P}S \times \mathcal{P}S \times I \times I$, where I is the set of non-negative integers. There is a natural constraint lattice formed by C under the definitions:

$$x = (l,u,m,n),\ y = (l',u',m',n') \in C \Rightarrow$$

$$x \sqcap y = (l \cup l', u \cap u', \max(m,m',\mathcal{C}(l \cup l')), \min(n,n',\mathcal{C}(u \cap u')))$$

$$x \sqcup y = (l \cap l', u \cup u', \min(m,m',\mathcal{C}(l \cap l')), \max(n,n',\mathcal{C}(u \cup u'))) \qquad (C1)$$

$$x = (\{\},S,0,\mathcal{C}S) \Leftrightarrow x = 1 \qquad (C2)$$

$$x = (l,u),\ l \not\subset u \vee m > n \Leftrightarrow x = 0 \qquad (C3)$$

The semantics of this lattice are those of constraints upon the subsets of S as before with the additional constraints on cardinality corresponding to **AT-LEAST** and **AT-MOST**. Figure 2 shows an example of this lattice for two elements.

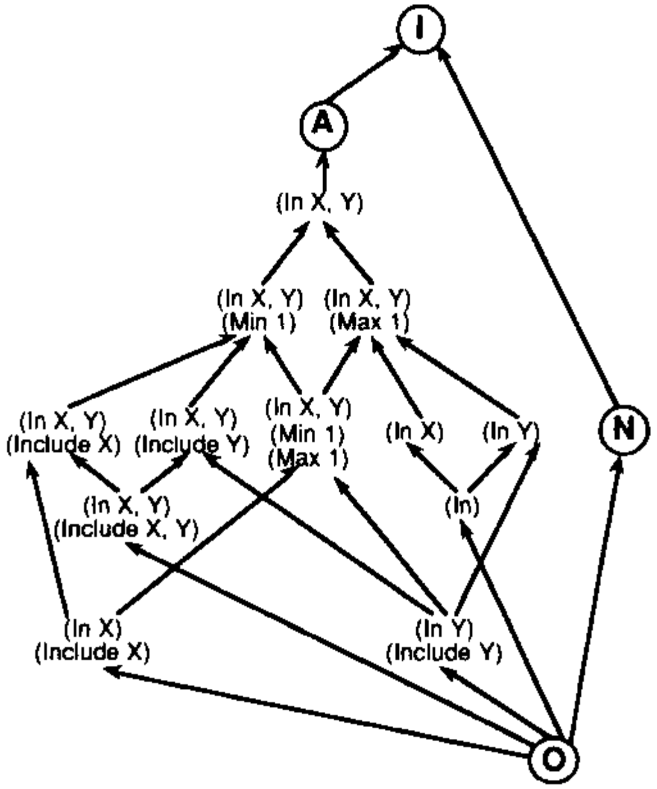

Figure 2 Example type lattice for set valued roles

PRIMITIVE and DISJOINT-PRIMITIVE constraints are implemented through a hidden role filled by elements of a P lattice based on the set of concepts rather than that of individuals. A concept defined to be primitive stores its own name as FILLS in this role. A concept defined to be disjoint to another stores the other's name as NOT in this role. This results in the proper semantics for primitive concepts.

Thus, the capabilities of CLASSIC arc implemented in KRS through a single data type which is a set with upper and lower bounds and cardinality constraints. It is a an unusual data type compared with those of conventional programming languages, but it is readily implemented and optimized for fast and memory-efficient operation.

A principled implementation of some applications of the TEST constraint used to extend CLASSIC is made available in KRS by direct support of data types such as integers, time intervals, strings, and so on, through the lattice constructions of Figure 1. This allows general abstract data types to be integrated in CLASSIC-like systems. What it does not support is operations upon them that do not correspond to the lattice joins and meets, and the use of side-effects to communicate with other subsystems.

6 Architecture of KRS

The KRS architecture is shown in Figure 3. It supports the normal features of KL-ONE-like systems in allowing concepts to be defined, and assertions to be made about individuals in terms of these concepts. What is essentially type propagation inference can then be used to deduce the consequences of the assertions through reference to the definitions. KRS also supports the rule schema of CLASSIC, extended to handle rules with exceptions, such that an individual recognized as satisfying one concept has another automatically asserted of it. This integrates production rule and frame-based reasoning. It also supports model checking [Halpern and Vardi, 1991] or "puzzle mode" reasoning in which, if necessary, after propagation of type and rule constraints a search of possible worlds is carried out to determine whether additional conclusions can be drawn because not to do so would lead to absurdity.

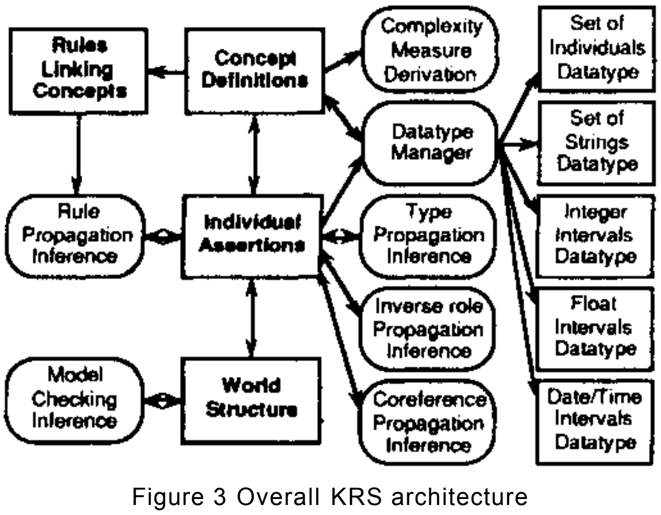
Rules Linking Concepts
Concept Definitions
Complexity Measure Derivation
Set of Individuals Datatype
Datatype Manager
Set of Strings Datatype
Rule Propagation Inference
Individual Assertions
Type Propagation Inference
Integer Intervals Datatype
Inverse role Propagation Inference
Float Intervals Datatype
Model Checking Inference
World Structure
Coreference Propagation Inference
Date/Time Intervals Datatype

Figure 3 Overall KRS architecture

KRS supports a wide variety of data types through a data type manager accessing separate modules through a uniform interface such that constraints supported, such as interval bounds on numbers, enter fully into concept definitions, individual assertions and deductive inference. Complexity measures are derived from the concept definitions that estimate the probable costs of problem solving in a particular domain.

The algebraic model of knowledge representation leads directly to an open architecture implementation as a class library as illustrated in Figure 4. The knowledge base class has two main instance variables, one holding individual records and the other holding concept records. The base knowledge base class implements the type lattice on the left of Figure 1, having codes for indeterminate, applicable, non-applicable and overdeterminate values and constraints. It also supports other constraints by reference codes containing a type code and pointer. The type code is used to access a list of data type support objects which is initially empty. For each data type implemented an object is added to the list that supports it by providing storage for values and constraints of that subtype, and methods to compute and support input/output with such values and constraints.

Figure 4 at the bottom left also illustrates another important feature of the class library construction. Type subsumption computation for record structures is simply expressed recursively. However, to attain the known complexity lower bound requires that already computed subsumption relations be cached to avoid duplicate computation. This caching is implemented in a sub-class in KRS, allowing the simple recursive computation to be called as an inherited method during debugging as a check on the correctness of the caching algorithm.

The utility of the class library construction are also apparent in the implementation of model checking reasoning in KRS. This is the mode that exhibits intractability so that it is appropriate to implement it as a separate service. It involves a search of possible worlds once other forms of inference have reached a fixed point. The form of reasoning is that certain further constraints upon existing individuals that appear possible may not be so because their consequences are inconsistent. If one searches all possible worlds resulting from the assertion of combinations of such constraints then constraints which apply in all consistent worlds are necessary and can be deduced. Puzzles are usually designed so that there is only one world that is consistent, but real problems usually involve multiple possible worlds as consistent extensions. In KRS this mode of reasoning is implemented by making the individual records instance variable shown at the top of Figure 4 a pointer to a tree structure of possible worlds. The rest of the reasoning and type system remains unchanged.

6 Conclusions

Now that the theoretical foundations of knowledge representation servers have been developed in terms of abstract data types, and the demand for embeddablc implementations is growing, it is attractive to investigate the possibility of class libraries supporting efficient, extensible implementations. This paper has specified the requirements, detailed the relevant theory, and shown how this has led to a class library implementation of a fast, principled, open architecture knowledge representation server.

Acknowledgements

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada.

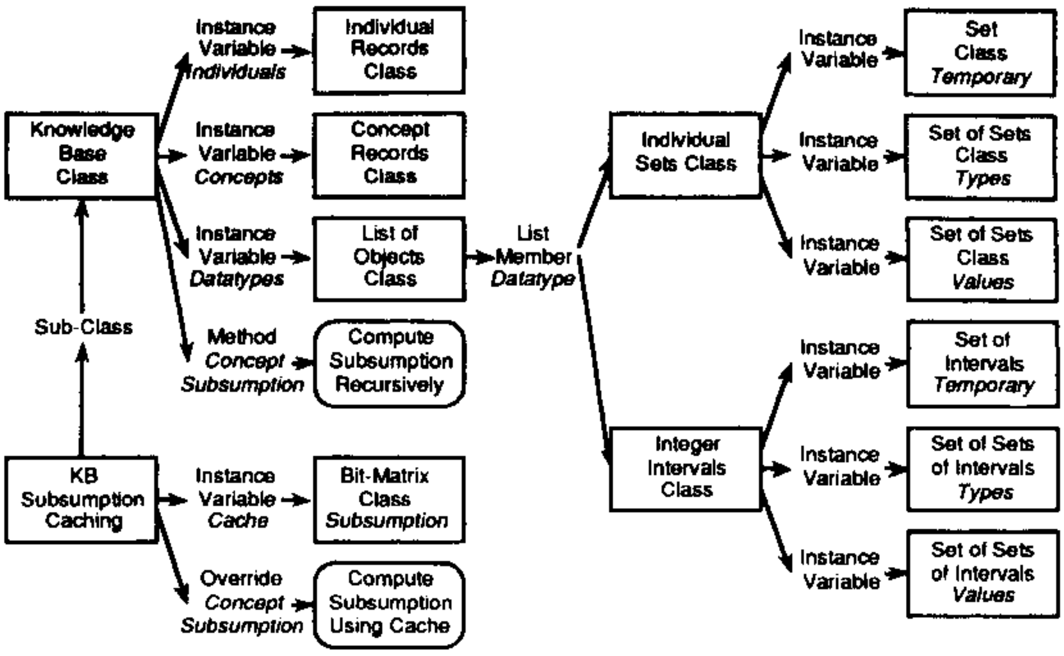
Knowledge Base Class
Instance Variable Individuals
Individual Records Class
Instance Variable Concepts
Concept Records Class
Instance Variable Datatypes
List of Objects Class
List Member Datatype
Method Concept Subsumption
Compute Subsumption Recursively
Sub-Class
KB Subsumption Caching
Instance Variable Cache
Bit-Matrix Class Subsumption
Override Concept Subsumption
Compute Subsumption Using Cache
Individual Sets Class
Instance Variable
Set Class Temporary
Set of Sets Class Types
Set of Sets Class Values
Integer Intervals Class
Set of Intervals Temporary
Set of Sets of Intervals Types
Set of Sets of Intervals Values

Figure 4 Part of knowledge base class lihrarv structure

References

[Ait-Kaci, 1984] H. Ait-Kaci. A lattice-theoretic approach to computation based on a calculus of partially-ordered types. PhD. University of Pennsylvania, 1984.

[Ait-Kaci, 1986] H. Ait-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science* 45 293-351, 1986.

[Borgida, Brachman, McGuiness and Resnick, 1989] A. Borgida, R.J. Brachman, D.L. McGuiness and L.A. Resnick. CLASSIC: a structural data model for objects. *Proceedings of 1989 SIGMOD Conference on the Management of Data,* pp.58-67. ACM Press. 1989.

I Brachman and Levesque, 1984] RJ. Brachman and H.J. Levesquc. The tractability of subsumption in frame-based description languages. *Proceedings of AAAI-84.* pp.34-37. Morgan Kauffman. 1984.

IDoyle and Paul, 1991] J. Doyle and R.S. Patil. Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence* 48(3) 261-297, 1991.

I Gaines, 1991a] B.R. Gaines. Empirical investigations of knowledge representation servers: Design issues and applications experience with KRS. *ACM SIGART Bulletin* 2(3)45-56,1991.

[Gaines, 1991b] B.R. Gaines. Integrating rules in term subsumption knowledge representation servers. *AAA/'9J : Proceedings of the Ninth National Conference on Artificial Intelligence,* pp.458-463. AAAI Press, 1991.

IGaincs, 1991c] B.R. Gaines. An interactive visual language for term subsumption visual languages. *IJCAI'91: Proc. Twelfth International Joint Conference on Artificial Intelligence,* pp.817-823. Morgan Kaufmann. 1991.

[Gaines, 1991d] B.R. Gaines. Organizational modeling and problem solving using an object-oriented knowledge representation server and visual language. *COCS'91: Proceedings of Conference on Organizational Computing Systems,* pp.80-94. ACM Press. 1991.

(Gratzer, 1971] G. Gratzer. *Lattice Theory.* Freeman, 1971.

[Halpern and Vardi, 1991] J.Y. Halpern and M.Y. Vardi. Model checking vs. theorem proving: a manifesto. *Artificial Intelligence and the mathematical Theory of Computation,* pp. 151-176. Academic Press. 1991.

[Lenzerini and Schaerf, 1991] M. Lenzerini and A. Schaerf. Concept languages as query languages. *AAAI'91: Proceedings of the Ninth National Conference on Artificial Intelligence,* pp.471-476. AAAI Press, 1991.

[Levesquc and Brachman, 1987] H.J. Levesquc and R.J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence* 3 78-93, 1987.

[Nebel, 1988] B. Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence* 34 371-383, 1988.

[Nebel, 1990] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems.* Berlin: Springer, 1990.

[Schmidt-SchauB, 1989] M. Schmidt-SchauB. Subsumption in KL-ONE is undecidable. *KR'89: First International Conference on Principles of Knowledge Representation and Reasoning,* pp.421-431. Morgan Kauffman. 1989.

[Smolka, 1992] G. Smolka. Feature-constraint logics for unification grammars. *Journal Logic Programming* 12(1-2)51-87, 1992.

[Vilain, 1991] M. Vilain. Deduction as parsing: tractable classification in the KL-ONE framework. *AAAI'91: Proceedings of the Ninth National Conference on Artificial Intelligence,* pp.464-470. AAAI Press, 1991.

[Zalta, 1988] E.N. Zalta. Intensional Logic and the Metaphysics of Intentionality. MIT Press, 1988.