Extracting Functional and Nonfunctional Contracts from Java Classes and Enterprise Java Beans

Nikola Milanovic and Miroslaw Malek
`{milanovi,malek}@informatik.hu-berlin.de`

Humboldt University Berlin

Workshop on Architecting Dependable Systems (WADS 2004) at the International Conference on Dependable Systems and Networks (DSN 2004)

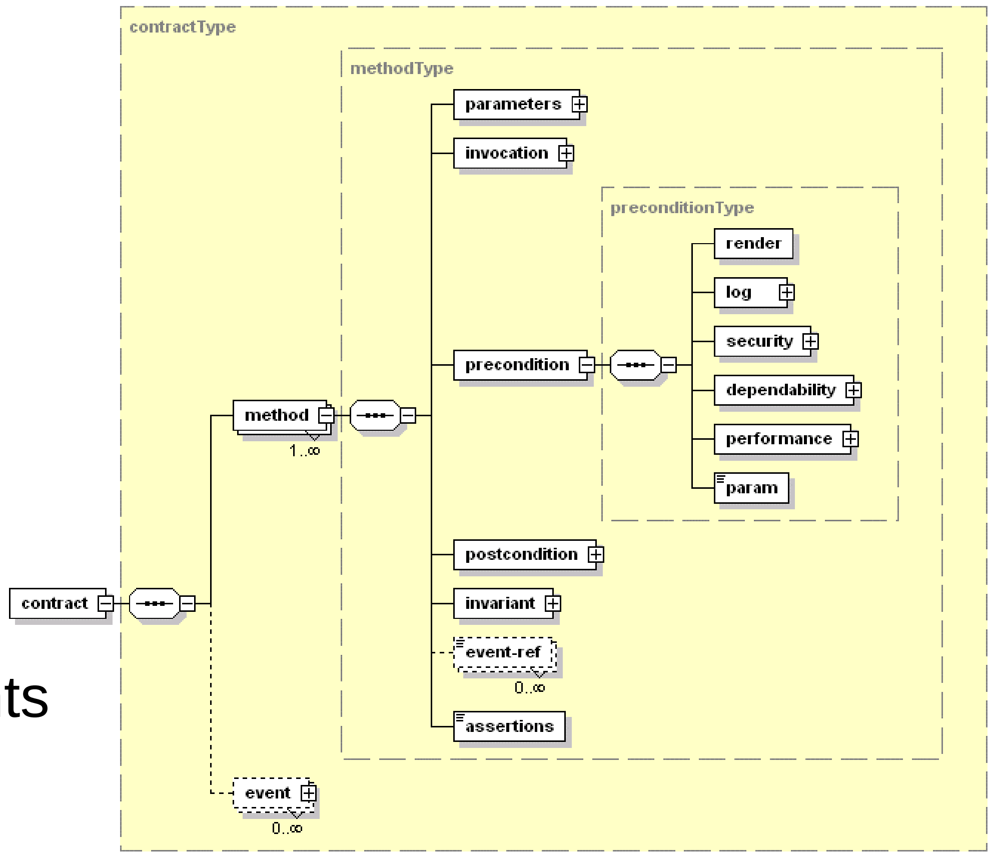
Table of Contents

Introduction

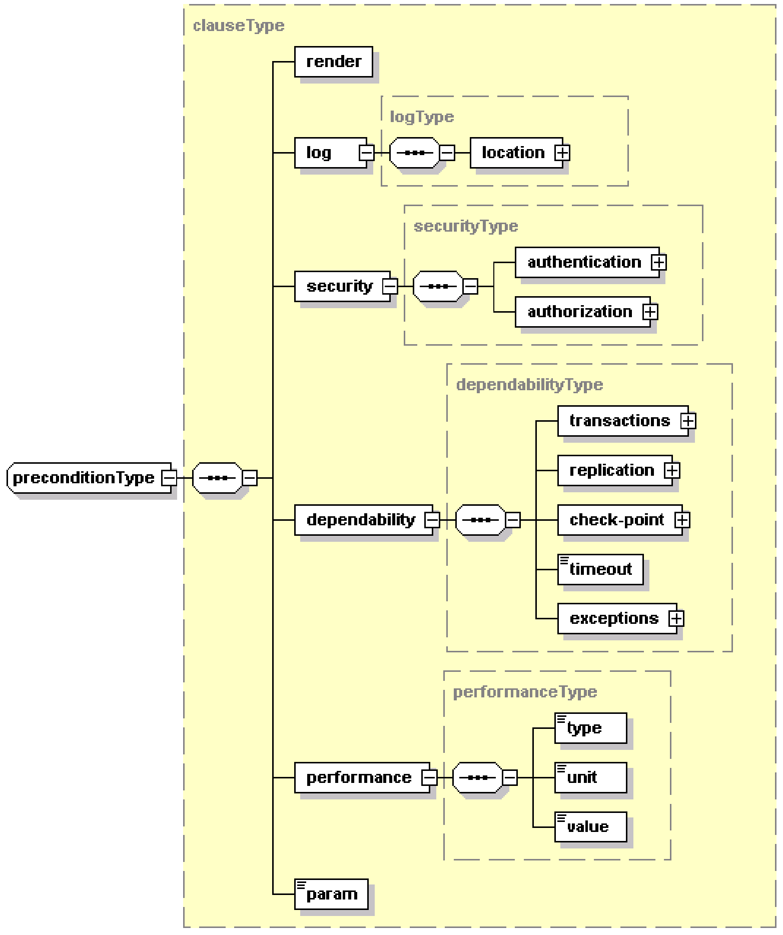
- Software component marketplace:
  - Maturity (applications servers have to mature)
  - Vendor policies (e.g., withholding information)
  - Questionable value
- Is software component marketplace a myth?

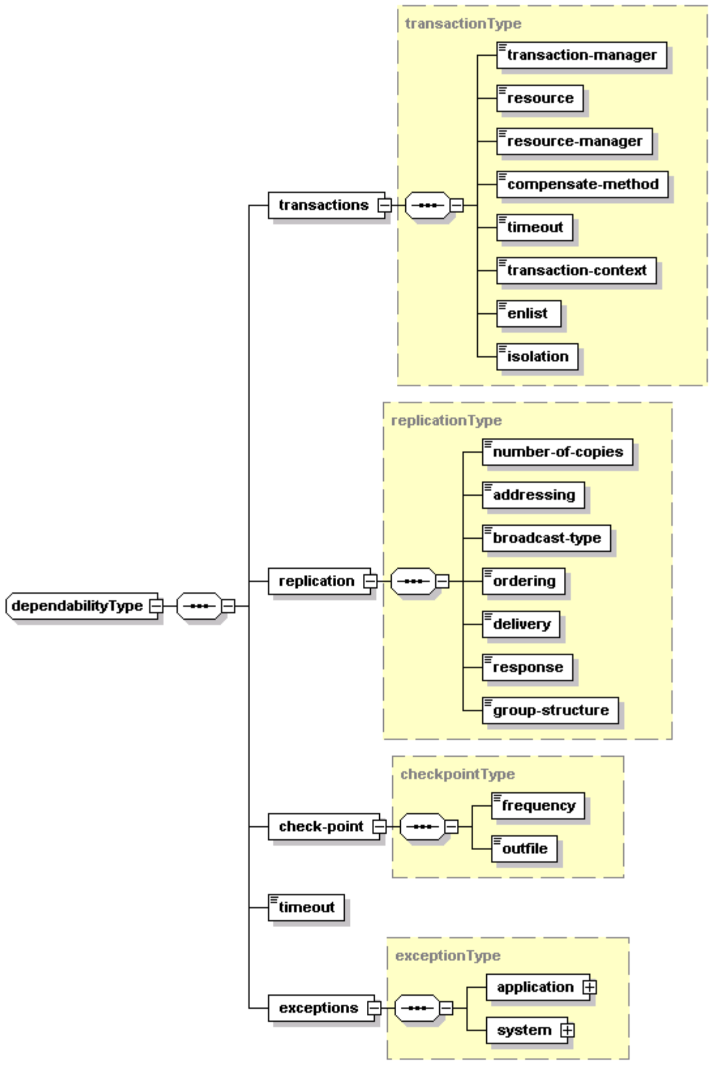
Contract Definition Language (CDL)

- Base, method, event contract
- Preconditions, postconditions, invariants
- Benefits:
  - Reuse
  - Comparing components
  - Correctness



Nonfunctional properties in CDL





Formalized Contracts

- Transferring CDL into formal notation
- Modeling service contract as an abstract machine
  - Statics (state variables)
  - Dynamics (functions)

```
MACHINE M(X,x)
CONSTRAINTS C
CONSTANTS Ct
SETS S
PROPERTIES P
VARIABLES V
INVARIANTS I
ASSERTIONS J
INITIALIZATION U
OPERATIONS
    u1 <- O1(w1) PRE Q1 THEN V1 END
    u2 <- O2(w2) PRE Q2 THEN V1 END
    …
    un <- On(wn) PRE Qn THEN Vn END
END
```

- Proof obligation
  - Assertion:

    $C \wedge P \wedge I \Rightarrow J$
  - Initialization:

    $C \wedge P \Rightarrow [U]I$
  - Operation:

    $C \wedge P \wedge J \wedge Q \Rightarrow [V]I$ *

    * generous specification

Contract Example – a Print Service

```
<?xml version="1.0" encoding="utf-8"?>
  <contract service name="printService"
  serviceURI="localhost/services/print"
  serviceDescription="Basic printing service"
  price="0" state="stateless">

  <method name="Print" methodDescription=
         "Prints a document">

    <parameters>
      <parameter direction ="in">
        <name>doc</name>
        <parameterType>Document</parameterType>
      </parameter>
      <parameter direction="in">
          <name>resolution</name>
          <parameterType>int</parameterType>
      </parameter>
      <parameter direction="out">
        <name>status</name>
        <parameterType>int</parameterType>
      </parameter>
    </parameters>

    <precondition>
      <param>Document.type=ps</param>
      <param>resolution=Printer.res</param>
    </precondition>

    <postcondition>
      <param>Doc.res=resolution</param>
      <performance>
        <type>number-of-concurrent-clients</type>
        <unit>int</type>
        <value>20</value>
      </performance>
     <dependability>
        <transactions model="split">
          <transaction-manager>jrun</transaction-
           manager>
          <resource-manager>/print/drv/file.drv</resource-
           manager>
          <compensate-
           method>printCompensate</compensate-
           method>
          <timeout unit="ms">1000</timeout>
          <enlist>required</enlist>
        </transactions>
      </dependability>
    </postcondition>

    <invariant>
      <param>Documen.pages<=Printer.paper</param>
    </invariant>

  </method>

  <event name="outOfPaper">
    <!-- event definition, similar to method -->
  </event>

</contract>
```

```
MACHINE printer
SETS Document,Resolution,Status,Transaction
VARIABLES doc, resolution, status, printer, ncc,
        trans_model,
trans_manager, compensate, timeout, enlist
INVARIANT doc.pages <= printer.pages
OPERATIONS status <- print(doc)
PRE doc.type = Document.ps AND resolution IN
        printer.res
THEN doc.res=resolution AND ncc=120 AND
        trans_model=Transaction.
SPLIT AND trans_manager=Transaction.JRUN AND
        compensate=
printCompensate AND timeout=Transaction.1000 AND
        enlist=
Trans.REQUIRED END
status <- compensatePrint() END
```

Contract Extraction

- Contracts are not a part of modern software engineering (mainstream languages)
- Mandatory contracts (Eiffel)
- Adding contracts *a posteriori*
  - Identifying locations where to look for hidden specification
  - Establishing algorithms/heuristics for extracting hidden specification

Invariants in Java Classes

- Locations to look for class invariants:
  - Documentation
  - Constructors
  - Implemented interfaces
  - Base class

Invariants in java.util.ArrayList

- Documentation:
  Each `ArrayList` instance has a capacity. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added in `ArrayList`, its capacity grows automatically.
- Constructors:

```
public ArrayList(int initialCapacity)
public ArrayList()
public ArrayList(Collection c)
```

- Exported methods:

```
public void trimToSize() {
  modCount++;
  int oldCapacity = elementData.length;
  if (size < oldCapacity) {
    Object oldData[]=elementData;
    elementData=new Object[size];
    System.arraycopy(oldData, 0, elementData, 0, size);
  }
}
```

- Implemented intefaces: `List,RandomAccess,Serializable,Cloneable`
- Base class: `AbstractList`

Preconditions in Java Classes

- Locations to look for class preconditions:
  - Documentation
  - Conditions in exported methods
  - Exception conditions

Preconditions in java.util.ArrayList

- Conditions in exported methods:

```
public Object set(int index, Object element) {
  RangeCheck(index);
  Object oldValue=elementData[index];
  elementData[index]=element;
  return oldValue;
}

private void RangeCheck(int index) {
  if (index >= size || index <=0)
    throw new IndexOutOfBoundsException("Index: "+index+", Size: "+size);
}
```

- Exceptions:

```
public boolean addAll(Collection c) {
 Object[] a = c.toArray();
 int numNew = a.length;
 ensureCapacity(size + numNew);  // Increments modCount
 System.arraycopy(a, 0, elementData, size, numNew);
 size += numNew;
 return numNew != 0;
}
```

Postconditions in Java Classes

- Locations to look for class postconditions:
  - Documentation
  - Return paths of exported methods

Postconditions in java.util.ArrayList

- Documentation:

  Returns the index of the last occurrence of the specified object in this list; returns -1 if the object is not found.

- Return paths:

```
public int indexOf(Object elem) {
  if (elem == null) {
    for (int i = 0; i < size; i++)
      if (elementData[i]==null)
    return i;
  }
  else
  {
    for (int i = 0; i < size; i++)
      if (elem.equals(elementData[i]))
    return i;
  }
  return -1;
}
```

Javadoc Tags in Contract Extraction

- `@throws`, `@exception`: extracting preconditions and invariants (if constructor is commented)
- `@param`: extracting method signature, preconditions, and invariants (if constructor is commented)
- `@return`: extracting postconditions (constructors not commented)
- `@see`: tracking inheritance, conflicting requirements

Extracting Contracts from Enterprise Java Beans

- Types of Enterprise Java Beans:
  - Session (e.g., business processes, flows)
  - Entity (e.g., data models, database views)
  - Message-driven
- Functional properties
- Nonfunctional properties

Preconditions, Postconditions, Invariants in EJBs

- Locations to look at:
  - `ejbCreate` and other CRUD (create, read, update, delete) methods
  - Setter methods
  - Primary key classes
  - Finder methods
  - Deployment descriptors

Guidelines for Different Bean Types

- Session beans:
  - Stateful – `ejbCreate`
  - Stateless – `ejbCreate` does not accept parameters
- Entity beans:
  - ejbCreate calls setters

```
public Object ejbCreate(CreditCard creditCard) throws CreateException {
   setCardNumber(creditCard.getCardNumber());
   setCardType(creditCard.getCardType());
   setExpiryDate(creditCard.getExpiryDate());
   return null; }
```

  - Bean managed persistence – setters
  - Container managed persistence – setters are abstract, SQL or EJB-QL

```
<operation>createTable</operation>
   <sql>
     CREATE TABLE "CreditCardEJBTable" ("__PMPrimaryKey" LONGINT ,
     "__reverse_creditCard___PMPrimaryKey" LONGINT , "cardNumber"
     VARCHAR(255) , "cardType" VARCHAR(255) , "expiryDate" VARCHAR(255),
     CONSTRAINT "pk_CreditCardEJBTabl" PRIMARY KEY ("__PMPrimaryKey") )
   </sql>
```

EJB Exception Hierarchy

- System level exceptions:
  - Remote exception
  - Network failure, database failure, special error
  - Sometimes not propagated to client
- Application level exceptions:
  - Regular problems (e.g., bad parameters)
- When extracting preconditions, we check for application exceptions only (propagated to client) + `javax.ejb.CreateException` and `javax.ejb.FindException`

Contracts in Message-driven Beans

- Only one, weakly typed method `onMessage`
- No return values (completely decoupled from the client) – no way to infer postconditions
- No exceptions can be sent to clients (prohibited)
- Message-driven beans require different extraction techniques as a consequence of asynchronous model they implement
- Since message-driven beans are not true and general asynchronous model (like asynchronous RMI or MS Queued Components), we do not cover them here as too specific and outside of scope

Nonfunctional Properties in EJBs

- CDL defines following nonfunctional properties:
  - Invocation
  - Security (authentication, authorization)
  - Dependability (transactions, checkpointing, replication, exceptions)
  - Performance
  - Rendering
  - Logging
- Current J2EE specification defines only bean management (lifecycle), persistence, transactions and security
- Other properties (load-balancing, clustering, logging) are vendor specific

Bean Invocation

- We use bean management information to form invocation part of a contract:
  - Remote, local, home interfaces
  - Temporal behavior
  - State management
  - Persistence management

Bean Transactions

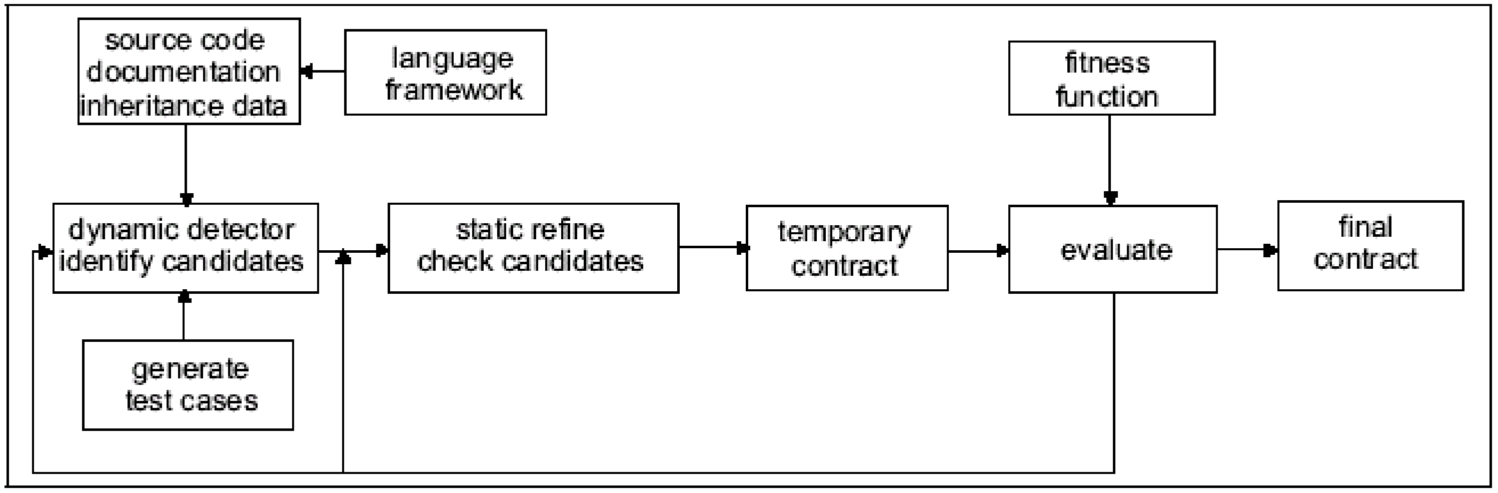
- CDL `transactionType` picks data from deployment descriptor:
  - Transaction manager (current J2EE container)
  - Resource (e.g., database)
  - Resource manager (e.g., database driver)
  - Compensation method (split transactions)
  - Timeout
  - Enlist (`trans-attribute`)
  - Isolation
    - No information in deployment descriptor
    - For bean managing transactions – `java.sql.Connection.SetTransactionIsolation()`
    - Otherwise, application server configuration files

Bean Security

- For the authentication purpose, contract stores information about:
  - Configuration module
  - Login module(s)
- For the authorization, contract stores security roles:
  - Declarative authorization: reading `security-role-ref`, `role-name` and `role-link` elements.
  - Programmatic authorization: scanning source for `getCallerPrincipal()` and `isCallerInRole(roleName)` methods

Static and Dynamic Extraction

- Dynamic detector
  - Runtime analysis
  - Efficient, but not general
- Static refine
  - Building a model of execution state
  - Theoretically complete, but inefficient


source code documentation inheritance data
language framework
fitness function
dynamic detector identify candidates
static refine check candidates
temporary contract
evaluate
final contract
generate test cases

iContract: Design by Contract in Java

- Adding Design by Contracts assertions (preconditions, postconditions, invariants) to Java code
- iContract is a preprocessor for Java
- @pre, @post, @invariant directives

```
/**
* @pre f>= 0.0
* @post Math.abs((return*return)-f)<0.001
*/
public float sqrt(float f) {…}

/**
*@invariant forall IEmployee e in getEmpolyees() |
   getRooms().contains(e.getOffice())
*/
```

- More information: `http://icplus.sourceforge.net/`

A Look Ahead: Component Composability

```
MACHINE creditRating
VARIABLES loanApplication(IN):invariant; creditRating(OUT):vanish
INVARIANT loanApplication ∈ Application:invariant
OPERATIONS rating(SYNC) ≜ PRE THEN creditRating=rating(loanApplication):transfer;
wcet=300ms:vanish; ncc=10:bounded
EXCEPTIONS invalidApplication:transfer
```

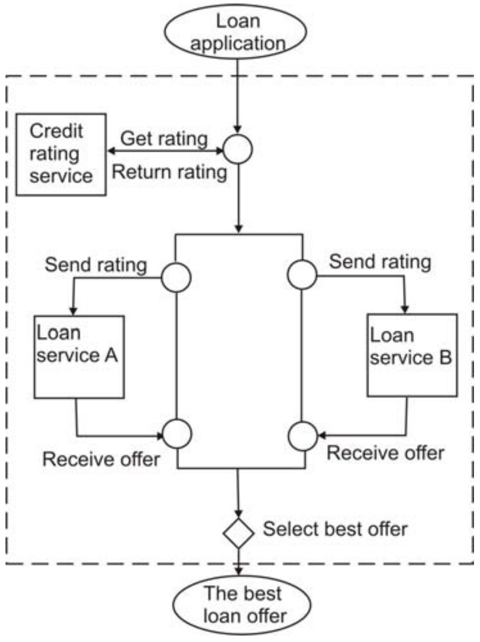
```
MACHINE loanCompany
VARIABLES creditRating(IN):vanish loanOffer(OUT):invariant interestRate(OUT):invariant
INVARIANT loanOffer ≥ 0
OPERATIONS offer(ASYNC) ≜ PRE THEN loanOffer=offer(creditRating):transfer;wcet=24h:vanish;
ncc=1000:bounded;
EXCEPTIONS invalidRating:transfer noLoan:invariant
```

$$\texttt{newService} = \texttt{creditRating} \times \textit{switch}(\texttt{loanCompanyA}, \texttt{loanCompanyB})_{\textit{min}(\texttt{interestRate})}$$

```
MACHINE newService
VARIABLES loanApplication(IN):invariant loanOffer(OUT): invariant
INVARIANT loanApplication ∈ Application loanOffer ≥ 0
OPERATIONS bestLoan (ASYNC)≜ PRE THEN loanOffer=bestLoan(loanApplication):transfer;ncc=10
EXCEPTIONS invalidAppRating:transfer noLoan:invariant
```

	exception	operation	ncc
transfer	∪	SEQUENCE	+
bound	/	/	min
NO_MATCH	/	/	/

$\texttt{SEQUENCE} = \textit{rating} \circ \textit{offer}$



Conclusion

- Contracts improve reuse and facilitate comparison
- We model functional and nonfunctional properties
- Improving software dependability:
  - Design phase
  - Early development phase
  - Reuse
- Addressing problems of composition correctness
- In many stages automated extraction is not possible at this time
- Specifying contracts during design and development time is of the outmost importance