

# TAKE – A Derivation Rule Compiler for Java

Jens Dietrich, Massey University  
Jochen Hiller, TopLogic  
Bastian Schenke, BTU Cottbus

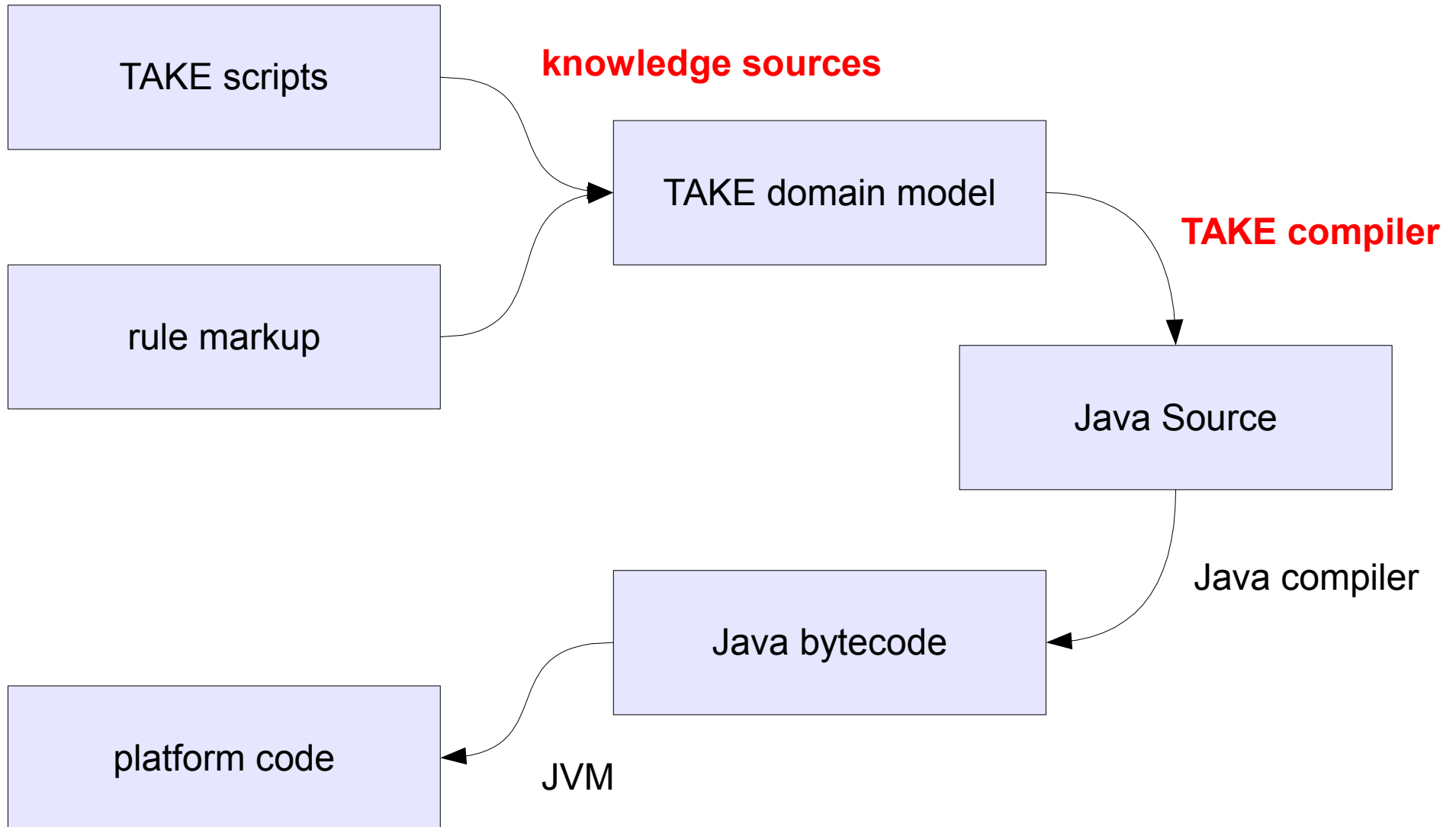
# Why Derivation Rules

- Query driven.
- Well defined semantics.
- Non state changing.
- Easy integration into pull based applications (http requests, DB queries).
- No need to replicate (keep and keep up-to-date) facts in memory – fetch as needed, cache if you want
- Good scalability in scenarios with large fact bases and small or medium sized (up to thousands) rule bases.

# Rule Engines vs Rule Compilers

- Rule Engine: interpret rules at runtime.
- Rule compiler: convert rules into executable code.
- Advantages: speed + free verification by underlying (Java) compiler.
- Disadvantage: compilation happens at build time - application must be redeployed if rules change.
- But: runtime compilation and deployment is possible and supported by standards (JSR-199).
- Example: web servers supporting server pages.

# Compilation



# Deployment

## **Option1**

1. Compile rules into classes
2. Build application
3. Deploy application

## **Option1**

1. Compile rules into interfaces
2. Build application
3. Deploy application
4. Compile rules into classes
5. Deploy rules into running applications

# Writing Rules

- TAKE scripting language – easy to use, slightly prologish
- TAKE R2ML adapter (beta) – imports rules from REVERSE R2ML markup language

# TAKE scripts

- Java type references
- Java object references (by name, must be bound)
- global (rule base) and local (rule) annotation
- Aggregations
- Queries

# TAKE example

// example

@@dc:creator=jens dietrich

@@dc:date=27/05/2007

var example.nz.org.take.compiler.eurent.RentalCar car

var example.nz.org.take.compiler.eurent.Rental rental

var example.nz.org.take.compiler.eurent.Branch branch

@take.compilerhint.class=IsAvailable

@take.compilerhint.slots=car,branch

@take.compilerhint.method=IsAvailable10

query availableAt[in,out]

// rules

rule1: if storedAt[car,branch] and not isScheduledForService[car,branch]

and not assignedTo[car,rental] then availableAt[car,branch]

annotations  
(meta data)

variables,  
types are  
Java classes

more annotations,  
names used by  
compiler

query, defines the public  
methods to be generated

rule



# Generated Code

- Generates simple PLOJO classes for predicates.
- Central main class with interface for queries.
- Actual code in fragment classes referenced by main query class for scalability.
- Call backs to Java methods in domain model.
- Large parts of the generated code are template based.

# Generated Data Structure

- PLOJOs are generated for all referenced predicates

```
public class IsAvailable {  
    public example.nz.org.take.compiler.eurent.RentalCar car;  
    public example.nz.org.take.compiler.eurent.Branch branch;  
  
    public IsAvailable() {  
        super();  
    }  
}
```

# The Resource Iterator Pattern

- Iterator - GangOf4 pattern – iterate over collections without knowing their internals
- Resource Iterator – Iterator + close – suitable to iterate over resources not necessarily available in memory (fetch as needed)
- Performance can be boosted by caching and background prefetching if needed
- Structure:
  - hasNext() - is there more?
  - next() - get next item
  - close() - close connection to source supplying items

# External Fact Stores

- Describe where facts are supplied by external resources like databases and web services.
- Compiler will generate interface that produces iterators.
- Interfaces can be implemented to access facts.
- Bound to implementation when compilation takes place.

# Query API

- Iterator based– pull results as needed.
- (Asynchronous) prefetching can be achieved by wrapping.
- Iterators have additional close method. Semantics: release external resources (network + db connections)
- Iterators are called “ResultSets”
- result sets have API to extract rules used

# Generated Query API

```
public class KB {
```

name can be set in annotation

```
    public Map<String, String> getAnnotations(String id) {...}  
    public Map<String, String> getAnnotations() {...}
```

runtime access  
to annotations

```
    public ResultSet<IsAvailable> isAvailable10(RentalCar car) {
```

```
        ..  
    }
```

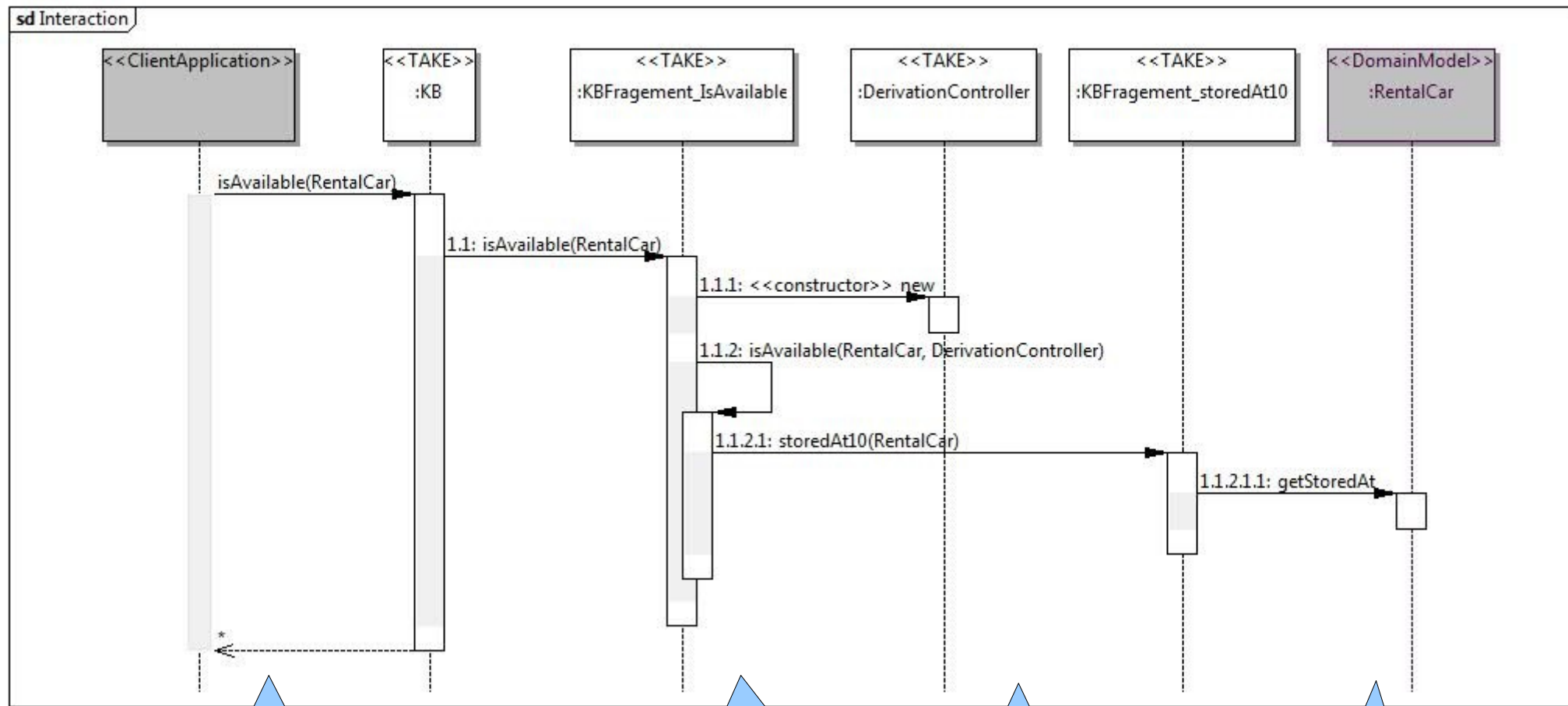
```
}
```

method generated for query, ResultSet  
is iterator

# Generated Code

- Use Iterator “algebra”
- Multiple rules supporting the same query – use chained iterator
- Evaluate prerequisites within a rule – use nested iterators
- Single supporting fact – use singleton iterator
- false – use empty iterator

# Generated Code (ctd)



public query

TAKE internals - clustering

wrapper

domain model  
call backs



# Semantic Reflection

- Generated code is reflective
- Information about rules used to generate application is accessible at runtime
- Similar to reflection is OO – but reflection is only about syntax
- API keeps not rules but only rule ids, and supports querying meta data by id
- examples: creator, date modified, description .. standard vocabularies (DC) should be used
- addresses many business use cases, such as tracing code to requirements
- alternative to dominating black box principle

# Example: Application API

```
RentalCar c = ...;
Branch b = ...;
Rental r = ...;
KB kb = new KB();
ResultSet<IsAvailable> rs = kb.isAvailable10(c);
IsAvailable result = rs.next();
List<DerivationLogEntry> log = rs.getDerivationLog();
System.out.println(result.branch.getName());
for (DerivationLogEntry e:log) {
    Map<String,String> annotations = kb.getAnnotations(e.getName());
    if (annotations!=null){
        System.out.println(annotations.get("take.auto.date"));
        System.out.println(annotations.get("take.auto.creator"));
        System.out.println(annotations.get("take.auto.toString"));
    }
}
```

# Validation

- UServ Product Derby Case Study
- execute by clicking on link on TAKE home page  
<http://code.google.com/p/take/>
- 69 rules -> 111 source code files, 354 classes
- Compilation takes 2567 ms on a system with a 2.0GH T5600 dual core processor, 2.0GB of RAM running on Ubuntu 7.10 with Java 1.6.
- Handicaps: logging and code pretty printing enabled

# Using TAKE

Project Home:

<http://take.googlecode.com/svn/trunk/take/>

Subversion:

<http://take.googlecode.com/svn/>

Example App (WebStart):

<http://www-ist.massey.ac.nz/jbdietrich/userv/userv.jnlp>

License: LGPL