Patrizia Marsura and Dirk Riehle. *Design and Implementation of the Java Any Framework.* Ubilab Technical Report 98-05-01. Zurich, Switzerland: UBS, 1998.

Abstract

This report describes the design and implementation of the Java Any framework, an object-oriented framework for self-describing data types. The framework classes let developers flexibly represent any data structure, from primitive value types to full-fledged object graphs. The framework provides abstractions that cover primitive value types, abstractions that represent common collection types, and abstractions that cover general object types (called frames). Frames come with their meta-description so that users can introduce their own object types.

Table of Contents

1 Introduction

This report describes the design and implementation of the Java Any framework, an object-oriented framework for self-describing data types. The framework classes let developers flexibly represent any data structure, from primitive value types to full-fledged object graphs. Thus, the framework provides abstractions that cover primitive value types, that represent common collection types, and that cover general object types (called frames). Frames come with their meta-description so that users can introduce new object types.

The uses of a generic data representation mechanism are manifold. In the Beyond-Sniff project, the Any framework is used to represent full symbolic information of the software system under development. It is used to integrate the tools of the development environment. It is used for semantic streaming [MB96]. It is for software system configuration, and it is used to manage project descriptions.

The history of the Any framework idea dates back to Andre Weinand, who invented the original framework. It was taken up and refined in the Beyond-Sniff project by Kai-Uwe Mätzel and Walter Bischofberger. Finally, we took it up in the Geo project.

In the Beyond-Sniff project, a C++ version was developed, and in the Geo project, a Java version was developed. Both versions are compatible with respect to the streaming protocol defined by their parsers.

The C++ framework has the same properties as the Java framework, even though its implementation is more complicated (the Body/Handle idiom had to be used to allow sharing of objects and provide customized garbage collection). In addition, it provides an OQL parser and query execution machine.

Both frameworks are available from the Ubilab website.

- www.ubs.com + ubilab + software engineering.

Alternatively, you can retrieve the documents and code from an external website:

- C++ framework: www.riehle.org/SourceIndex.html#AnysCpp
- Java framework: www.riehle.org/SourceIndex.html#AnysJava

We would like to thank our colleague Kai-Uwe Mätzel for supporting us in the design and implementation of the Java Any framework.

Patrizia Marsura, Dirk Riehle. Zurich, Switzerland, 1998.

2 Basic Framework Structure

This chapter gives an overview of the interfaces and classes that make up the framework.

Package *Any* contains the interfaces and package *AnyImpl* the implementation classes.

2.1 Interfaces

The following figure shows the hierarchy of *Any* interfaces:

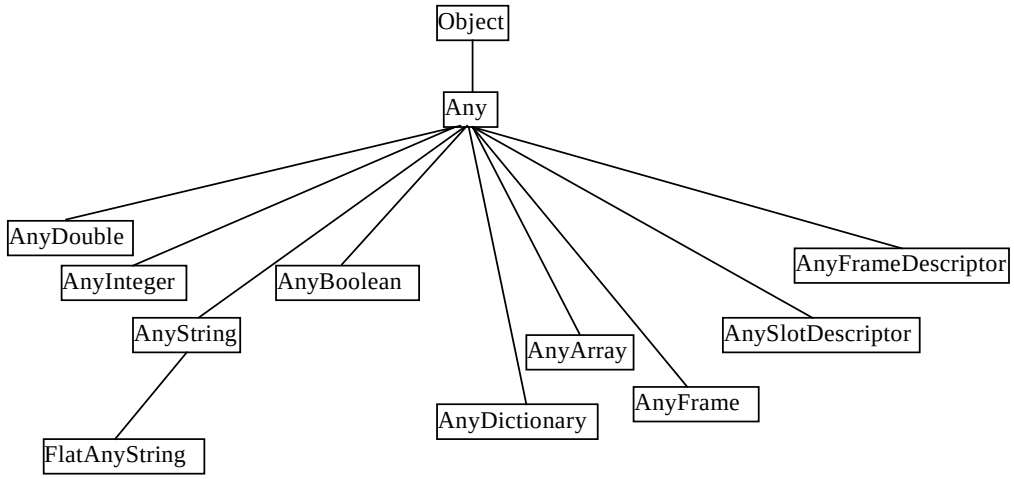

Fig. 1: Any Interfaces

Value Types

AnyDouble, *AnyInteger*, *AnyString* and *AnyBoolean* are used to hold double, int, String and boolean values, respectively. A *FlatAnyString* holds a String containing a streamed out *Any*.

Value type instances are created by using dedicated *AnySap* methods (refer to 6.3).

Object Types

AnyArray represents a flexibly growing array of *Any*s, *AnyDictionary* a collection of key-value pairs, where key is an *AnyString* and value is one or several *Any*(s). *AnyFrame* is used to create instances of user defined types, the type structure being defined by an *AnyFrameDescriptor*. A frame descriptor holds the name and an *AnySlotDescriptor* for each slot. The slot descriptor defines type and properties of a slot.

AnyDictionary and *AnyFrame* have multi-value semantics, refer to 7.8 for more information on how this is implemented. *AnyArray* and *AnySymbolTable* have single-value semantics.

Object type instances are created by instantiating the corresponding implementation class.

The value types and *AnySlotDescriptor* are simple types, the other classes are compound types.

This figure shows the hierachy of other interfaces (non-*Any*) that are part of the framework:

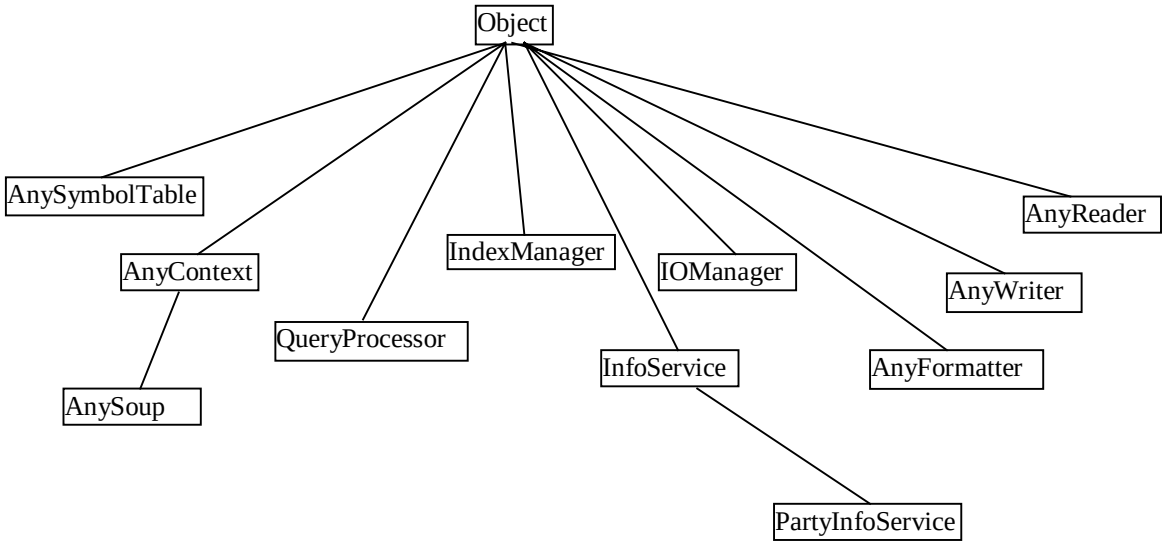

Fig. 2: Other Interfaces

An *AnySymbolTable* manages a collection of *AnyFrameDescriptor*s. *AnyContext* is used to keep a reference to a *AnySymbolTable*. *AnySoup* extends *AnyContext* by managing frames, providing locking, query and IO mechanisms and keeping a reference to an *IndexManager*.

QueryProcessor gives the possibility to evaluate OQL queries over a collection of frames. There is no implementation of a query processor in the current version of the framework.

IndexManager manages frame indices and other user-defined indices. Frame indices keep track of all frames of a given frame descriptor.

InfoService provides general mechanisms to explore and manipulate frames in a given soup. *PartyInfoService* is a demo class to show how a specific service can be implemented. Refer to 1 for more details.

IOManager provides functionality to store and load data from/to a given soup.

AnyWriter writes an Any to a String. *AnyReader* creates a new Any from a String. *AnyFormatter* gives indentation support for formatting.

2.2 Classes

In this paragraph a textual description is only given for classes that do not implement any of the interfaces described in the previous paragraph.

The following diagram shows the hierarchy of *AnyImpl* implementation classes. Exception classes are not included in the diagram, see 6.4 for a list of exceptions.

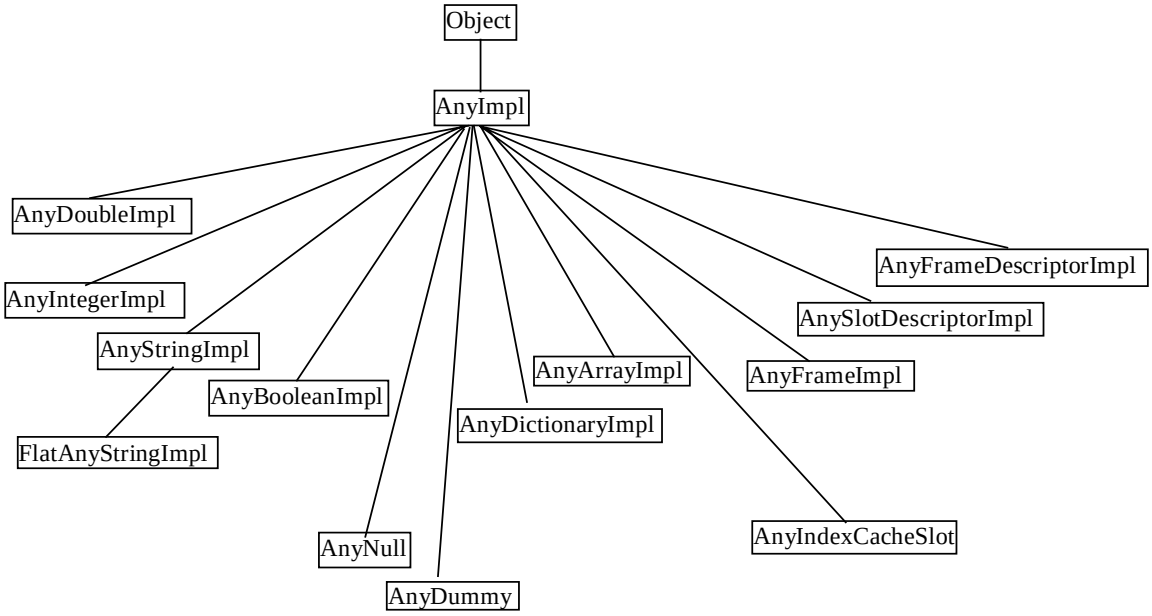

Fig.3: Any Classes

AnyNull is transparently used in componed objects to replace null in order to avoid null pointer exceptions. It is of value type and there is only one instance which can be referenced using *AnySap* (refer to 6.3).

AnyDummy is used as a place holder when reading *Any*s with circular references, refer to 4.4.

AnyIndexCacheSlot is used by *AnyFrameDescriptor* for caching slot names (needs to be subclass of *Any* so that instances can be inserted into an *AnyArray*, as sort capability is needed).

Note on Value Types:

Only two instances (true/false) of *AnyBooleanImpl* exist. For the other value types a new instance is created for every user request, even if two instances have the same value. Refer to 6.1 for the reason why.

This figure shows the hierachy of other classes (non-*AnyImpl*) that are part of the framework:

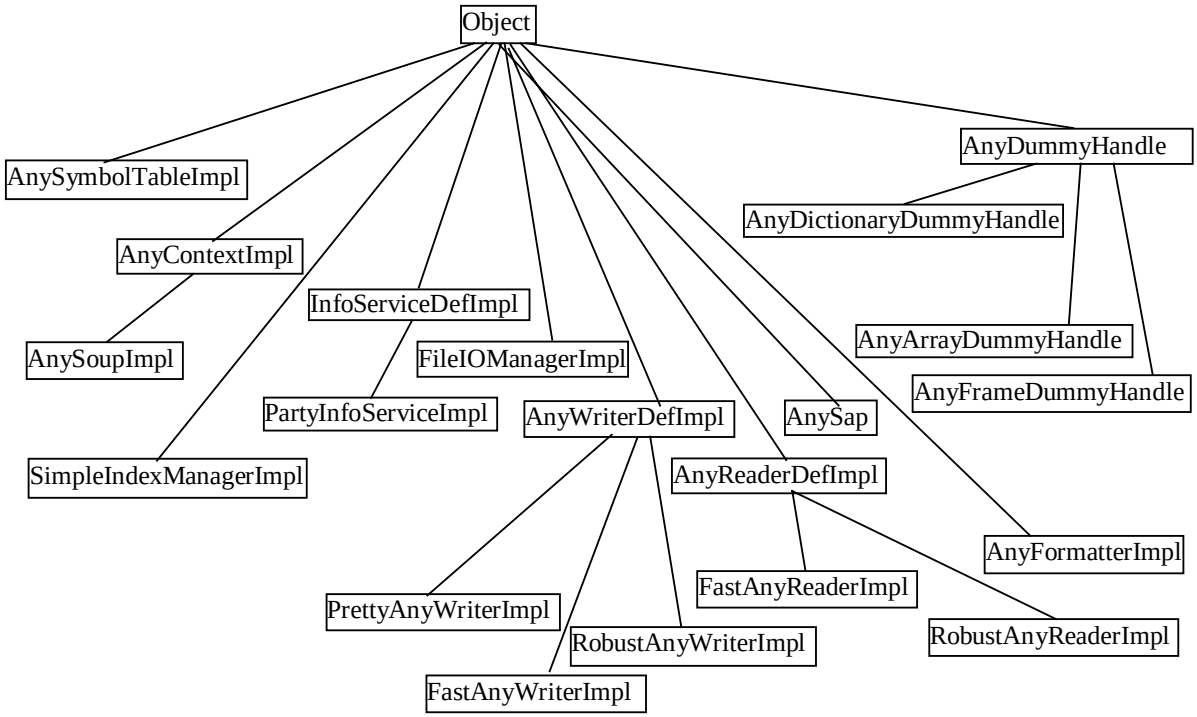

Fig. 4: Any Classes

AnySap: single access point, refer to 6.3.

AnyHandle, *AnyDictionaryDummyHandle*, *AnyArrayDummyHandle* and *AnyFrameDummy-Handle* are used to keep track of *AnyDummy*s that have been used while reading an *Any*. Refer to 4.4 for details on how dummy handles are used.

3 AnyContext and AnySoup

3.1 AnyContext

An *AnyContext* holds an *AnySymbolTable* which is used to manage a collection of *AnyFrameDescriptor*s. A frame descriptor is always created in a specific context and then automatically added to the corresponding symbol table. When creating a new frame the user can either pass the frame descriptor name and a context or directly the frame descriptor itself as parameter.

3.2 AnySoup

AnySoup extends *AnyContext* by keeping track not only of frame descriptors but also of frames. The frames need to be manually added and removed by the user. *AnySoup* also provides locking with commit and rollback, IO and query mechanisms as well as a reference to an *AnyIndexManager*. IO and query requests are forwarded to the IO manager or query processor, respectively. If an index manager has been set for a soup, frames are automatically enrolled when being added to the soup and unrolled when being removed.

The following implementation exist for IOManager and IndexManager, respectively:

FileIOManagerImpl

Data is stored using *RobustAnyWriter*, meta information using *FastAnyWriter*. Writer creates a String which is then written to file. When loading the whole file is read into a String which is then processed by *AnyReader*.

SimleIndexManagerImpl

When being enrolled frames are only added to frame indices, the handling of other (user-defined) indices is completely up to the user.

4 AnyReader and AnyWriter

4.1 Pretty Writer

PrettyAnyWriterImpl produces a human-readable formatted output. As emphasis is on nice output, no corresponding reader is provided. All compound objects are buffered and referenced by a sequence number to reduce lines of output.

AnyFormatter is used for indentation.

4.2 Fast Reader/Writer

FastAnyWriterImpl writes *Any*s in a compact form to maximize speed. All object type objects are buffered. Difference to robust format:

- For *AnyFrame*s: only slot values are written (not slot names), null value slots are written
- For *AnyFrameDescriptor*s: description is not written

4.3 Robust Reader/Writer

Difference to fast format:

For *AnyFrame*s: every slot value is preceded by the slot name, null value slots are omitted

For *AnyFrameDescriptor*s: description is written

4.4 Handling Circular References

When writing an *Any* it is first checked if it has already been written before (i.e. if it is in the buffer). If not, an id is assigned to the *Any*, the *Any* is buffered and the id preceded by an '&' is written after the *Any*. Otherwise the *Any* is not written but the referenced id (as stored in the buffer) preceded by an '*' instead.

When reading an id (&i) the *Any* that has just been read is added to the buffer. If a reference to an id is found (*i) the buffer is searched for an *Any* with that id. For circular references the referenced *Any* will not be in the buffer yet. Therefore an instance of *AnyDummy* is created and referenced instead. Once the whole input string has been processed all *Any*s will be in the

buffer and any *AnyDummy* will need to be replaced with the corresponding *Any*. To be able to do that an *AnyDummyHandle* (*AnyArrayDummyHandle*, *AnyFrameDummyHandle* or *AnyDictionaryDummyHandle*, depending on context referencing the missing *Any*) is created for every *AnyDummy* and kept in a list. The handle keeps the information required to replace the dummy (position for *AnyArrayDummyHandle*, slotName for *AnyFrameDummyHandle*, key for *AnyDictionaryDummyHandle*).

5 Information services

InfoServiceDefImpl is an abstract class that gives the possibility to query or manipulate the contents of a soup. Queries can be done by *Any* kind and slot value, or by specifying OQL queries (OQL processor not implemented yet). Manipulation support includes setting a specific slot or removing *Anys* by OQL query. Note that *InfoServiceDefImpl* does not make use of *IndexManager* because the frame index mechanisms do not provide by kind but only by type queries.

PartyInfoServiceImpl is given as an example on how to implement domain specific information services. It basically just maps domain specific request to the general ones of *InfoServiceDefImpl*.

6 Implementation

6.1 Using Java Garbage Collection

Java's garbage collection makes the implementation of the framework much easier than the C++ version because reference counting is not needed. Still on several occasions not knowing how many references exist to a certain object make things more complicated:

- *AnyWriter* of C++ version only buffers objects to whose reference count is bigger than 1. In the Java version all need to be buffered.
- In the C++ implementation *AnyString*s are treated like Smalltalk strings, i.e. for same value there is only one instance. This was implemented by keeping a string table. This is not possible in Java because without reference counting the last instance can never be removed from the table and *AnyStringImpl*s therefore would never be garbage collected.
- Index managing is under the responsibility of *AnyContext* in the C++ implementation. Constructor/destructor of *AnyFrame* enrolls respectively unrolls frame to/from index. As every index keeps a list of the frames that are part of it, the frames would never get garbage collected in a corresponding Java implementation. Therefore index managing was put under the responsibility of *AnySoup* and the user needs to manually remove frames from the soup. Frames are enrolled/unrolled when being added/removed to/from the soup.

6.2 No Pointers

Having no pointers in Java makes handling circular references in the readers much more complicated than in the C++ framework implementation. In C++ a reference to an *Any* that has not been read just points to the appropriate entry in the buffer which is null yet but will be set later. In Java the problem had to be solved using *AnyDummy* and *AnyDummyHandler* as described in 4.4.

6.3 Using AnySap

AnySap is used for the instantiation of value type objects. The following static methods are provided:

```
public static AnyBoolean booleanValue(boolean b);
public static AnyBoolean trueValue();
public static AnyBoolean falseValue();
```

```
public static AnyDouble doubleValue(double d);
public static AnyString stringValue(String s);
public static FlatAnyString flatAnyValue(String s);
public static AnyInteger integerValue(int i);
public static AnyNull nullValue();
```

6.4 Exceptions

The following runtime exceptions have been defined:

AnyReaderException	wrong input format, invalid reader for this input type, no reader registered for this input type, frame descriptor not in symbol table
AnyClassCastException	*asAnyXXX()* to wrong type
AnyFrameSlotException	specified slot does not exist in frame descriptor
AnyNullPointerException	trying to insert null (to *AnyDictionary*)
AnyNoQueryProcessorException	no query processor set
AnyWriterException	no write method for *Any* of this type (when inserting new subclasses of *Any*)

6.5 Documented Errors

- Circular references cause infinitive loops in *isEqual* and *hashCode* for *Any*s. The reason for that is that a structural compare is done to test for equality and so the messages are recursively sent to all components of an *Any*.
- Robust and fast writers currently write to a single string. An overflow will occur for very large *Any*s.
- *FileIOManagerImpl* reads the whole file into a single string so that it can be processed by the *AnyReader*s. This will not work for large files.

7 Interface Documentation

This chapter lists all interfaces of the framework. Descriptions are given for complex interfaces and where behavior is not apparent from the signature.

7.1 Any

```
// comparing
public boolean isEqual(Any anAny);
public boolean notEqual(Any anAny);
public boolean isIdentical(Any anAny);
public boolean notIdentical(Any anAny);

// copying
public Any shallowCopy();
public Any deepCopy();
public Any deepCopy(AnyContext anAnyContext);
  //copy using different context

// testing
public boolean isObjectType();
public boolean isValueType();
public boolean isSimple();

// casting
public AnyArray asAnyArray();
public AnyBoolean asAnyBoolean();
public AnyDictionary asAnyDictionary();
public AnyDouble asAnyDouble();
public AnyFrame asAnyFrame();
public AnyFrameDescriptor asAnyFrameDescriptor();
public AnyInteger asAnyInteger();
public AnySlotDescriptor asAnySlotDescriptor();
public AnyString asAnyString();

// accessing value
public boolean asBoolean();
public double asDouble();
public int asInteger();
public String asString();

// printing
public void print();
  // pretty to System.out
public String printOnString(boolean fast);

// misc
public void addAllTo(AnyArray anAnyArray);
```

7.2 AnyBoolean

```
// accessing value
public boolean asBoolean();
public int asInteger();

// comparing
public boolean isEqual(boolean b);
public boolean notEqual(boolean b);
```

7.3 AnyDouble

```
// accessing value
public double asDouble();

// testing
public boolean isEqual(double d);
public boolean notEqual(double d);
```

7.4 AnyInteger

```
// accessing value
public int asInteger();

// comparing
public boolean isEqual(int i);
public boolean notEqual(int i);
```

7.5 AnyString

```
// accessing value
public String asString();
public int asInteger();

// comparing
public boolean isEqual(String aString);
public boolean notEqual(String aString);
```

7.6 FlatAnyString

```
// creating frame
public AnyFrame copyToFrame(AnyContext context);
```

7.7 AnyArray

AnyArray has single-value semantics.

```
// accessing
public int size();

// accessing elements
public Any at(int pos);
  // IndexOutOfBoundsException is raised for invalid position

// manipulating elements
public int append(Any newElement);
public void appendArray(AnyArray newElements);
public Any atPut(int pos, Any newElement);
  // If required (i.e. pos>=size) array is expanded.
public void insert(int pos, Any newElement);
```

```
public Any remove(int pos);
  // return null if pos does not exist
public void remove(Any element);
  // removes all identical entries
public void removeAll();

// finding elements (test for equality)
public AnyArray findAll(Any element);
public AnyArray findAllWithSlot(AnyString key, Any anAny);
  // find frames or dictionaries whose value at key is equals anAny
public AnyArray findAllWithSlot(String key, Any anAny);
public Any findFirst(Any element);
public int findFirstPosition(Any element);
public Any findFirstWithSlot(AnyString key, Any a);
  // find frame or dictionary whose value at key is equals anAny
public Any findFirstWithSlot(String key, Any a);

// misc
public void expand(int newSize);
public void sort();
public void sort(AnyString key);
  // use key as sort criteria
public void sort(String key);
  // in AnyFrame or AnyDictionary
```

7.8 AnyDictionary

Multi-Value Semantics

AnyDictionary has multi-value semantics. Multiple values are kept in *AnyArrays*. If there is only one value for a given key it is directly stored without using an array for performance reasons.

To make things clearer the term *element* will be used when talking about a single value, and *value* to refer to the whole value for a given key, which can be either an *Any* or an *AnyArray*.

When a new element is being added for a key whose value currently consists of only one element, value will be transformed to an array of size two.

Invalid Key Policy

Additional slots can be added to an *AnyDictionary* using *atPut*, *append*, *appendArray* and *setSlot*. If the given key does not exist in the dictionary a new slot is created.

Null is returned when trying to access a slot using an invalid key (except where return type is int (e.g. *sizeAt*), there –1 is returned).

Null Policy

It is not possible to insert null in an *AnyDictionary* as a return value of null means that the slot does not exist. *AnyNullPointerException* is raised when trying to do so.

```
// accessing
public AnyArray keys();
public AnyArray values();
public int size();
  // number of slots
public int fullSize();
  // number of elements

// accessing slots
public Any at(AnyString key);
  // corresponds to at(key, 0)
public Any at(String key);
public Any at(AnyString key, int pos);
  // if key does not exist: return null
```

```
  // if value is an array: return element at position pos, raise
  //      IndexOutOfBounds for invalid positions
  // otherwise: if pos=new return value, else throw IndexOutOfBounds
public Any at(String key, int pos);
public Any slotAt(AnyString key);
  // if key does not exist: return null
  // otherwise: return value (as Any or AnyArray, whatever it is)
public Any slotAt(String key);
public AnyArray arrayAt(AnyString key);
  // if value is an array: return value
  // otherwise: return array with value as element
public AnyArray arrayAt(String key);
public int sizeAt(AnyString key);
  // if key does not exist: return -1
  // if value is an array: return size of array
  // otherwise: return 1
public int sizeAt(String key);

// manipulating slots
public void append(AnyString key, Any newElement);
  // if key does not exist: add key, rest like atPut(key,newElement)
  // if value is an array: append newElement to array
  // otherwise: set value to array where first element is old value
  //      and second element is newElement
public void append(String key, Any newElement);
public void appendArray(AnyString key, AnyArray newElements);
  // if key does not exist: add key, set value to newElements
  // if value is an array: append new elements to array
  // otherwise: set value to array where first element is old value
       and new elements are appended
public void appendArray(String key, AnyArray newElements);
public void atPut(AnyString key, Any newElement);
  // corresponds to atPut(key, 0, newElement)
public void atPut(String key, Any newElement);
public void atPut(AnyString key, int pos, Any newElement);
  // if key does not exist: add key, rest like otherwise
  // if value is an array: replace element at pos, dynamically
  //      adjust size of array if required
  // otherwise:
  //      pos=0: replace value, unless newElement is an array,
  //            then set value to array containing newElement
  //      otherwise: set value to array of size required where
  //            first element is old value and last newElement
public void atPut(String key, int pos, Any newElement);
public void setSlot(AnyString key, Any value);
  // if key does not exist: add key, rest like otherwise
  // otherwise: set value to new value
public void setSlot(String key, Any value);
public Any remove(AnyString key);
  // corresponds to remove(key, 0)
public Any remove(String key);
public Any remove(AnyString key, int pos);
  // if key does not exist: return null
  // if value is an array:
  //      if pos=0 and there is only one element: remove slot
  //      if pos invalid: return null
  //      otherwise: remove element at pos, return it
  // otherwise:
  //      if pos=0: remove slot
  //      otherwise: throw IndexOutOfBounds
public Any remove(String key, int pos);
public AnyArray removeSlot(AnyString key);
  // if key does not exist: return null
  // if value is an array: remove slot, return array
  // otherwise: remove slot, return array containing removed value
public AnyArray removeSlot(String key);


// testing
public boolean hasSlot(AnyString key);
public boolean hasSlot(String key);
```

7.9 AnyFrame

Multi-Value Semantics

Same as *AnyDictionary*

Invalid Key Policy

An *AnyFrame* consists of a closed set of slots as described in its *AnyFrameDescriptor*. Therefore trying to access a slot using an invalid slot name results in an *AnyFrameSlotException* being thrown.

Null Policy

Null can be inserted in an *AnyFrame*, it means that this slot has not been set so far.

```
// accessing
public AnyContext getContext();
public AnyFrameDescriptor getFrameDescriptor();
public AnyString getTypeName();
public int size();
  // return number of slots
public int fullSize();
  // return number of elements
public AnyArray slotNames();
public AnyArray slotValues();

// accessing slots
public Any at(AnyString slotName);
  // corresponds to at(key, 0)
public Any at(String slotName);
public Any at(AnyString slotName, int pos);
  // if slotName invalid: throw AnyFrameSlotException
  // if value is an array: return element at position pos, raise
  //      IndexOutOfBounds for invalid positions
  // otherwise: if pos=new return value, else throw IndexOutOfBounds
public Any at(String slotName, int pos);
public Any slotAt(AnyString slotName);
  // if slotName invalid: throw AnyFrameSlotException
  // otherwise: return value (as Any or AnyArray, whatever it is)
public Any slotAt(String slotName);
public AnyArray arrayAt(AnyString slotName);
  // if value is an array: return value
  // if value is null: return empty array
  // otherwise: return array with value as element
public AnyArray arrayAt(String slotName);
public int sizeAt(AnyString slotName);
  // if slotName invalid: throw AnyFrameSlotException
  // if value is an array: return size of array
  // if value is null: return 0
  // otherwise: return 1
public int sizeAt(String slotName);

// manipulating slots
public void append(AnyString slotName, Any newElement);
  // if slotName invalid: throw AnyFrameSlotException
  // if value is an array: append newElement to array
  // if value is null: like atPut(slotName, newElement)
  // otherwise: set value to array where first element is old value
  //      and second element is newElement
public void append(String slotName, Any newElement);
public void appendArray(AnyString slotName, AnyArray newElements);
  // if slotName invalid: throw AnyFrameSlotException
  // if value is an array: append new elements to array
  // if value is null: set value to newElements
  // otherwise: set value to array where first element is old value
       and new elements are appended
public void appendArray(String slotName, AnyArray newElements);
public void atPut(AnyString slotName, Any newElement);
```

```
  // corresponds to atPut(slotName, 0, newElement)
public void atPut(String slotName, Any newElement);
public void atPut(AnyString slotName, int pos, Any newElement);
public void atPut(String slotName, int pos, Any newElement);
  // if slotName invalid: throw AnySlotNameException
  // if value is an array: replace element at pos, dynamically
  //      adjust size of array if required
  // otherwise:
  //      pos=0: replace value, unless newElement is an array,
  //             then set value to array containing newElement
  //      pos!=0:set value to array of size required where
  //             first element is old value (unless it was null)
  //             and last newElement
public void setSlot(AnyString slotName, Any value);
  // if slotName invalid: throw AnySlotNameException
  // otherwise: set value to new value
public void setSlot(String slotName, Any value);
public Any remove(AnyString slotName);
  // corresponds to remove(slotName, 0)
public Any remove(String slotName);
public Any remove(AnyString slotName, int pos);
  // if slotName invalid: throw AnySlotNameException
  // if value is an array:
  //      if pos=0 and there is only one element: set value to null
  //      if pos invalid: return null
  //      otherwise: remove element at pos, return it
  // otherwise:
  //      if pos=0: set value to null
  //      otherwise: throw IndexOutOfBounds
public Any remove(String slotName, int pos);

// testing
public boolean hasSlot(AnyString slotName);
public boolean hasSlot(String slotName);
public boolean isA(AnyString descriptorName);
  // name of frame descriptor is equal descriptorName
public boolean isA(String descriptorName);
public boolean isKindOf(AnyString descriptorName);
  // name of frame descriptor or of one in the super frame
  // descriptor chain is equal descriptorName
public boolean isKindOf(String descriptorName);
```

7.10 AnyFrameDescriptor

```
// accessing
public AnyDictionary getContents();
  // get collection of slot name / slot descriptor pairs
public AnyContext getContext();
public AnyString getDescription();
public AnyString getName();
public AnyFrameDescriptor getSuperFrameDescriptor();
public int numberOfSlots();
public int totalNumberOfSlots();
  // including super frame descriptor chain
public AnyArray slotNames();
public AnyArray allSlotNames();
  // including super frame descriptor chain

// setting
public void setContents(AnyDictionary anAnyDictionary);
public void setDescription(AnyString description);
public void setDescription(String description);
public void setName(AnyString anAnyString);
public void setName(String anAnyString);
public void setSuperFrameDescriptor(
  AnyFrameDescriptoranAnyFrameDescriptor);

// accessing slots
public AnySlotDescriptor at(AnyString key);
```

```
public AnySlotDescriptor at(String key);
public int offset(AnyString key);
  // return index of slotName key within frame

// manipulations slots
public void append(AnyString key, AnySlotDescriptor desc);
public void append(String key, AnySlotDescriptor desc);
public AnySlotDescriptor remove(AnyString key);
public AnySlotDescriptor remove(String key);

// misc
public AnyFrame convertToFrame();
```

7.11 AnySlotDescriptor

```
// accessing
public AnyString getTypeName();
public boolean isMayNotRemove();
public boolean isMustHave();
public boolean isNotEditable();
public boolean isSingleValue();
public String flagsToString();

// setting
public void setTypeName(AnyString aString);
public void setMayNotRemove();
public void setMustHave();
public void setNotEditable();
public void setSingleValue();
public void resetMayNotRemove();
public void resetMustHave();
public void resetNotEditable();
public void resetSingleValue();
```

7.12 AnyContext

```
// accessing
public AnySymbolTable getSymbolTable();
```

7.13 AnySoup

```
// accessing
public IndexManager getIndexManager();
public int getNextId();
public AnyArray getData();
  // return frames in soup
public AnyArray getAll();

// setting
public void setQueryProcessor(QueryProcessor aQueryProcessor);
public void setData(AnyArray newData);
  // set frames in soup

// frame management
public void add(AnyFrame anAnyFrame);
public void remove(AnyFrame aFrame);

// IO
public void store(AnyString fileName)
  throws IOException, FileNotFoundException;
public void store(String fileName)
  throws IOException, FileNotFoundException;
public void load(AnyString fileName)
  throws IOException, FileNotFoundException;
public void load(String fileName)
  throws IOException, FileNotFoundException;
```

```
public void storeMetaInformation(AnyString fileName)
  throws IOException, FileNotFoundException;
public void storeMetaInformation(String fileName)
  throws IOException, FileNotFoundException;
public void loadMetaInformation(AnyString fileName)
  throws IOException, FileNotFoundException;
public void loadMetaInformation(String fileName)
  throws IOException, FileNotFoundException;

// locking
public boolean lock();
public boolean unlock(boolean doCommit);
public boolean isLocked();
public void setClientInformation(boolean isLockerActive);

// queries
public AnyArray evaluateQuery(AnyString queryString);
public AnyArray evaluateQuery(String queryString);
public void removeByQuery(AnyString queryString);
public void removeByQuery(String queryString);

// printing
public void print();
  // pretty to System.out
```

7.14 AnySymbolTable

AnySymbolTable is basically like an *AnyDictionary* except that it has single-value semantics. For all insert operations *descriptor.name()* is used as key.

```
// accessing
public int size();
public AnyArray all();

// frame descriptor management
public AnyFrameDescriptor at(AnyString key);
  // null is returned if slot is not found
public void insert(AnyFrameDescriptor descriptor);
public void insertAll(AnyArray descriptors);

// printing/reading
public String printOnString(boolean fast);
public void readFromString(String in, AnyContext context);
```

7.15 AnyReader

```
// accessing
public String getType();

// reading
public Any readAny(String in, AnyContext context);
```

7.16 AnyWriter

```
// accessing
public String getType();

// writing
public String writeAny(Any anAny);
```

7.17 IOManager

```
// accessing
```

```
public AnySoup getSoup();
public int getNextId();

// setting
public void setSoup(AnySoup soup);

// IO
public void store(AnyString fileName) throws IOException;
public void load(AnyString fileName)
  throws FileNotFoundException, IOException;
public void storeMetaInformation(AnyString fileName)
  throws IOException;
public void loadMetaInformation(AnyString fileName)
  throws FileNotFoundException,IOException;
```

7.18 IndexManager

```
// managing frame indices
public AnyArray getFrameIndex(AnyString key);
public AnyArray getFrameIndex(String key);
public void addFrameIndex(AnyString indexName);
public void addFrameIndex(String indexName);
public void removeFrameIndex(AnyString indexName);
public void removeFrameIndex(String indexName);

// managing other indices
public AnyArray getIndex(AnyString indexName);
public AnyArray getIndex(String indexName);
public void addIndex(AnyString indexName, AnyArray index);
public void addIndex(String indexName, AnyArray index);
public void removeIndex(AnyString indexName);
public void removeIndex(String indexName);

// enrolling / unrolling    (only to/from frame indices)
public void enroll(AnyFrame anAnyFrame);
public void unroll(AnyFrame anAnyFrame);

// printing
public void printStatistics();    // to System.out

// misc
public IndexManager duplicateWithSettings();
```

7.19 InfoService

```
// exploring soup
public AnyArray getByKind(String descriptorName);
public AnyArray getBySlotStartingWith(
    String descriptorName, String slotName, String prefix);
public AnyArray getBySlotValue(
    String descriptorName, String slotName, Any value);
public AnyFrame getFirstBySlotValue(
    String descriptorName, String slotName, Any value);

// manipulating soup
public void setSlot(
    String descriptorName, String slotName, Any newValue);
public void cleanSlot(String descriptorName, String slotName);
    // set values to null

// queries
public AnyArray evaluateQuery(String queryExpression);
public void removeByQuery(String queryExpression);
```

7.20 PartyInfoService

```
// exploring soup
public AnyFrame getById(int id);
public AnyArray getByName(String name);
public AnyArray getByNameStartingWith(String prefix);
public AnyArray getByRole(String role);
public AnyArray getByContactPerson(AnyFrame contactPerson);

// manipulating soup
public void cleanCallHistory();
public void removeAdvisor(String name);
```

7.21 QueryProcessor

```
// accessing
public AnySoup getSoup();

// setting
public void setSoup(AnySoup anAnySoup);

// processing queries
public AnyArray evaluateQuery(AnyString queryString);
```

7.22 AnyFormatter

```
// accessing
public String getIndentation();

// setting
public void setDoIndent();
public void setNoIndent();
public void setIndentationString(String identString);

// formatting
public void decrement();
public void increment();
```

8 Samples Uses

8.1 Sample1

```
import Geo.Frameworks.Any.*;
import Geo.Frameworks.AnyImpl.*;
import java.io.IOException;

public class Sample1 {

// a simple frame descriptor and several frames are created
// in a context
  // frames are written in several formats and read again

  public static AnyContext context;
  public static AnyDictionary parents;

  private static final String PERSON = "Person";
  private static final String FIRST_NAME = "FirstName";
  private static final String LAST_NAME = "LastName";
  private static final String CHILDREN = "Children";
  private static final String MOTHER = "Mother";
  private static final String FATHER = "Father";

  private static void createAnys() {
      AnyFrame anna = new AnyFrameImpl(context, PERSON);
      anna.append(FIRST_NAME, AnySap.stringValue("Anna"));
      anna.append(LAST_NAME, AnySap.stringValue("Gili"));

      AnyFrame klaus = new AnyFrameImpl(context, PERSON);
      klaus.append(FIRST_NAME, AnySap.stringValue("Klaus"));
      klaus.append(LAST_NAME, AnySap.stringValue("Mueller"));

      AnyFrame vreni = new AnyFrameImpl(context, PERSON);
      vreni.append(FIRST_NAME, AnySap.stringValue("Vreni"));
      vreni.append(LAST_NAME, AnySap.stringValue("Spiss"));

      AnyFrame eugen = new AnyFrameImpl(context, PERSON);
      eugen.append(FIRST_NAME, AnySap.stringValue("Eugen@ "));
      eugen.append(LAST_NAME, AnySap.stringValue("Spring"));

      AnyArray kids = new AnyArrayImpl();
      kids.append(anna);
      kids.append(klaus);
      vreni.append(CHILDREN, kids);
      eugen.append(CHILDREN, kids);

      parents.append(MOTHER, vreni);
      parents.append(FATHER, eugen);
  }

  private static void createFrameDescriptor() {
```

```
      AnyFrameDescriptor personDesc =
          new AnyFrameDescriptorImpl(context, PERSON);
      personDesc.append(FIRST_NAME,
          new AnySlotDescriptorImpl("AnyString"));
      personDesc.append(LAST_NAME,
          new AnySlotDescriptorImpl("AnyString"));
      personDesc.append(CHILDREN,
          new AnySlotDescriptorImpl("AnyArray"));
  }

  public static void main(String[] args) throws IOException {
      parents = new AnyDictionaryImpl();
      context = new AnyContextImpl();

      createFrameDescriptor();
      createAnys();

      // pretty output
      parents.print();

      AnyWriter writer;
      AnyReader reader;
      String outString;
      Any newParents;

      // fast read and write parents (using registered reader)
      writer = new FastAnyWriterImpl();
      outString = writer.writeAny(parents);
      AnyReaderDefImpl.register(new FastAnyReaderImpl());
      newParents = AnyReaderDefImpl.read(outString, context);

      // robust read and write (using instantiated reader)
      writer = new RobustAnyWriterImpl();
      reader = new RobustAnyReaderImpl();
      outString = writer.writeAny(parents);
      newParents = reader.readAny(outString, context);
  }
}
```

8.2 Sample2

```
import Geo.Frameworks.Any.*;
import Geo.Frameworks.AnyImpl.*;
import java.io.*;

public class Sample2 {

  // frame descriptors with super frame descriptors are created
  // in a soup
  // frames are created and added to soup
  // soup including meta information is stored to file and re-loaded

  public static AnySoup soup;

  // frame descriptor names
  public static final String CREATURE = "creature";
  public static final String PERSON = "person";
  public static final String FAMILY_MEMBER = "familyMember";
  public static final String PROFESSIONAL = "professional";

  // slot names
  public static final String NAME = "name";
  public static final String AGE = "age";
  public static final String IS_MARRIED = "isMarried";
  public static final String CHILDREN = "children";
  public static final String FAVORITES = "favorites";
  public static final String ROLES = "Roles";
  public static final String SALARY = "salary";
  public static final String OCCUPATION = "occupation";
```

```
  public static final String EMPLOYER = "employer";

  private static void createFrameDescriptors() {

    AnyFrameDescriptor creature =
          new AnyFrameDescriptorImpl(soup, CREATURE);
    creature.append(NAME, new AnySlotDescriptorImpl("AnyString",
          false, true, true, true));
          // mayNotRemove, mustHave, notEditable
    creature.append(AGE, new AnySlotDescriptorImpl("AnyInteger",
          true, true, true, true));
          // singleValue, mayNotRemove, mustHave, notEditable

    AnyFrameDescriptor person =
          new AnyFrameDescriptorImpl(soup, PERSON);
    person.setDescription("this is a person");
    person.append(CHILDREN, new AnySlotDescriptorImpl("AnyArray"));
    person.append(IS_MARRIED,
          new AnySlotDescriptorImpl("AnyBoolean"));
    person.append(FAVORITES,new AnySlotDescriptorImpl("AnyDictionary",
          false, true, false, false));
          // mayNotRemove
    person.setSuperFrameDescriptor(creature);

    AnyFrameDescriptor familyMember =
          new AnyFrameDescriptorImpl(soup, FAMILY_MEMBER);
    familyMember.append(ROLES, new AnySlotDescriptorImpl("AnyArray",
          false, true, true, false));
          // mustHave, mayNotRemove
    familyMember.setSuperFrameDescriptor(person);

    AnyFrameDescriptor professional =
          new AnyFrameDescriptorImpl(soup, PROFESSIONAL);
    professional.append(SALARY,
          new AnySlotDescriptorImpl("AnyDouble"));
    professional.append(OCCUPATION,
          new AnySlotDescriptorImpl("AnyString"));
    professional.append(EMPLOYER,
          new AnySlotDescriptorImpl("AnyString"));
    professional.setSuperFrameDescriptor(person);
  }

  private static void createFrames() {
    AnyDictionary annaFavorites = new AnyDictionaryImpl();
    annaFavorites.append("drink", AnySap.stringValue("Coke"));

    AnyFrame anna = new AnyFrameImpl(soup, PROFESSIONAL);
    anna.append(NAME, AnySap.stringValue("Anna"));
    anna.append(NAME, AnySap.stringValue("Luisa"));
    anna.append(AGE, AnySap.integerValue(26));
    anna.append(IS_MARRIED, AnySap.trueValue());
    anna.append(OCCUPATION, AnySap.stringValue("Surgeon"));
    anna.append(EMPLOYER, AnySap.stringValue("St. Georges Hospital"));
    anna.append(SALARY, AnySap.doubleValue(23946.75));
    anna.append(FAVORITES, annaFavorites);
    soup.add(anna);

    AnyDictionary klausFavorites = new AnyDictionaryImpl();
    klausFavorites.append("drink", AnySap.stringValue("Rivella"));
    klausFavorites.append("person", anna);
    klausFavorites.append("number", AnySap.integerValue(5));

    AnyFrame klaus = new AnyFrameImpl(soup, PROFESSIONAL);
    klaus.append(NAME, AnySap.stringValue("Klaus"));
    klaus.append(IS_MARRIED, AnySap.falseValue());
    klaus.append(OCCUPATION, AnySap.stringValue("Chiropractor"));
    klaus.append(FAVORITES, klausFavorites);
    soup.add(klaus);

    AnyArray kids = new AnyArrayImpl();
    kids.append(anna);
```

```
    kids.append(klaus);

    AnyArray vreniRoles = new AnyArrayImpl();
    vreniRoles.append(AnySap.stringValue("mother"));

    AnyFrame vreni = new AnyFrameImpl(soup, FAMILY_MEMBER);
    vreni.append(NAME, AnySap.stringValue("Vreni"));
    vreni.append(IS_MARRIED, AnySap.booleanValue(true));
    vreni.append(ROLES, vreniRoles);
    vreni.append(CHILDREN, kids);
    vreni.append(FAVORITES, klausFavorites);
    soup.add(vreni);

    AnyArray eugenRoles = new AnyArrayImpl();
    eugenRoles.append(AnySap.stringValue("father"));
    eugenRoles.append(AnySap.stringValue("oncle"));

    AnyFrame eugen = new AnyFrameImpl(soup, FAMILY_MEMBER);
    eugen.append(NAME, AnySap.stringValue("Eugen"));
    eugen.append(ROLES, eugenRoles);
    eugen.append(CHILDREN, kids);
    soup.add(eugen);
  }

  public static void main(String[] args)
        throws IOException, FileNotFoundException {
    soup = new AnySoupImpl();
    createFrameDescriptors();
    createFrames();

    // store
    System.out.println("\n-- Storing to \"soup.meta\"...");
    soup.storeMetaInformation("soup.meta");
    System.out.println("-- Storing to \"soup.data\"...");
    soup.store("soup.data");

    // load
    AnyReaderDefImpl.register(new FastAnyReaderImpl());
    AnyReaderDefImpl.register(new RobustAnyReaderImpl());
    System.out.println("-- Loading...");
    AnySoup newSoup =
          AnySoupImpl.readFromFile("soup.meta", "soup.data");
  }
}
```

8.3 Sample3

```
import Geo.Frameworks.Any.*;
import Geo.Frameworks.AnyImpl.*;

class Sample3 {

  // create frame descriptor in soup with index manager
  // add frame index for both frame descriptors
  // create frames and add to soup
  // use index manager to get list of all frames with a
  // specific frame descriptor
  // lock soup, manipulate data and rollback afterwards
  // lock soup, manipulate data and commit afterwards

  private static AnySoup soup;
  private static AnyFrameDescriptor person;
  private static AnyFrame mariChecking;
  private static AnyFrame mariSavings;

  private static void createFrameDescriptors() {
    person = new AnyFrameDescriptorImpl(soup, "person");
    person.append("firstName",
          new AnySlotDescriptorImpl("AnyString"));
```

```
    person.append("lastName", new AnySlotDescriptorImpl("AnyString"));

    AnyFrameDescriptor account =
          new AnyFrameDescriptorImpl(soup, "account");
    account.append("type", new AnySlotDescriptorImpl("AnyString"));
    account.append("number", new AnySlotDescriptorImpl("AnyString"));
    account.append("owner", new AnySlotDescriptorImpl("AnyFrame"));
    account.append("amount", new AnySlotDescriptorImpl("AnyDouble"));
  }

  private static void createFrames() {
    AnyFrame susi = new AnyFrameImpl(person);
    susi.append("firstName", AnySap.stringValue("Susanne"));
    susi.append("lastName", AnySap.stringValue("Mueller"));
    soup.add(susi);

    AnyFrame mari = new AnyFrameImpl(person);
    mari.append("firstName", AnySap.stringValue("Marilena"));
    mari.append("lastName", AnySap.stringValue("Gonzales"));
    soup.add(mari);

    AnyFrame susiSavings = new AnyFrameImpl(soup, "account");
    susiSavings.append("type", AnySap.stringValue("savings"));
    susiSavings.append("number", AnySap.stringValue("491.664.01M"));
    susiSavings.append("owner", susi);
    susiSavings.append("amount", AnySap.doubleValue(45967.45));
    soup.add(susiSavings);

    AnyFrame susiChecking = new AnyFrameImpl(soup, "account");
    susiChecking.append("type", AnySap.stringValue("checking"));
    susiChecking.append("number", AnySap.stringValue("491.664.01T"));
    susiChecking.append("owner", susi);
    susiChecking.append("amount", AnySap.doubleValue(345.56));
    soup.add(susiChecking);

    mariSavings = new AnyFrameImpl(soup, "account");
    mariSavings.append("type", AnySap.stringValue("savings"));
    mariSavings.append("number", AnySap.stringValue("491.587.01M"));
    mariSavings.append("owner", mari);
    mariSavings.append("amount", AnySap.doubleValue(1728.4));
    // mariSavings is not added to soup yet

    mariChecking = new AnyFrameImpl(soup, "account");
    mariChecking.append("type", AnySap.stringValue("checking"));
    mariChecking.append("number", AnySap.stringValue("491.587.01T"));
    mariChecking.append("owner", mari);
    mariChecking.append("amount", AnySap.doubleValue(489.06));
    soup.add(mariChecking);
  }

  public static void main(String[] args) {
    soup = new AnySoupImpl(new SimpleIndexManagerImpl());
    soup.getIndexManager().addFrameIndex("person");
    soup.getIndexManager().addFrameIndex("account");

    createFrameDescriptors();
    createFrames();

    soup.getIndexManager().printStatistics();
    soup.getIndexManager().getFrameIndex("ALL_person").print();

    // lock and make changes
    soup.lock();
    soup.add(mariSavings);
    soup.remove(mariChecking);
    soup.getIndexManager().removeFrameIndex("person");
    soup.getData().at(0).asAnyFrame().atPut("firstName",
          AnySap.stringValue("Susi"));

    // rollback
    soup.unlock(false);
```

```
    System.out.println("\n\n-- After rollback:\n");
    soup.getIndexManager().printStatistics();

    // lock and make changes again
    soup.lock();
    mariSavings.atPut("owner", soup.getData().at(1));
    soup.add(mariSavings);
    soup.remove(mariChecking);
    soup.getIndexManager().removeFrameIndex("person");
    soup.getData().at(0).asAnyFrame().atPut("firstName",
          AnySap.stringValue("Susi"));

    // unlock and commit
    soup.unlock(true);
    System.out.println("\n\n-- After commit:\n");
    soup.getIndexManager().printStatistics();
  }
}
```

8.4 Party

```
import Geo.Frameworks.Any.*;
import Geo.Frameworks.AnyImpl.*;

public class Party {

  // Frames are created and added to soup
  // PartyInfoService is then used to explore and manipulate soup.

  private static int nextId;
  private static AnySoup soup;
  private static PartyInfoServiceImpl infoService;
  private static boolean printDetails;
  private static AnyFrame jeff;

  // frame descriptor names
  public static final String PARTY = "party";
  public static final String PERSON = "person";
  public static final String COOPERATION = "cooperation";

  // slot names
  public static final String ID = "id";
  public static final String NAME = "name";
  public static final String ADDRESS = "address";
  public static final String PHONE = "phone";
  public static final String ROLE = "role";
  public static final String CALL_HISTORY = "callHistory";
  public static final String ADVISOR = "advisor";
  public static final String PRIVATE_PHONE = "privatePhone";
  public static final String CONTACT_PERSON = "contactPerson";

  // roles
  public static final AnyString CUSTOMER =
  AnySap.stringValue("customer");
  public static final AnyString CONTACT =
      AnySap.stringValue("contact");
  public static final AnyString BUERGE =
      AnySap.stringValue("buerge");

  private static void createFrameDescriptors() {
    // party is super frame descriptor of person and cooperation

    AnyFrameDescriptor party =
          new AnyFrameDescriptorImpl(soup, PARTY);
    party.append(ID, new AnySlotDescriptorImpl("AnyInteger"));
    party.append(NAME, new AnySlotDescriptorImpl("AnyString"));
    party.append(ADDRESS, new AnySlotDescriptorImpl("AnyString"));
    party.append(PHONE, new AnySlotDescriptorImpl("AnyString"));
    party.append(ROLE, new AnySlotDescriptorImpl("AnyString"));
```

```
    party.append(CALL_HISTORY,
          new AnySlotDescriptorImpl("AnyString"));
    party.append(ADVISOR, new AnySlotDescriptorImpl("AnyString"));

    AnyFrameDescriptor person =
          new AnyFrameDescriptorImpl(soup, PERSON);
    person.append(PRIVATE_PHONE,
          new AnySlotDescriptorImpl("AnyString"));
    person.setSuperFrameDescriptor(party);

    AnyFrameDescriptor cooperation =
          new AnyFrameDescriptorImpl(soup, COOPERATION);
    cooperation.append(CONTACT_PERSON,
          new AnySlotDescriptorImpl("AnyFrame"));
    cooperation.setSuperFrameDescriptor(party);

    if (printDetails) {
      System.out.println("\nFrame Descriptors:\n");
      soup.getSymbolTable().all().print();
    }
  }

  private static void createFrames() {
    AnyFrame eileen = new AnyFrameImpl(soup, PERSON);
    eileen.append(ID, getNextId());
    eileen.append(NAME, AnySap.stringValue("Jefferson, Eileen"));
    eileen.append(ADDRESS,
      AnySap.stringValue("1201 Florida Avenue, Fargo, SD 31056"));
    eileen.append(PHONE, AnySap.stringValue("217 337 5041"));
    eileen.append(ROLE, CUSTOMER);
    eileen.append(CALL_HISTORY, AnySap.stringValue("15.7.97"));
    eileen.append(CALL_HISTORY, AnySap.stringValue("22.9.93"));
    eileen.append(PRIVATE_PHONE, AnySap.stringValue("225 398 4594"));
    soup.add(eileen);

    AnyFrame andrew = new AnyFrameImpl(soup, PERSON);
    andrew.append(ID, getNextId());
    andrew.append(NAME, AnySap.stringValue("Floyd, Andrew"));
    andrew.append(ADDRESS,
      AnySap.stringValue("45-25B Church Drive, Dakota, SD 31098"));
    andrew.append(PHONE, AnySap.stringValue("234 489 3245"));
    andrew.append(ROLE, CUSTOMER);
    andrew.append(PRIVATE_PHONE, AnySap.stringValue("578 234 0864"));
    soup.add(andrew);

    AnyFrame joy = new AnyFrameImpl(soup, PERSON);
    joy.append(ID, getNextId());
    joy.append(NAME, AnySap.stringValue("Pinkerton, Joy Mary"));
    joy.append(ADDRESS,
      AnySap.stringValue("258 Maloney Avenue, Jefferson, SD 31097"));
    joy.append(PHONE, AnySap.stringValue("234 356 4583"));
    joy.append(ROLE, CUSTOMER);
    soup.add(joy);

    AnyFrame kenneth = new AnyFrameImpl(soup, PERSON);
    kenneth.append(ID, getNextId());
    kenneth.append(NAME, AnySap.stringValue("Sun, Kenneth"));
    kenneth.append(ADDRESS,
      AnySap.stringValue("135 Green Hill Road, Fargo, SD 31056"));
    kenneth.append(PHONE, AnySap.stringValue("217 335 4735"));
    kenneth.append(ROLE, BUERGE);
    kenneth.append(PRIVATE_PHONE, AnySap.stringValue("357 357 5693"));
    soup.add(kenneth);

    AnyFrame jennifer = new AnyFrameImpl(soup, PERSON);
    jennifer.append(ID, getNextId());
    jennifer.append(NAME, AnySap.stringValue("Bennett, Jennifer"));
    jennifer.append(ADDRESS,
      AnySap.stringValue("34 High School Drive, Fargo, SD 31056"));
    jennifer.append(PHONE, AnySap.stringValue("217 378 9734"));
    jennifer.append(ROLE, BUERGE);
```

```
jennifer.append(ROLE, CUSTOMER);
jennifer.append(PRIVATE_PHONE,
  AnySap.stringValue("357 357 5693"));
soup.add(jennifer);

AnyFrame nuala = new AnyFrameImpl(soup, PERSON);
nuala.append(ID, getNextId());
nuala.append(NAME, AnySap.stringValue("Bennett, Nuala"));
nuala.append(ADDRESS,
  AnySap.stringValue("34 Park Avenue, William Creek, ND 87357"));
nuala.append(PHONE, AnySap.stringValue("467 345 6543"));
nuala.append(ROLE, CUSTOMER);
nuala.append(ROLE, CONTACT);
soup.add(nuala);

AnyFrame nuala1 = new AnyFrameImpl(soup, PERSON);
nuala1.append(ID, getNextId());
nuala1.append(NAME, AnySap.stringValue("Bennett, Nuala"));
nuala1.append(ADDRESS,
  AnySap.stringValue("15 Kingston Street, Willing, ND 87352"));
nuala1.append(PHONE, AnySap.stringValue("378 942 5783"));
nuala1.append(ROLE, CONTACT);
nuala1.append(ROLE, CUSTOMER);
nuala1.append(ADVISOR, AnySap.stringValue("Hannelore"));
nuala1.append(ADVISOR, AnySap.stringValue("Hildegard"));
soup.add(nuala1);

jeff = new AnyFrameImpl(soup, PERSON);
jeff.append(ID, getNextId());
jeff.append(NAME, AnySap.stringValue("Brown, Jeff"));
jeff.append(ADDRESS,
  AnySap.stringValue("45 High School Drive, Fargo, SD 31056"));
jeff.append(PHONE, AnySap.stringValue("217 463 5698"));
jeff.append(ROLE, CONTACT);
jeff.append(ADVISOR, AnySap.stringValue("Hannelore"));
soup.add(jeff);

AnyFrame powerPlant = new AnyFrameImpl(soup, COOPERATION);
powerPlant.append(ID, getNextId());
powerPlant.append(NAME, AnySap.stringValue("Fargo Power Plant"));
powerPlant.append(ADDRESS,
  AnySap.stringValue("345 Power Plant Street, Fargo, SD 31056"));
powerPlant.append(PHONE, AnySap.stringValue("217 457 2398"));
powerPlant.append(ROLE, CUSTOMER);
powerPlant.append(CALL_HISTORY, AnySap.stringValue("12.4.89"));
powerPlant.append(ADVISOR, AnySap.stringValue("Kunigunde"));
powerPlant.append(ADVISOR, AnySap.stringValue("Hannelore"));
powerPlant.append(CONTACT_PERSON, jeff);
soup.add(powerPlant);

AnyFrame macDonnell = new AnyFrameImpl(soup, COOPERATION);
macDonnell.append(ID, getNextId());
macDonnell.append(NAME, AnySap.stringValue("Mac Donnell Ltd."));
macDonnell.append(ADDRESS,
  AnySap.stringValue("345 Highway Drive, Fargo, SD 31056"));
macDonnell.append(PHONE, AnySap.stringValue("217 356 4378"));
macDonnell.append(ROLE, CUSTOMER);
macDonnell.append(ADVISOR, AnySap.stringValue("Kunigunde"));
macDonnell.append(ADVISOR, AnySap.stringValue("Melchior"));
macDonnell.append(CONTACT_PERSON, jeff);
soup.add(macDonnell);

AnyFrame fargo = new AnyFrameImpl(soup, COOPERATION);
fargo.append(ID, getNextId());
fargo.append(NAME, AnySap.stringValue("Fargo Medical Systems"));
fargo.append(ADDRESS,
  AnySap.stringValue("356 Highway Drive, Fargo, SD 31056"));
fargo.append(PHONE, AnySap.stringValue("217 356 3278"));
fargo.append(ROLE, CUSTOMER);
fargo.append(ROLE, BUERGE);
fargo.append(CONTACT_PERSON, nuala);
```

```
    soup.add(fargo);

    if (printDetails) {
      System.out.println("\nFrames:\n");
      soup.print();
    }
  }

  private static void exploreSoup() {
    System.out.println("\nFrame with id 5:");
    System.out.println("---------------\n");
    infoService.getById(5).print();

    System.out.println("\nFrames with name = \"Bennett, Nuala\":");
    System.out.println("-----------------------------------\n");
    infoService.getByName("Bennett, Nuala").print();

    System.out.println(
      "\nFrames with name starting with \"Bennett\":");
    System.out.println("----------------------------------------\n");
    infoService.getByNameStartingWith("Bennett").print();

    System.out.println("\nFrames with role = \"contact\":");
    System.out.println("----------------------------\n");
    infoService.getByRole(CONTACT.toString()).print();

    System.out.println("\nFrames with contact person = jeff:");
    System.out.println("---------------------------------\n");
    infoService.getByContactPerson(jeff).print();
  }

  private static void manipulateSoup() {
    System.out.println(
      "\nCleaning Call History, removing advisor \"Hildegard\"\n");
    infoService.cleanCallHistory();
    infoService.removeAdvisor("Hannelore");
    if (printDetails)
      soup.print();
  }

  private static AnyInteger getNextId() {
    return AnySap.integerValue(nextId++);
  }

  public static void main(String[] args) {
    if ( args.length > 0 ) printDetails = true;
    else printDetails = false;

    soup = new AnySoupImpl();
    infoService = new PartyInfoServiceImpl(soup);
    nextId = 0;
    createFrameDescriptors();
    createFrames();

    exploreSoup();
    manipulateSoup();
  }
}
```

References

[MB96] Mätzel Kai-Uwe and Bischofberger Walter (1996) “The Any Framework - A Pragmatic Approach to Flexibility.” In *Proceedings of the 2nd USENIX Conference on Object-Oriented Technologies and Systems* (Toronto, Canada, June 1996), pp. 179-190.