AN ONTOLOGY FOR LINEAR SPATIAL REASONING

F.Coenen, B. Beattie, T.J.M.Bench-Capon, M.J.R.Shave and B.M.Diaz

Department of Computer Science, The University of Liverpool, Chadwick Building, P.O. Box 147, Liverpool L69 3BX, England. email: frans@uk.ac.liverpool.compsci Tel: 0151 794 3698 Fax: 0151 794 3715

Abstract. An ontology for spatial reasoning based on a tesseral representation of space is presented. The principal advantage offered is that the representation has the effect of linearising multi-dimensional space while still supporting translation through the space in any direction and through any number of dimensions. Consequently, all multi-dimensional spatial reasoning can be implemented using one dimensional (temporal) reasoning techniques. As a result, many of the concerns associated with conventional multi-dimensional spatial reasoning systems, based on more traditional representations, no longer apply.
KEYWORDS: Multi-dimensional spatial reasoning, Ontology

1 INTRODUCTION

An ontology is presented for multi-dimensional spatial reasoning based on an addressing system that has the effect of linearising space (regardless of the number of dimensions under consideration). As a result all multi-dimensional reasoning can be implemented in terms of a single dimension and hence many of the concerns associated with more traditional forms of multi-dimensional reasoning no longer apply.
There are many methods whereby the linearisation of space can be achieved, particularly in the field of computer graphics. Generally the *object space*, the two, three or four dimensional space in which the spatial objects we wish to reason about exist, is *tessellated* (divided up) into a set of isohedral (same shape) sub-spaces down to some predetermined resolution. Each sub-space has the same number of dimensions as the original object space: in two dimensions the sub-space is referred to as a *tile*, and in three dimensions as a *cube*, etc. Every sub-space is allocated a unique numeric address following some sequence (based on either a Cartesian conceptualisation of space or ideas concerning the hierarchical decomposition of space) so that the entire space can be "unravelled" to form a linear sequence of addresses. Spatial objects can then be represented as sets of addresses and related by comparing individual addresses or sequences of addresses using less than, equals to and greater than operators. We can also perform simple arithmetic to (say) determine the immediate predecessor or successor of an address or to *translate* backwards and forwards through the sequence. Further,

by regarding addresses as complex numbers, we can use "complex" arithmetic to translate through the object space in any direction and through any number of dimensions, and rotate and scale objects with respect to a single axis, in a manner that obviates recourse to computationally expensive matrix or trigonometric techniques.
The ontology is presented in a series of tables. The syntax used is based on the MIR knowledge representation langauge which was developed as part of the MAKE (Maintenance Assistance for Knowledge Engineers) project [?]. This was a collaborative project between ICL, British Coal and the University of Liverpool, that addressed the role of maintenance in knowledge based systems. The result was the Make Authoring and Development Environment (MADE) which encouraged the production of systems which can be maintained through an intermediate representation - the Make Intermediate Representation (MIR). MIR is essentially a simple language to create ontologies describing objects and rules arranged in hierarchies which can then be translated into a target executable representation. The original target was a language called Compiled MIR (CMIR), but could equally well be any other executable representation. As such MIR has similarities with other tools for creating ontologies such as Ontolingua [?].

2 ADDRESSES

The primitive unit of the system is the (tesseral) *address* which can be regarded as a unique label associated with some two, three or four dimensional sub-space. The class hierarchy associated with addresses is presented in Table 1. An address has some integer value that lies within a minimum and maximum defined by the addressing mechanism adopted. We then define an arithmetic for the addressing system (based on complex number theory) to implement translation through an object space using complex number addition and subtraction. Similarly we define rotation and scaling (with respect to one axis) using complex number multiplication and division.

Table 1: Address ontology

```
CLASS Address SUPERCLASS TopClass {
"The primitive unit of the system"
    slot address integer range (Min .. Max);
    method +(Address, Address -> Address);
    method -(Address, Address -> Address);
    method *(Address, Address -> Address);
    method /(Address, Address -> Address);
    }

CLASS Sequence SUPERCLASS Address {
"Data structure used to store linear sequences of addresses"
    slot sequence (Address.address .. Address.address);
    }
```

```
CLASS SetAddresses SUPERCLASS Sequence {
"Data structure used to store set of addresses defining the shape
       and/or location of spatial objects"
    slot setAddresses list (NULL, [Address.address|SetAddresses],
       [Sequence.sequence|SetAddresses]);
    method subset(setAddresses, setAddresses -> Boolean);
    method notSubset(setAddresses, setAddresses -> Boolean);
    method equals(setAddresses, setAddresses -> Boolean);
    method notEquals(setAddresses, setAddresses -> Boolean);
    method intersection(setAddresses, setAddresses -> setAddresses);
    method union(setAddresses, setAddresses -> setAddresses);
    method complement(setAddresses, setAddresses -> setAddresses);
    method box(Address.address, Address.address -> setAddresses);
    method line(Address.address, Address.address -> setAddresses);
    method circle(Address.address, Address.address -> setAddresses);
    }

CLASS ObjectSpace SUPERCLASS TopClass {
"The multi-dimensional space in which the spatial objects of interest
       are contained"
    slot minAddress integer (Min .. Max);
    slot maxAddress integer (minAddress .. Max);
    slot objectSpace Sequence (minAddress .. maxAddress);
    }
```

Numeric sequences of addresses can be expressed simply using a “range” operator, such as `'..'`. This mechanism facilitates data compression to reduce the storage requirements associated with objects, and computationally-inexpensive comparison of sets of addresses. For example, with respect to the latter, to determine whether the set of addresses {`p..q`} is contained within the set {`r..t`} we simply compare the start and end addresses of the two ranges, rather than considering all individual addresses (elements) involved in turn. Object spaces are defined in terms of a sequence of addresses ranging from some minimum to some maximum address.
A set of addresses is then a list of zero (`NULL`), one or more elements where each element can be either a single tesseral address or a sequence of addresses (in the tables the `'|'` operator should be read as a traditional (Prolog) head-tail operator). There are a number of operations that can be performed on pairs of sets of addresses, namely the following Boolean operations:

$$\text{X} \subset \text{Y (subset) X} \not\subset \text{Y (notSubset)}$$
$$\text{X} = \text{Y (equals) X} \neq \text{(notEquals)}$$

and the functions:

$$\text{X} \cup \text{Y (union)} \qquad \text{X} \cap \text{Y (intersection)}$$
$$\text{X} \setminus \text{Y (complement)}$$

The superset Boolean operation is not included because this can simply be expressed as a subset relation with the operands transposed. Similarly we do not

differentiate between a subset and a “proper subset” (or a superset and a “proper superset”) because, in spatial reasoning terms, we have not found an application where the distinction is significant. Each operation is defined in terms of a method associated with the `SetAddresses` class.
To allow the user to express simply sets of addresses it has also been found useful to provide a facility to define sets of addresses in terms of geometric shapes. Some two dimensional examples are given in Table 1:

1. *Box.* A box of addresses defined in terms of (say) its “south-west” and “north-east” corner addresses.
2. *Line.* A line of addresses defined in terms of its start and end addresses.
3. *Circle.* A circle of addresses defined in terms of its centre address and its radius defined in addresse units.

These can be extended to three dimensions by including shapes such as *sphere* and *cube.*

3 SPATIAL OBJECTS

The ontology associated with spatial objects is presented in Table 2. Spatial objects have a number of attributes associated with them:

Table 2: Object ontology

```
CLASS SpatialObject SUPERCLASS TopClass {
"A spatial object is any object contained in the object space that we
        might wish to reason about"
    slot objectName string;
    slot location SetAddresses;
    slot ref Address;
    slot locationList list (NULL, [[location,ref]|locationList]);
    slot locationListPair list (NULL, [locationList, locationList]);
    }

CLASS FixedObject SUPERCLASS SpatialObject {
"A fixed object is a spatial object with a fixed location, and
        consequently a definite shape"
    method solveBoolean(locationListPair,
        booleanConstraint.definition -> locationListPair);
    }

CLASS FreeObject SUPERCLASS SpatialObject {
"A free object is a spatial object which has no fixed location
        associated with it"
    slot locationSpace SetAddresses;
    }

CLASS ShapedObject SUPERCLASS FreeObject {
"A shaped object is a spatial object that has a definite shape
```

```
      associated with it, but an unknown location"
   slot shape SetAddresses;
   slot ref Address (cardinality many);
   method candidateLocations(shape, locationSpace -> locationList);
   method solveBoolean(locationListPair, booleanConstraint.definition ->
      locationListPair);
   }

CLASS ShapelessObject SUPERCLASS SpatialObject {
"A shapeless object is a spatial object that has an undefined (or only
      partialy defined) shape and an unknown location"
   slot minSize integer range (ObjectSpace.minAddress ..
      ObjectSpace.maxAddress);
   slot maxSize integer range (ObjectSpace.minAddress..
      ObjectSpace.maxAddress);
   slot contiguityFlag boolean values (true, false);
   method solveFunctional(locationListPair,
      functionConstraint.definition -> locationListPair);
   }
```

1. NAME. The most obvious attribute that a spatial object must have is a label which allows the object to be identified and which ties all the other attributes associated with it together under a single unifying name.
2. LOCATION. An object can either be *fixed* in that it has a "fixed" location associated with it, or it may be *free* in that it has some *location space* (which might extend to the entire object space) associated with it in which the object may be contained. The classes `FixedObject` and `FreeObject` are thus sub-classes of the class `SpatialObject`. Both locations and locations spaces are expressed in terms of sets of addresses (see Table 1).
3. SHAPE. An object can either be *shaped* in that it has a definite shape associated with it (again defined in terms of a set of addresses), or *shapeless*. A fixed object, by definition, must have a "shape" associated with it although there is no need for this to be specifically defined. However, a free object may be either a shaped or a shapeless object. The classes `ShapedObject` and `ShapelessObject` are thus a sub-class of the `FreeObject` class. A shaped object may be "fitted" into its associated location space in a precise number of ways. We say the object has a number of candidate locations associated with it. The `ShapedObject` class includes a class method to calculate this. If such an object has no candidate locations then it cannot be physically realised.
4. REF ADDRESS. When objects are manipulated (translated, rotated etc.) this must be done with respect to some reference address associated with the object. Similarly the shape of an object must be defined with respect to some reference address (such as the south-west corner address of the minimum bounding box surrounding the shape definition). Thus the `spatialObject` class also has an attribute `ref` associated with it.
5. MINIMUM AND MAXIMUM SIZE, and CONTIGUITY. With respect to

shapeless objects there are some additional attributes which the object may take and which serve to partially define the shape of an object. Such attributes include a minimum and/or maximum size in terms of addresses, or a requirement that it is contiguous. In the case of shaped objects these attributes are of course implied. These have been included in the `Shapeless Object` class definition.

Attributes such as location, location space and shape must either be extracted from a database or supplied by a user. In the case of the latter, to facilitate user interaction a scripting language has been developed (together with an appropriate lexical analyser and parser) in which users can define the desired objects without requiring any familiarity with the particular addressing systems used. Experiments have indicated that the use of the Cartesian coordinate system is still the approach which users are most familiar with. Further, so that users do not have to identify every address that forms part of a spatial object it has been found that use of a number of class methods (such as the square, line and circle methods associated with the `SetAddresses` class) to identify groups of addresses substantially simplifies the definition task.
The locations associated with objects are stored in location lists (defined in the `SpatialObject` class) where each location is linked to a reference address. In the case of a fixed object the location list will consist of only one location-reference pair, in the case of a shaped object the list will comprise one or more candidate location-reference pairs, otherwise the list will comprise a single location space definition and an arbitrary reference address.
Individual locations associated with particular objects can be related through the seven relationships presented in Section 2. The relations expressed in this manner are then referred to as constraints. Boolean relations are only meaningful with respect to shaped objects, while the functions are only appropriate where the prefix operand is a shapeless object. Both types of constraint are solved taking a pair of location lists and a constraint definition and returning a revised pair of location lists that satisfy the constraint. If the constraint cannot be satisfied `NULL` will be returned. The definition of constraints is discussed in the following Section.

4 CONSTRAINTS

Given a set of objects defined as described in Section 3 and using the relations given in section 2 we can define both boolean and functional constraints to express the relationships which we require certain pairs of objects to display. For example we can express the desire that a location associated with some shaped object is located wholly within a location associated with some other object (or not), or whether the location is identical to the location of some other object (or not). Alternatively we can cause a location associated with some object to become a complement (not a subset), intersection (subset) or union with respect to some other location. However, for spatial reasoning purposes this is not enough; we also want to be able to define predicates such as “is X to the north west of Y?" or "cause X to be to the north-west of Y". To do this we must apply an appropriate *offset* to the location concerned, `Y` in this case. This can be applied in two ways:

1. To the reference associated with an object's location in a location list (see `SaptialObject` class definition in table 2).
2. To all the addresses defining a particular location associated with an object.

(By application we mean addition or subtraction of addresses as defined in Section 2 to achieve the appropriate translation). The first of the above allows us to define spaces (say) within another object or at some location outside an object. The second allows us to expand an object in all or some direction(s). In this manner all the standard multi-dimensional topological relations such as those identified by (say) Egenhofer [?], Hernández [?] or Cohn [?] can be defined. Of course it is only appropriate to apply such offsets to shaped objects - applying offsets to shapeless objects will result in unpredictable conclusions.
Thus a constraint definition has an operator and two operands plus possibly two offsets to be applied to the operands with the proviso that the operand in question must indicate a shaped object. The ontology associated with constraints is given in Table 3.

Table 3: Constraint ontology

```
CLASS Operand SUPERCLASS TopClass {
"An operand is the prefix or postfix operand associated with a relation operator"
    slot operandValue setAddresses;
    }

CLASS ShapelessOperand SUPERCLASS Operand {
"An operand associated with a shapeless object"
    slot operandDefinition list (["shapeless", spatialObject.objectName]) ;
    method operand(fixedObject.location -¿ operandValue);
    }

CLASS ShapedOperand SUPERCLASS Operand {
"An operand associated with a shaped object"
    slot offset SetAddresses;
    slot type string ("offset", "reference");
    slot operandDefinition list ([type, spatialObject.objectName, offset]);
    }

CLASS OffsetOperand SUPERCLASS ShapedOperand {
"An operand associated with a shaped object that has been generated by applying
        an offset to all the addresses associated with a particular location"
    slot type string ("offset");
    method operandOffset(fixedObject.location, offset -¿ operandValue);
    }

CLASS ReferenceOperand SUPERCLASS ShapedOperand {
"An operand associated with a shaped object that has been generated by applying
```

```
        an offset to a reference address associated with a particular location"
    slot type string ("reference");
    method operandRefOffset(fixedObject.reference, offset -¿ operandValue);
    }

CLASS Constraint SUPERCLASS TopClass {
"A constraint comprises a relation operator and two operands generated from the
        locations associated with the spatial objects we wish to relate"
    slot operand Operand, cardinality 2;
    }

CLASS BooleanConstraint SUPERCLASS constraint {
"A constraint that evaluates to true or false with the effect that the locations
        associated with the spatial objects of interest are valid or invalid.
        Can only be used to relate shaped objects"
    slot operator string values (subset, notSubset, equals, notEquals);
    slot definition list ([ShapedOperand.operandDefinition, operator,
        ShapedOperand.operandDefinition]);
    }

CLASS FunctionalConstraint SUPERCLASS constraint {
"A constraint that relates a shapeless object to a shaped object with the purpose
        of redefining the location/shape of the shapeless object."
    slot operator string values (intersection, complement, union);
    slot definition list ([ShapelessOperand.operandDefinition, operator,
        ShapedOperand.operandDefinition]);
    }
```

5 CONSTRAINT SATISFACTION

Spatial problems couched in terms of sets of object declarations and constraints as described here can more generally be defined as comprising a set of variables that can take values from a finite set of domains, and a set of constraints which specify which values are compatible with each other. A solution to such a problem is thus an assignment of values to all variables which satisfy all constraints. Given that the domains under consideration are finite, and assuming that the given constraints are satisfiable, a solution (or solutions) is always possible. The real problem associated with the satisfaction of such problems is that of complexity (they are NP-complete) and, as a result, efficiency.
There are two broad techniques that can be used to solve spatial problems couched in terms of constraints. One approach is to generate all possible assignments of values to variables and test these against the given set of constraints, in a manner that will result in early identification of fruitless assignments (see [**?**] for further discussion). An alternative approach is to use constraint propagation techniques where variables range over a set of values and the constraints are used to prune this set of values (see [**?**] and [**?**] for further discussion).

Current implementations of the system described here adopt this second approach. This requires a constraint selection strategy and some mechanism for storing results. The current implementations store all partial solutions and end solutions in a solution tree structure, each node of which represents a solution "state" after one or more constraints have been satisfied. Branches in the tree indicate nodes where, on satisfaction of a constraint, more than one solution has been produced. The current constraint selection strategy is geared to limiting the growth of this solution tree and the early identification of fruitless branches.

6 CONCLUSIONS

In this paper we have presented an ontology for spatial reasoning based on a tesseral representation of space. The advantages gained (over more traditional representations) can be summarised as follows:

1. Linearisation of space so that all multi-dimensional spatial reasoning need take place in only one-dimension while at the same time supporting the description of complex multi-dimensional shapes.
2. Simple manipulation of groups of addresses making full use of the sequencing of addresses and the complex arithmetic that can be superimposed over the representation.

As a result many of the concerns associated with traditional approaches to multi-dimensional reasoning are no longer relevant. The system has been implemented using a quad-tesseral representation based on the hierarchical decomposition of space (see [?]) and a Cartesian "grid-referencing" system. The ontolog has been used in demonstration systems which address a number of application areas: examples include support for environmental impact assessment with respect to building projects [?], provision of spatial reasoning capabilities for Geographic Information Systems [?], and to address timetabling problems [?]. Earlier versions of the system operated using only one dimension and were used to address temporal reasoning problems (see [?]).

References

1. B. Beattie, F.P. Coenen, A. Hough, T.J.M. Bench-Capon, B. Diaz and M.J.R. Shave. 'Spatial Reasoning for Environmental Impact Assessment', to be presented at Third International Conference/Workshop on Integrating GIS and Environmental Modelling, Santa Fé, 1996.
2. B. Beattie, F.P. Coenen, T.J.M. Bench-Capon, B. Diaz and M.J.R. Shave, 'Spatial Reasoning for GIS using a Tesseral Data Representation', in N. Revell and A.M. Tjoa (eds.), *Database and Expert Systems Applications, (Proceedings DEXA'95), Lecture Notes in Computer Science 978*, Springer Verlag, 207-216, 1995.
3. F.P. Coenen, B. Beattie, T.J.M. Bench-Capon, Shave, M.J.R and B. Diaz, 'Spatial Reasoning for Timetabling: The TIMETABLER system', *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT'95)*, Napier University, Edinburgh, 57-68, 1995.

4. F.P. Coenen, B. Beattie, B. Diaz, T.J.M. Bench-Capon and M.J.R. Shave, 'A Temporal Calculus for GIS Using Tesseral Addressing', in M.A. Bramer and A.L. Macintosh (eds), *Research and Development in Expert Systems XI, Proceedings of ES'94*, 261-273, 1994
5. F.P. Coenen and T.J.M. Bench-Capon, 'Maintenance and Maintainability in Regulation Based KBS', *ICL Technical Journal*, **9**-3, May, 67-84, 1992.
6. A.G. Cohn, 'A More Expressive Formulation of Many Sorted Logic', *Jo of Automation and Reasoning*, **3**-2, 113-200, 1987.
7. B. Diaz and S.B.M. Bell, *Spatial Data Processing Using Tesseral Methods*, Natural Environment Research Council publication, Swindon, England, 1986
8. M.J. Egenhofer, 'Deriving the Composition of Binary Topological Relations', *Journal of Visual Languages and Computing*, **5**, 133-149, 1994.
9. T.R. Gruber, 'Ontolingua: A Mechanism to Support Portable Ontologies', *Technical Report KSL 91-66*, Stanford University, Knowledge Systems Laboratory, Stanford, USA, 1992.
10. P. van Hentenryck, 'Constraint Satisfaction in Logic Programming', MIT Press, Cambridge, Massachusetts, 1989.
11. D. Hernández, 'Relative Representation of Spatial Knowledge: The 2-D Case' in D.M. Mark and A.U. Frank, A.U. (eds), *Cognitive and Linguistic Aspects of Geographic Space*, Kluwer, Dordrecht, Netherlands, 373-385, 1991.
12. A.K. Mackworth, 'Consistency in Networks of Relations', AI Journal, **8**-1, 99-118, 1977.
13. A.K. Mackworth and E.C. Freuder. 'The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems' *Artificial Intelligence*, **25**, 65-74, 1985

This article was processed using the LaTeX macro package with LLNCS style