Revisiting the Anything Pattern

Stefan Tramm et. al.
Netcetera
403

AGENDA

- The Anything data container
- Applications of the Anything
- Implementation Details
- Enhancements for Database Access
- “data as code”
- Existing Alternatives

AGENDA

- The Anything data container
- Applications of the Anything Pattern
- Implementation Details
- Enhancements for Database Access
- “data as code”
- Existing Alternatives

Intro

The Anything design pattern provides a generic structured data container that is useful as universal (catch-all) operation parameter. In addition Anythings are well-suited as a flexible means of storing, retrieving and transmitting structured data values. This makes Anything an ideal implementation technique for configuration data.

Peter Sommerlad and Marcel Rüedi, 1998

Historic Evolvement

- early 90's: ET++, a C++ based framework, which introduces Anything
  - evolved into SNiFF+
- late 90's: the Anything found its way into the WebDisplay Framework
- Marcel Rüedi et al. ported the Anything from C++ to Java 1.1 (LGPL)
- some years of silence
- Java Anything now public available on Google Code (2006)
- Integration of the Anything pattern into an enterprise level data processing system in 2006, for
  - run time configuration
  - parameter passing
  - unit test data
  - simple database access

Essence

- The Anything implements a self describing, recursive data structure, eg:
  - it supports simple type values: Boolean, Long, Float, String
  - it supports (nested) sequences of values (‘*vectors*’ contain ‘*slots*’)
  - values can carry an optional key name (called ‘*slot name*’)

```
{
    /name “joe user”
    /age  25
    /comments {
       “this is a \”string\””
       “this_is_a_multiline\nstring”
    }
    /flag True
    # comments are also possible
}
```

AGENDA

- The Anything data container
- Applications of the Anything Pattern
- Implementation Details
- Enhancements for Database Access
- “data as code”
- Existing Alternatives

Anything Pattern

- **context**: you implement a framework, where users/developers can supply new subclasses
- **problem**: how to provide method parameters (or object data attributes) that fit the need for future subclasses?
- **problem**: how to make different subsystems compile-time independent (enable loose coupling)?
- **problem**: how to provide a generic configuration or communication data structure that is also easily extensible?
- elaboration:
  - pass an open set of structured (and typed) data to an (abstract) operation
  - easy to use internal and external representation
  - avoid dependency on yet another framework
- **solution**: use the Anything

Applications of the Anything: Configuration data

- the Anything stores a tree nodes
  - roughly the same as XML, but…
  - nodes carry a type, and can be easily coerced to another representation
  - a nicer and simpler API than DOM or SAX
  - one Anything class file instead of a whole framework
  - more concise external representation
- so embrace the features a tree can give you:
  - put structured parameters in files or into a DB (terse string representation)
  - use it for mock data input (unit test configuration)
  - compare test input and test results: the serialization is stable (insertion order), so you can compare easily (normal hashtables cant provide this)
  - a tree is more than simple property list key-value-pairs

Applications of the Anything: DOM

- the DOM for ‘the rest of us’
  - every Anything can be mapped directly into XML
  - most XML can be mapped into an Anything in a generic way

```
{
   /name “joe user”
   /age  25
   /comments {
      “this is a \”string\””
      “this_is_also_a_string”
   }
   /flag True
   # remark
}
```

```
<root>
   <name>joe user</name>
   <age>25</age>
   <comments>
      this is a “string”
   </commtents>
   <comments>
      this_is_also_a_string
   </comments>
   <flag>True</flag>
   <!-- remark -->
</root>
```

AGENDA

- The Anything data container
- Applications of the Anything Pattern
- Implementation Details
- Enhancements for Database Access
- “data as code”
- Existing Alternatives

API I

- Constructors
  - Anything a = new Anything(); // empty Any
  - Anything b = new Anything(1234); // an Any containing a Long
  - Anything c = new Anything(b); // an Any containing another Any
- Setters
  - a.put("slotname", "value"); // append or modify a slot
  - a.put(4, 2.3); // put at slot #4 in 'a' the double '2.3' (slot numbers start at 0)
  - a.append(True); // append an unnamed slot to 'a' containing the value True

API II

- Getters
  - Anything a = b.get(0); // read the slot #0 from Any b
  - Anything a = b.get("foo"); // read the slot named "foo"
  - String s = b.get("foo").asString("def"); // read the slot named "foo" and convert the Anything into a String; return "def" if slot "foo" does not exist
  - long s = b.get("foo").asLong(-1); // this works either, because the Any casts always to the requested target type or returns the given default
  - int type = b.getType(); // returns an int describing the real type of the slot

API III

- Predicates
  - bool b = a.isNull(); // do we have a Null Anything?
    (new Anything()).isNull() == True
  - bool b = a.isDefined("foo"); // does a contain a slot named "foo"?
- Helpers
  - int s = a.getSize(); // return the number of slots
  - String s = a.slotName(1); // return the name of slot #1
  - int i = a.findValue("something"); // return the slot index of the slot which equals to "something"

API IV

- Serialization
  - a.Print(os); // serialize Anything a in a human readable form onto os
  - a.PrintTerse(os); // serialize Anything a as one terse line onto os (which is nice for logging)
  - a.write("name.any"); // serialize Anything a into file "name.any"
  - Slots are serialized in insertion order! *(you will love this)*
- Deserialization
  - Anything a = (new Anything()).load(is); // load from input stream or reader
  - Anything a = Anything.read(is); // static method to deserialize from stream or reader
  - Anything a = Anything.read("name.any"); // read from file "name.any"
  - Anything a = Anything.create("{/k foobar}"); // create from string

Implementation Details

- Every Anything has three attributes:
  - a tag describing its content/type
  - an object containing the contents (the slots vector or a simple object)
  - an optional hash table to store slot names

```
public class Anything extends Object implements Serializable {
    public static final int eNull   = 0;
    public static final int eLong   = 1;
    ...
    public static final int eVector = 4;
    ...
    Object fContents;
    Hashtable fDict;
    int fTag;
    ...
```

Implementation Details

> Construction and automatic vectorization:

```
public Anything(int i) { // all constructors look equally
   fTag = eLong;
   fContents = new Long(i);
 }

public void append(Anything value) {
   if (fTag != eVector) {        // if not already a vector
     vectorize();                // make it so
   }
   int index = size();
   Vector v = (Vector) fContents;
   v.setSize(index + 1);         // make room for a new slot
   v.setElementAt(value, index); // store value in slot
 }
```

Implementation Details

- current implementation is based on Java 1.1
  - simple vector, hashtable and parser are used internally
  - multithreaded performs is not as good as possible, either use unthreaded containers or more modern ones from java.util.concurrent
  - the two containers may be replaced by an ordered tree implementation
- a read-only Anything is missing
  - an immutable after construction Anything
  - now ill behaving programs may change configuration data ‘on-the-fly’
  - hard to diagnose errors

AGENDA

- The Anything data container
- Applications of the Anything Pattern
- Implementation Details
- Enhancements for Database Access
- “data as code”
- Existing Alternatives

Support for the Builder pattern

- Setters return `this`, which makes cascading of setters possible
- this allows a more natural construction of non trivial Anythings:

```
Anything rows = new Anything();
for (int c = 0; c < 100; c++) {
    rows.get(c).put("EMPNO", 8000+c) // long/number
               .put("ENAME", "bulk") // string/varchar2
               .put("DEPTNO", 20)    // long/number
               .put("HIREDATE", new Date( // datetime
                     System.currentTimeMillis()));
}
r = AnyDAO.insertRows(connection, "EMP", rows);
```

- astonished? no?
  - think about, what get() has to do, to make this nice code work…

Database Access

- Anything as Data Transfer Object (DTO) / Value Object (VO)
  - it contains no business logic, only storage and retrieval
  - it supports sets (with its vector and hashtable)
  - it allows introspection at runtime (no code generation at compile needed)
- extension 1: more essential data types to support SQL
  - BigDecimal
  - Date
- extension 2: Data Access Object (DAO) aka JDBC access functions
  - AnyDAO implements: query(), insertRows(), statement()
  - query supports returning the complete result set
  - and invoking a callback handler for every row (streaming)
  - AnyDAO is not DAO in the JEE sense but a thin JDBC wrapper

DAO example: select & insert

- a simple query, demonstrating the self inspection possibilities:

```
Anything res = AnyDAO.query(con, "SELECT * FROM EMP");

BigDecimal sum = new BigDecimal(0);
for (int i = 0; i < res.get("Data").size(); i++) {
    sum = sum.add(res.get("Data").get(i).get("SAL")
                  .asBigDecimal(0));
}
System.out.println("Salary sum is “ + sum.toString());
```

- insertRows() was shown before

DAO example: statement

- you can reuse results from a select as input for a JDBC statement, eg. an update:

```
Anything inp = AnyDAO.query(connection,
 "SELECT * FROM EMP WHERE EMPNO = 1234");
// returns all attributes
inp.get(“Data”).get(0)
                  .put(“SAL”, 10000) // change attributes
                  .put(“ENAME”, “Stefan”);

Anything res = AnyDAO.statement(connection,
     new String[] {
      "UPDATE EMP SET ENAME=?, SAL=? WHERE EMPNO=?",
      "ENAME", “SAL", "EMPNO“
     }, inp);
// only the slots named "ENAME", “SAL", "EMPNO“ of inp
// are used, the other slots will be ignored
```

AGENDA

- The Anything data container
- Appliances of the Anything Pattern
- Implementation Details
- Enhancements for Database Access
- “data as code”
- Existing Alternatives

“data as code”

- as already mentioned, the Anything stores trees
- so you can store an AST (abstract syntax tree) in one Anything
- the step from a static AST to execution is simple
  - one eval() method, containing a huge switch (result: a DSL)
  - example: a simple Lisp interpreter based on Anything
    based on the classic paper by Steele, 1973 and Graham, 2001
- also included in source tree as ALang.java

- the scripting language Lua uses hash tables as main internal data structure
  - its implementation shows the convenience of tables in contrast to lists

- but: Anything does not mean *Everything*

eval() method

```
// see also http://paulgraham.com/rootsoflisp.html
public Anything eval(Anything e, Anything a) {
        if (atom(e)==t) {
            if (e.fTag==Anything.eLong || e.fTag==Anything.eDouble) return e;
            return assoc(e, a);
        } else if (atom(car(e))==t) { // invoke primitive
            String f = car(e).asString("");
            if (f.equals("quote"))  return cadr( e);
            if (f.equals("atom"))   return atom( eval(cadr(e),  a));
            if (f.equals("eq"))     return eq(   eval(cadr(e),  a),
                                                 eval(caddr(e), a));
            if (f.equals("car"))    return car(  eval(cadr(e),  a));
            if (f.equals("cdr"))    return cdr(  eval(cadr(e),  a));
            if (f.equals("cons"))   return cons( eval(cadr(e),  a),
                                                 eval(caddr(e), a));
            if (f.equals("cond"))   return evcon(cdr(e),        a);
            if (prims.isDefined(f)) return call( f, evlis(cdr(e), a), a);
            // else replace symbol with assoc
            return eval(cons(assoc(car(e), a), cdr(e)), a);
        } else if (caar(e).asString("").equals("label")) {
            return eval(cons(caddar(e), cdr(e)),
                        cons(list(cadar(e), car(e)), a));
        } else if (caar(e).asString("").equals("lambda")) {
            return eval(caddar(e), append(pair(cadar(e),
                                       evlis(cdr(e), a)), a));
        }
        return nil;
    }
```

AGENDA

- The Anything data container
- Applications of the Anything Pattern
- Implementation Details
- Enhancements for Database Access
- “data as code”
- Existing Alternatives

Existing Alternatives I

- JSON
  - many similarities: simple text representation, list and hash table containers, long and string values
  - but: serialization order of hashtables is undefined: bad for regression tests based on textual diffs
  - no support for Databases Type: BigDecimals and Date
- S-Expressions
  - based on a weaker data type (list instead of vector/hash table)
    constant vs. linear runtime of random access
  - supports only list, strings and binary objects
  - serialization format is reduced to the max; Anything has a richer and more human readable external format

Existing Alternatives II

- XML provides the same as the Anything and more
  - but, at a much higher price (code size, complexity, memory, runtime)
  - and no typing at all (without a complex and expensive schema checker)
  - but see Mark Reinholds talk at Javaone 2006 on ‘Integrating XML into the Java Programming Language’ (XOM)

```
XML newFeature(String name,
      String reviewer, String time)
{
  return #feature {
           #id { ++nFeatures },
           #name { name },
           #state { "submitted" },
           #reviewed {
             #who { reviewer },
             #when { time }
           }
         };
}
```

```
Anything newFeature(String name,
           String reviewer, String time)
{
  return
     new Anything()
        .get(“feature”)
           .put(“id”, ++nFeatures)
           .put(“name”, name)
           .put(“state”, "submitted“)
           .get(“reviewed”)
              .put(“who”,  reviewer)
              .put(“when”, time);
}
```

```
# Anything serialized
# an example tree
{
  /feature {
    /id 1234
    /name “Stefan”
    /state "submitted"
    /reviewed {
      /who “Marc”
      /when DATE 20070716
    }
  }
}
```

Existing Alternatives III

- Property lists
  - only support key value pairs
  - only strings as keys and values

- Database Tables
  - structure/schema cannot be expanded at runtime
  - expensive access
  - chicken-egg problem: what if, configuration should contain the database connection string?

Take home message

- replace adhoc object trees by well structured Anything trees
- use typed configuration data
- use a homogenous serialization syntax
- do not wait for a Java language extension (XOM) to use structured tree data in your programs

- think of the Anything as 'XML-Lite' or 'DOM for the rest of us'
  - simpler and more concise than DOM-API
  - more concise serialization format (no end-element)
  - good safety / flexibility / performance compromise

  - there is life outside the XML universe ;-)

Finis

http://code.google.com/p/java-anything/

Further readings

- P. Sommerlad and M. Rüedi, *Do-it-yourself reflection*, in EuroPLOP 98: Third European Conference on Pattern Languages of Programming and Computing, 1998. [Online]. Available: http://hillside.net/europlop/HillsideEurope/Papers/DIY_Reflection.pdf
- R. L. Rivest, *S-expressions*, Internet Engineering Task Force, Internet Draft, 1997. [Online]. Available: http://theory.lcs.mit.edu/~rivest/sexp.txt
- D. Crockford, *The application/json media type for javascript object notation (JSON),* Internet Engineering Task Force, RFC 4627, Jul 2006. [Online]. Available: http://ds.internic.net/rfc/rfc1738.txt
- M. Reinhold, *XOM: Integrating XML into the Java programming language*, in JavaONE 2006: Proceedings of the JavaOne Conference. (TS-3441) Sun, 2006.
- A. Weinand, E. Gamma, and R. Marty, *ET++ an object oriented application framework in C++,* in OOPSLA '88: Conference proceedings on Object Oriented programming systems, languages and applications. New York, NY, USA: ACM Press, 1988, pp. 4657.
- Kai-Uwe Mätzel, Walter Bischofberger, *The Any Framework - A Pragmatic Approach to Flexibility*, USENIX COOTS Toronto, 1996. [Online]. Available: http://www.ubilab.com/publications/print_versions/pdf/coots96.pdf

Further readings

- Data as code
  - Paul Graham, 2001
    http://paulgraham.com/rootsoflisp.html
  - Guy L. Steele, 1973
    http://repository.readscheme.org/ftp/papers/ai-lab-pubs/AIM-453.pdf

Stefan Tramm
Jason Brazile
David Oetiker
Stefan Rufer
Stefan Ferstl
Daniel Eichhorn

Netcetera

http://netcetera.ch/

stefan.tramm@netcetera.ch