Intelligent Java Objects

Hans Diel

Diel Software Beratung und Entwicklung, Seestr. 102, 71067 Sindelfingen, Germany,
diel@s.netic.de

Abstract. Intelligent Java Objects provides java classes in support of a variety of artificial intelligence functions. Business objects such as people, products, projects are contained in Object Sets on which predicates can be applied. The predicates can be combined in a way similar to Logic Programming. This enables applications in support of expert systems, data mining, plan determination and classification.
The functions provided are offered in form of a framework which can easily be applied to existing data (bases) and can be integrated with existing applications.
Due to the very dynamic nature of the intelligence functions, the power of Java Beans was required for a satisfying implementation.

1 Introduction

Applying (artificial) intelligence to computer programs has made very much progress during the past 10 years, not only with the 'classical' AI disciplines (expert systems, logical reasoning, fuzzy logic, neural networks, etc.), but also with a newer area which is described by the term 'Business Intelligence' (see [8]). The latter includes technologies such as data mining, decision support, and OLAP (online analytic processing, see [1]). Intelligent Java Objects aim at the same direction. However rather than providing a single product, Intelligent Java Objects provides

1. A multitude of "intelligence functions" in an integrated way. Thereby, emphasis is put on the breadth of function set, while the depth (i.e. power) of the individual functions may be limited.
2. The intelligence functions are offered as part of an object oriented framework, conforming to Java Beans, which can easily be combined with existing software for handling business objects.

The functions provided are build on BOSets (Business Object Sets). BOSets are collections of objects representing the business data (e.g. people, projects, products, financial data) being considered. Another important concept for the Intelligent Java Objects are Predicates. Most of the "intelligence functions" of Intelligent Java Objects are defined in terms of predicates. Predicates can be combined in expressions similar as with Logic Programming (see [3], [4]). They can be applied to the BOSets. The predicates are defined in a textual declarative specification.

2 BOSets, Business Object Sets

BOSets are the base elements on which the intelligence functions are applied. In addition, also the results of the intelligence functions are stored in BOSets. This allows for a repetitive and recursive application of the intelligence functions.

Prior to introducing the BOSets, the terms Business Object and Collection of Business Objects are explained below from the point of view of Intelligent Java Objects. A Business Object is a Java Bean representing business data such as person, project, product, financial data, asset.

BO Collections. Business Objects can be grouped in collections. The Java Collection API (see [2]) is used for this purpose.

BOSet. BOSets are sets of BO Collections. In addition, special methods for support of intelligence functions are provided as methods of the class BOSet.

A single BOSet containing all the required (collections of) business objects is the starting point for applying the Intelligent Java Objects functions. The results of applying an Intelligent Java Objects function is again a BOSet (and possibly further information). The results may be visualized or used for further processing.

The following methods are available with BOSets

- BOSet (decl-spec)
- BOSet (decl-spec, bo-collections)
- addPredicate (predicate)
- addBOs (class-name, bo-collection)
- classify (object)
- contentsView ()
- copy ()
- evaluate (predicate)
- getCorrelationDescription ()
- getTaskEvaluationDescription ()
- getFirst ()
- getNext ()
- subset (class-name)
- tupleView ()

Instantiation of BOSet. BOSets can be instantiated either directly by the user program or by Intelligent Java Objects. In order to enable Intelligent Java Objects the instantiation of Business Objects and BOSets, the user of Intelligent Java Objects has to provide for each type of business object participating in a Intelligent Java Objects session two kinds of classes

1. The Java Bean representing the business object.
2. A data access class which creates BO Collections as a result of its methods "select" and "retrieve". For data base access a generic data access class "DBAccess" is provided by the Intelligent Java Objects framework.

3 Intelligence Functions

The various intelligence functions are offered in terms of different predicate types. Predicates are specified in a textual declarative specification. The general form of a Predicate statement is:

```
PREDICATE  p-Name(invocation-Parms) = type type-specific-parameters
```

Besides the PREDICATE statement, further related statements, such as CONSTRAINT, TASK, and SUBCLASS are supported and described below.

Selection. Selection of business objects according to specified attribute values is supported by this type of predicate.

```
PREDICATE p-Name ( invocation-Parms ) = SELECTION selection-Spec
```

Selection-spec is compatible with a SQL selection clause (see [6],[7]).
Examples:

```
PREDICATE  P1 ( Person p ) = SELECT department  =  '3265'
PREDICATE  P2 ( Room r   ) = SELECT bath = 'Yes' AND price >150
```

Relation. Determines the business object pairs which satisfy a specified relation.

```
PREDICATE p-Name(invocation-Parms) = RELATION attribute = attribute
```

Example:

```
PREDICATE  P3 (Room r, Person p)= RELATION r.owner = p.name
```

Correlation. In support of data-mining-like applications (see [9]), the Correlation predicate determines whether a certain correlation exists between attributes of business objects. It supports numerical attributes, as well as enumerated value sets.

```
PREDICATE p-Name ( invocation-Parms ) =
                        CORRELATION attribute WITH attribute
```

Examples:

```
PREDICATE  P5 ( Room r ) = CORRELATION   rnumber  WITH   phone
PREDICATE  P6 ( Room r ) = CORRELATION   view     WITH   price
PREDICATE  P7 ( Room r ) = CORRELATION   price    WITH   *
```

Using * as attribute specification determines all possible correlations. For example, P7 finds out all the possible attributes which are correlated with room.price.

Fuzzy Logic. The Fuzzy predicate allows the specification of a (yet) simplified fuzzy logic condition (see [5]).

```
PREDICATE p-Name ( invocation-Parms ) = FUZZY  fuzzy-Spec ...
```

Examples:

```
PREDICATE    MorbusCrohnConstraint( Disease d ) =
   FUZZY    fever 3,  pain 9,  diarrhoea1 9,  diarrhoea2 0,
            skin 5,  eyes 5,  arthritis 5,  liver 5,  fistula 2

PREDICATE    ColitisConstraint( Disease d ) =
   FUZZY    fever 3,  pain 0,  diarrhoea1 0,  diarrhoea2 0,
            skin 1,  eyes 1,  arthritis 5,  liver 1,  fistula 1
```

User-provided Predicate. Complex intelligence functions can be provided by user-defined subclasses.

```
PREDICATE p-Name(invocation-Parms) = USERDEFINED (argument-List)
```

Example:

```
PREDICATE MoveAction(THanoi t, String s) = USERDEFINED(t, s)
```

The user-defined predicate may also be used to invoke non-predicate functions returning a BOSet.

Logical Expression. Logical expressions support the combination of the above described predicates by use of AND and OR operations. The logical expressions supported by Intelligent Java Objects have much commonolity with Logic Programming (e.g. Prolog, see [12]).), however they apply to business objects.

```
PREDICATE p-Name ( invocation-Parms ) = LogicalExpression
```

LogicalExpression is constructed by use of AND operators, OR operators, and Predicate references:

```
  LogicalExpression::= predicate-Call | logExpr1  |  logExpr2
  logExpr1         ::= LogicalExpression logOperator LogicalExpression
  logOperator      ::= AND | OR
  predicate-Call   ::= [NOT] predicate-Ref
  logExpr2         ::= [NOT] ( LogicalExpression )
```

The following example shows a Logical Expression and also some forms of predicate references.

```
PREDICATE P4(Room x, Person y) = ( P3(x, y) OR PP( PPP(x), x) )
               OR ( P0( y ) AND NOT ( P1( y ) OR P2( x )))
```

SUBCLASSES Statement. In order to implement a classification function the SUBCLASSES statement may be used:

```
SUBCLASSES ( BO-Class ) = subclass ...
```

BO-Class is the name of the Business Object class whose subclasses are specified. subclass is the name of a subclass of BO-Class.

Example:

```
SUBCLASSES(Disease) = Morbus-Crohn Colitis-Ulcerosa Colon-Irritabile
```

CONSTRAINT Statement. Constraints for a Business Object class are specified by the Constraint statement:

```
CONSTRAINT ( BO-Class ) = predicate-Ref
```

BO-Class is the name of the Business Object class whose constraints are specified. Example for a classification specification by use of Subclasses, Constraint, and Predicate statements:

```
SUBCLASSES(Disease) = Influenza Morbus-Crohn Colitis-Ulcerosa
                      Colon-Irritabile
CONSTRAINT ( Influenza ) = InfluenzaConstraint
PREDICATE  InfluenzaConstraint( Disease d ) =
           FUZZY fever 10, cough 7, catarrh 5,  pain 3, gastroib 3
```

TASK Statement. Task statements are used to specify tasks for which a plan is to be determined. Besides the general Plan Determination case, two special tasks, namely Allocation and Routing are supported.

General Plan Determination. In order to determine a plan (i.e. the sequence of actions) for performing a task, which is described in terms of a goal predicate and a set of allowed rules, use the Task statement:

```
TASK p-Name(invocation-Parms) = GOAL predicate-Ref
                                RULES predicate-Ref ...
```

Example:

```
TASK  THanoiPlan(THanoi t) = GOAL AllAt3(t) RULES Move(t)
PREDICATE  Move(Cond12( THanoi t ) ) =  MoveAction( t, "12" )
PREDICATE  Move(Cond13( THanoi t ) ) =  MoveAction( t, "13" )
    ....
PREDICATE  Cond12(THanoi t) = SELECT t.stack2h > t.stack1h
PREDICATE  Cond13(THanoi t) = SELECT t.stack3h > t.stack1h
    ....
PREDICATE  AllAt3(THanoi t) = SELECT t.stack32 = '3'
                                 OR t.stack22 = '3'
```

The Move predicates also show the usage of predicates on the left-hand side of the = sign (i.e. in the clause head).

Routing. Determines the sequence of business objects with the minimal total distance (e.g. travelling salesman problem).

```
TASK p-Name ( invocation-Parms ) = GOAL MIN_DISTANCE ( x, y )
```

Example:

```
TASK Routing(City c) = GOAL MIN_DISTANCE(xCoordinate, yCoordinate)
```

Allocation. Allocates one set business objects (e.g. people) to a second set of business objects (e.g. rooms) by satisfying given rules.

```
TASK p-Name(invocation-Parms) = GOAL ALLOCATION(bo-Class, bo-Class)
                                     RULES  predicate-Ref ...
```

Example:

```
TASK   ALLOC ( Person p, Room r ) =  GOAL   ALLOCATION( p, r )
                  RULES R1( p, r )    R2( p ,r )    R3( p, r)

PREDICATE   R1 ( Person p1, Room r1 ) = PA1( r1 ) OR PA2( p1 )
PREDICATE   R2 ( Person p1, Room r1 ) = PA3( r1 ) OR PA4( p1 )
PREDICATE   R3 ( Person p1, Room r1 ) = PA5( r1 ) OR PA6( p1 )
PREDICATE   PA1( Room r2 )   = SELECT   bath  =  'Yes'
PREDICATE   PA3( Room r2 )   = SELECT   view =   'vineyard'
PREDICATE   PA5( Room r2 )   = SELECT   price  =  150
PREDICATE   PA2( Person p2 ) = SELECT   sex  =  'female'
PREDICATE   PA4( Person p2 ) = SELECT   department  = '3265'
PREDICATE   PA6( Person p2 ) = SELECT   areaCode = '07031'
```

4 Declarative Specification of Intelligent Java Objects

The Declarative Specification of Intelligent Java Objects defines the business objects involved in a Intelligent Java Objects session and optionally predicate, task, subclass, and constraint statements.

Rather than defining the detailed syntax of the Intelligent Java Objects Declarative Specification, an example containing the major statement types is shown below.

```
BEGIN
BOSET = Person
DATAACCESS( Person ) = DBAccess ( PeopleDB, admin, pwxxx )
PREDICATE loves ( Person x, Person "Maria" ) = mad( x )
                              AND rich( x ) AND "Maria"
PREDICATE loves ( Person x, Person "Alice" ) = mad( x )
                              AND hatter( x ) AND "Alice"
PREDICATE loves ( Person x, Person y )  = rich( x ) AND poor( y )
```

```
PREDICATE hatter ( Person p ) = SELECT  attr1 = 'hatter'
PREDICATE mad ( Person z ) = logician( z )
PREDICATE logician ( Person p ) = SELECT attr2 = 'logician'
PREDICATE authors ( Person p ) = SELECT profession = 'author'
PREDICATE rich ( Person x ) = SELECT wealth > 1000000
PREDICATE poor ( Person y ) = SELECT wealth < 5000
PREDICATE WhichAuthorLovesWhom( Person y1, Person y2 ) =
                            loves( y1, y2 ) AND authors( y1 )
END
```

5 Results of Predicate Evaluation

Intelligence functions are, in general, invoked by applying the evaluate(predicate) method to a BOSet. The result of such an invocation depends on the type of predicate (see above). It may include:

- a boolean variable indicating whether the predicate is satisfied
- a probability value giving the probability for the predicate satisfaction (e.g. with fuzzy logic)
- the (sub-) set of objects which satisfy the given predicate
- the set of object tuples which satisfy the given predicate (for example, a Relation predicate)
- a ResultDescriptionObject describing the result of special function types. The following types of ResultDescriptionObjects are supported:
  - CorrelationDescription
  - ClassificationDescription
  - TaskEvaluationDescription

6 The Architecture - An extensible Framework

With a running system, the following parts are distinguished:

User Application. The user application contains the logic for setting up the environment and for visualizing the results of the intelligence function invocations.

Business Object. The Business Objects are provided by the user. The classes must support the Java Beans interface, i.e. get and set methods for the attributes, and the getClass, getProperty and getMethod methods.

BOSet. The BOSet represents the main part of the Intelligent Java Objects framework. It provides the linkage between the Business Objects and the intelligence functions and maintains the results of the predicate evaluations.

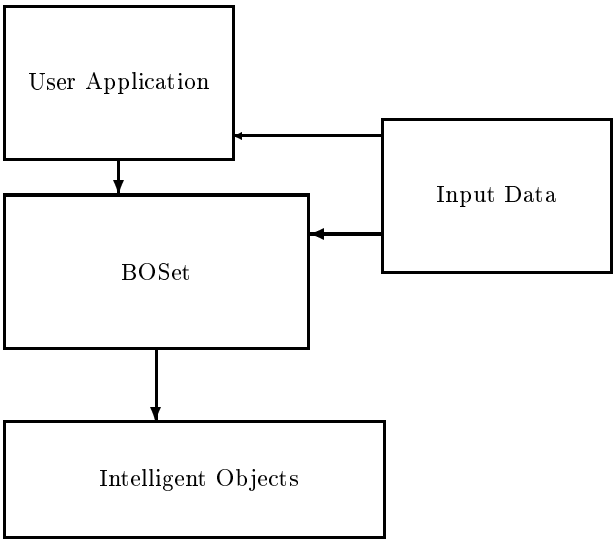

Fig. 1. Structure Of Intelligent Java Objects Framework.

Input Data. The input data being retrieved through the Business Objects can reside in various data sources. If the objects are to be instantiated by Intelligent Java Objects (rather than by the user program), data access classes are to be provided. For data base access, a generic data access class is provided by Intelligent Java Objects. Others can be easily developed with the help of various tools (e.g. IBM Visual Age for Java, see [10],[11]) or may be provided by a certain middleware or by the user application.

Intelligent Objects. In addition to the above described components, Intelligent Java Objects is structured in a way that supports a clean interface to "Intelligent Objects". Intelligent Objects provide the functions of the various predicate types (e.g. selection, correlation, classification). The Intelligent Object Interface can be used to replace the system-provided intelligence functions by user provided classes and to add new user provided intelligence functions and predicate types.

7 Why Java ?

After first implementations in Smalltalk and C++, Intelligent Java Objects has now been implemented in Java. Besides the general Java advantages of being a modern, operating system independent language, the benefits which make Java especially suitable for Intelligent Java Objects are:

1. Intelligent Java Objects requires the dynamic combination of functions whose names are determined at run-time. Java Beans provide this capability better than any other language.

2. Intelligent Java Objects should also be applicable to existing data sources. The Java DB Connection interface in combination with Java Beans is well-suited for this purpose.

8 Summary and Related Work

Intelligent Java Objects provides a framework for support of a multitude of intelligence functions in an integrated way. Thereby, rather than trying to rebuild all the various AI functions emphasis was put on

1. The ability to combine the various function and predicate types to form new more powerful functions. For example, the routing task supports the search for the minimal length path, but can be combined with additional predicates requesting, for example, that city A must be reached before city B. This combination capability is supported (a) by the way the individual predicate types are defined, and (b) by support of the Logical Expression predicate.
2. The ability to replace the system-provided functions by intelligence functions offered somewhere else (for example, by the customer or by some commercial tool).
3. The ability to access existing data and data descriptions. Even the most powerful AI tool is of limited value, if it requires a new format of the input data or new data descriptions. Intelligent Java Objects avoids this problem by utilizing Java Beans.

Related work is mainly seen in three directions:

1. Work in the area of artificial intelligence: Countless avtivities exist in this area. Intelligent Java Objects does not offer any progress in any specific area of artificial intelligence, but tries to offer an integration among the different areas.
2. Work on the combination of Logic Programming and Object Oriented Programming: Quite a number of efforts address this subject (see, for example [13], [14]). The special direction chosen by Intelligent Java Objects is the "set theoretical" approach and the emphasis on Java.
3. Work in the area of business intelligence: In contrast to a growing number of commercially successfull products in this area, Intelligent Java Objects tries to offer a more general, non-proprietary, Java-based framework.

After further completions it is envisaged to offer the Intelligent Java Objects functions at the internet such that it can be downloaded and extended by users.

References

1. Thomsen,E.: OLAP Solutions, John Wiley & Sons, 1999
2. Weber,J.: Using Java 2 Platform, QUE, 1999

3. Levi,G., Rodriguez Artalejo,M.: Algebraic and Logic Programming, Springer, Berlin, 1994
4. Beierle,C., Pluemer,L.: Logic Programming: Formal Methods and Pratical Applications, Elsevier, Amsterdam, 1995
5. MacNeill,D., Freiberger,P.: Fuzzy Logic, Droemer-Knaur, Muenchen, 1994
6. Marsch,J., Fritze.J.: SQL, Vieweg, Braunschweig, 1992
7. Groff,J.R., Weinberg,P.N.: Using SQL, MacGraw-Hill, Berkeley, 1990
8. Dhar,V., Stein,R.: Seven Methods for Transforming Corporate Data into Business Intelligence, Prentice-Hall, Englewood Cliffs, 1997
9. Cabena,P., Hadjinian,P., Stadler,R.: Discovering Data Mining, Prentice-Hall, Upper Saddle River, 1998
10. Carrel-Billiard,M., Akerley,J.: Programming with Visual Age for Java, Prentice-Hall, Englewood Cliffs, 1998
11. Jakab,P., Nilsson,D.P.: Developing Java Beans using Visual Age for Java, Wiley, New York, 1998
12. Clark,K.L., McCabe,F.G.: PROLOG: A language for implementing expert systems, Machine Intelligence 10, 1982
13. Ng,K.W., Luk,C.K.: A survey of languages integrating functional, object oriented and logic programming, Microprocessing and Microprogramming 41, 1995
14. Jenkins,M., Chester,D.: A Combined Object-Oriented and Logic Programming Tool for AI, Proc. of the 1993 IEEE Int.'l Conf. with AI, Boston, 1993