A Database-Based Service for Handling Logical Variable Bindings

Wolfgang May

Institut für Informatik, Universität Göttingen, Germany

may@informatik.uni-goettingen.de

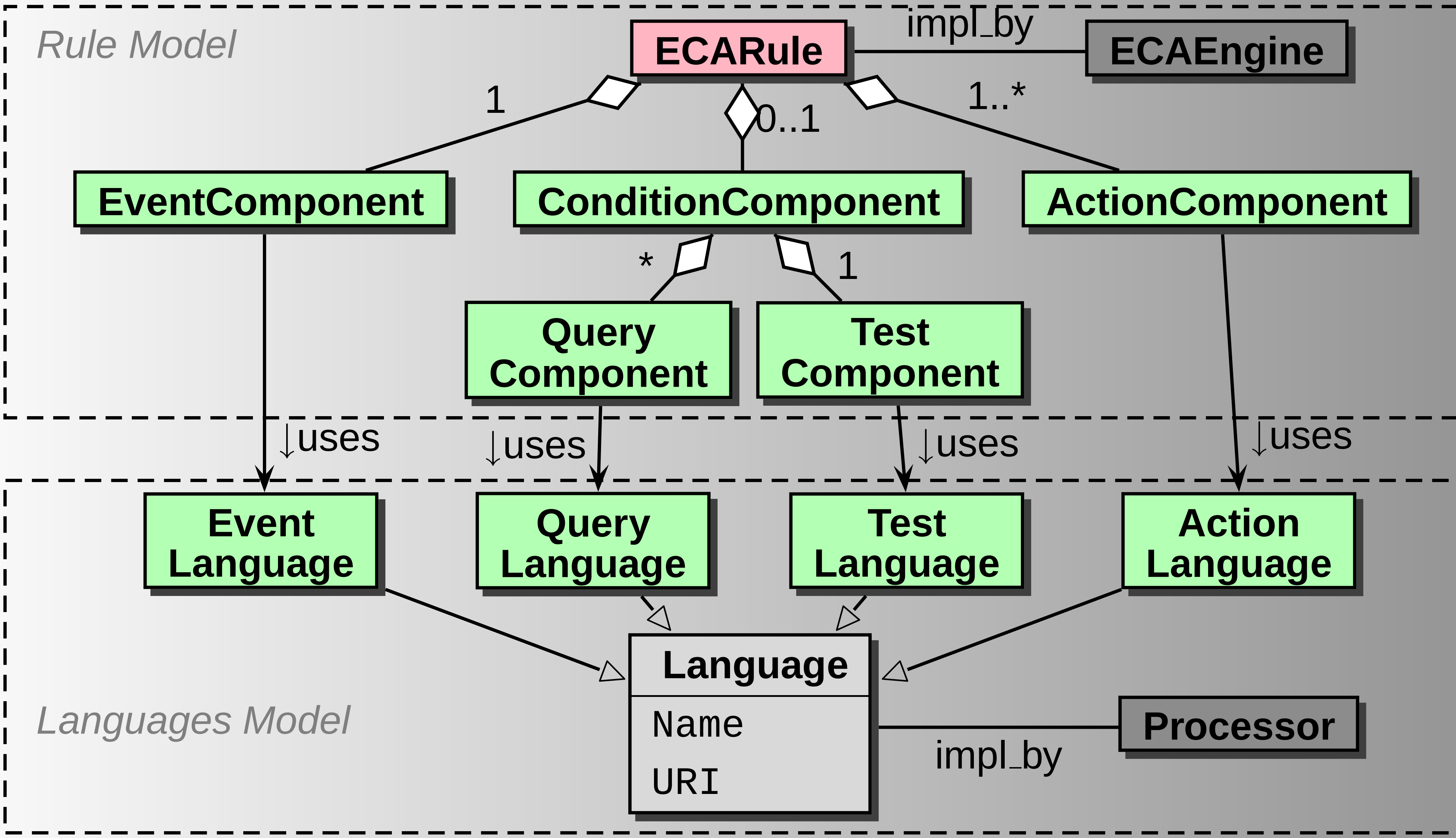
DaaS 2009, Münster, Germany
March 3rd, 2009

MARS

Modular Active Rules on the Semantic Web

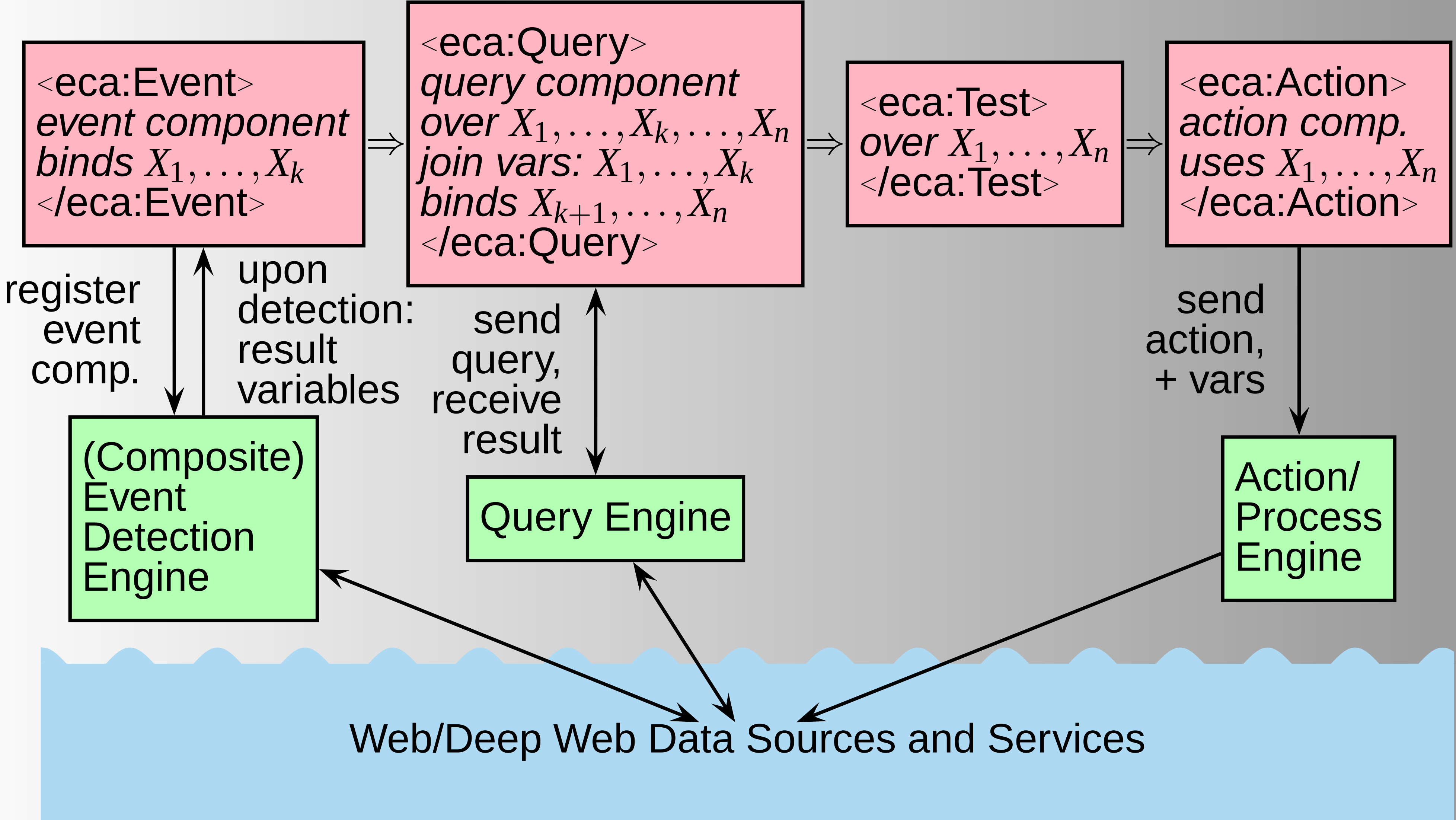
- Rule-based description of behavior in the Semantic Web
- Paradigm: ECA Rules
  "On Event check Condition and then do Action"
- subontologies/-languages for specifying *Events*, *Conditions*, *Actions*,
- modular, declarative specification
- data flow by logical variables (i.e., sets of tuples ...)
- services that implement these sublanguages.
- analogous/(sub)language: CCS with relational data flow

Modular ECA Concept: Rule Structure



Binding and Use of Variables in ECA Rules

$$action(X_1,\ldots,X_n) \leftarrow$$
$$event(X_1,\ldots,X_k),\ query(X_1,\ldots,X_k,\ldots X_n),\ test(X_1,\ldots,X_n)$$



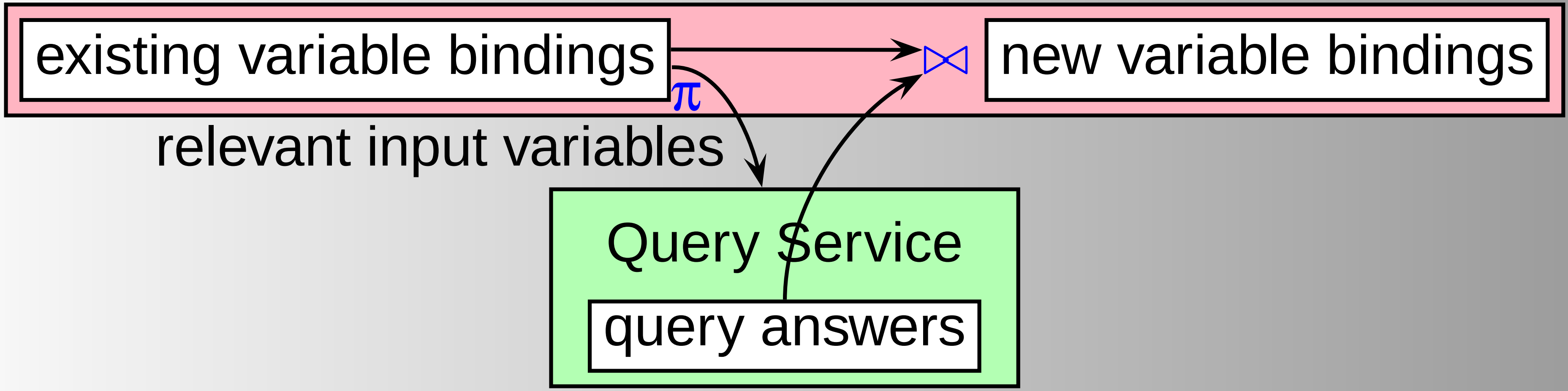
Control Flow and Communication

Control flow moves from one processor/service to the other.
Data exchange:

- the respective fragment
- (projection of) the current set of variable bindings

Answers:

- usually a set of variable bindings ($\rightarrow$ join)



Requirements

- ECA rules: small number of tuples
- (query) workflows (specified in CCS with relational data flow), e.g. Deep Web querying: high number of tuples
- Data exchange between services located at different locations
  closer look: some services (ECA, CCS) do not actually need to have the variable bindings locally, but only to *apply operations on them*!
- operations: projection, join
  clone (concurrent execution)
  delete (exclusive guarded nondeterministic alternative)
- dynamic schema (variable names + datatypes)

Implementation Alternatives

- a Java class VariableBindings based on in-memory data structure
  - actual exchange of data, working locally,
  - sufficient for small amounts of data.
- VariableBindings class uses a local database via JDBC.
  - still actual exchange of data, working locally,
  - requires local DB installation.
- ... or use a *Database as a Service*:

Variable Bindings Service (VBS)

- Separate (remote) Variable Bindings Service (VBS),
- provides the interface of the abstract datatype and uses *its* own database,
- no data exchange, but cooperation on the data in the database,
- VariableBindings class is then only a stub that forwards its methods to the Web Service.

Polymorphism/Separation of Tasks

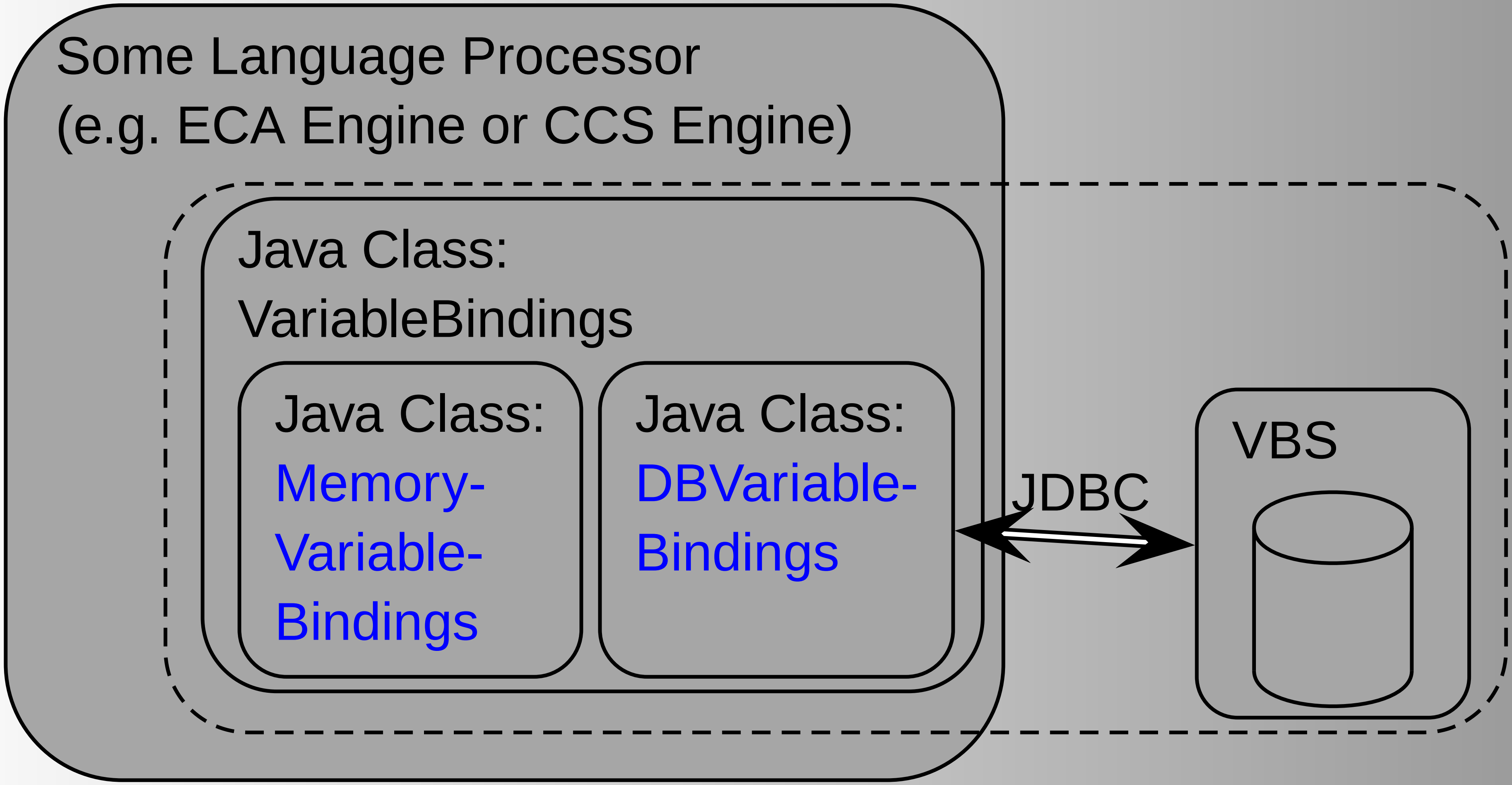
Choice and dynamic, transparent switching between MemoryVarBindings and DBVarBindings:

- schema/variable information maintained preferably in the Java part.
- Java: not as subclasses (instance cannot change class membership), but class with delegation.
- VariableBindings provides the common functionality (metadata management, API of the abstract datatype)
- data member myVariableBindings that is either an instance of MemoryVariableBindings or of DBVariableBindings
- these are implementations of an interface VariableBindingsImpl that is *only* concerned with the storage issue and the actual operations.

Final Design: DBVariableBindings

- access to actual VBS by DBVariableBindings via JDBC
- SQL statements dynamically generated and submitted to database.
- service config:
  - ± DB available, optional: preferred “own” JDBC URL/user/passwd
  - threshold: size for switching to DB-VB
- services can also (read-)access foreign VBS via JDBC (communicate URL/user/passwd)

Final Design



DBVariableBindings

- attributes mytablename, jdbcURI/user/pwd
- DBVariableBindings())
  DBVariableBindings(MemoryVariableBindings vb)
  addTuple(Tuple t):
- unary relational operations:
  - projection: ALTER TABLE *t* DROP COLUMN *variable*,
  - selection: DELETE FROM *t* WHERE NOT *selection condition*,
  - bindInAllTuples(name, value): adds a new variable with a given value to all tuples:
    ALTER TABLE *t* ADD *var datatype* DEFAULT *value*;
    ALTER TABLE *t* MODIFY COLUMN *var* DEFAULT NULL;
- iterator getTuples() (VarBindings implements Iterable).

Binary Relational Operations

- preparing step before (switch both to DB or both to Memory),
- transparent for the outside service
- natural join: create a new table that contains the result:

  INSERT INTO t_{new}

  (SELECT * FROM t, t' WHERE *equality of all shared variables*)

  and set mytablename to the new table name;
- union: add tuples given in main memory representation, or the contents of a complete table,
- minus (as removeMatchingBindings(other)):

  DELETE FROM t WHERE EXISTS

  (SELECT * FROM t' WHERE *equality of all shared variables*);

Distributed Usage

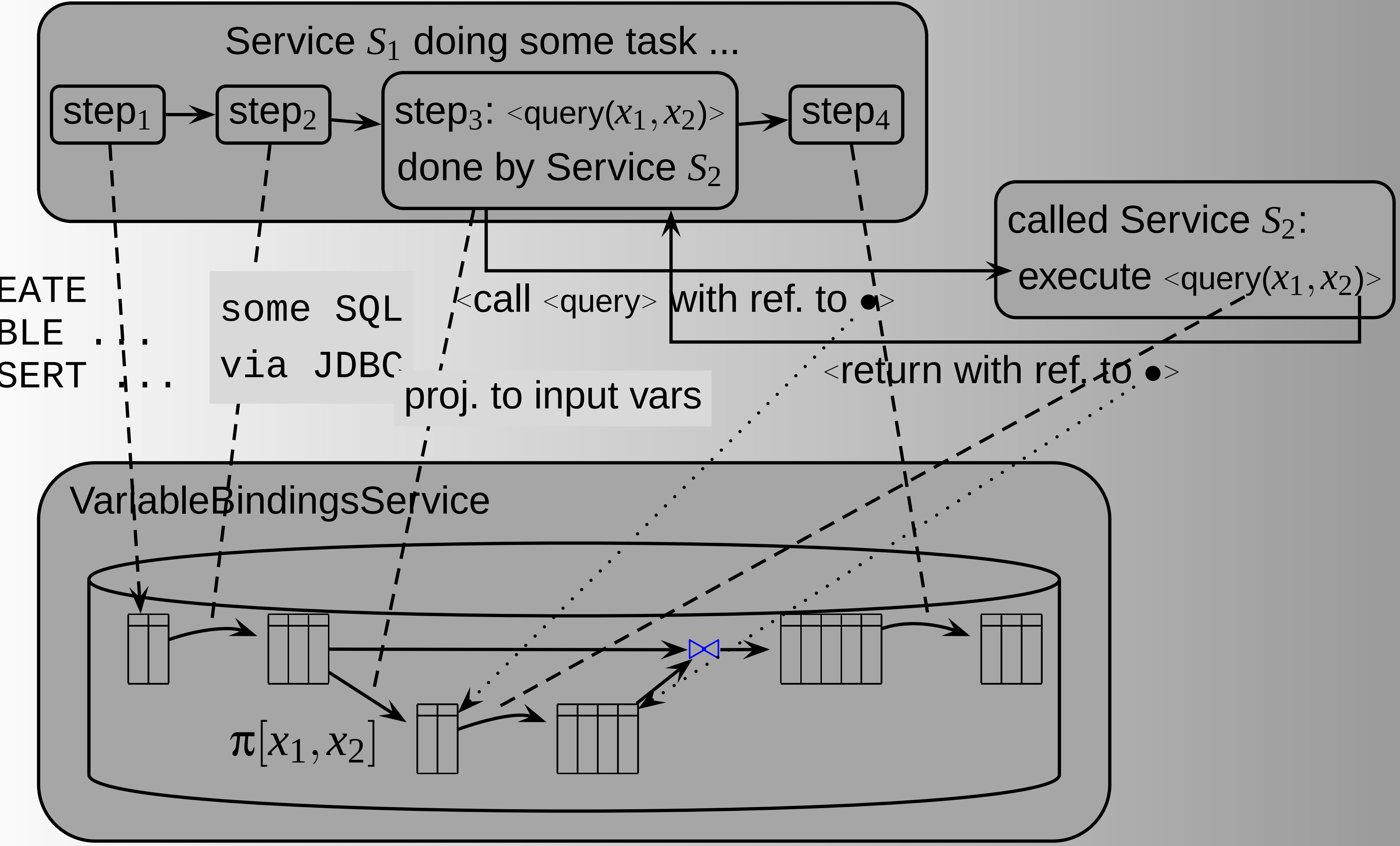
- communication of variable bindings: exchange reference to VBS

  <variable-bindings database="$f\left(\textit{jdbc-uri, user, password}\right)$"

  tablename="*tablename*"/>

- constructor from the above XML communication format (creates just a "small" DBVariableBindings stub instance)

Distributed Usage



Some SQL Details

Case-Sensitive and Other Column Names

- SQL column names usually case-insensitive.
- variable names usually case-sensitive, all symbols allowed
  column names:
  CREATE TABLE MARS_VARS_xyz (“*varname*” VARCHAR2(20));

SQL XMLType: No Comparison, no Join

- use MemoryVariableBindings

Related Work

Frequently Asked Question:
What about Tuple Spaces/TSpaces?

- unstructured set of tuples of arbitrary schema
  MARS: sets of homogeneous relations
- insert, associative access (read, delete)
- no support for operations on relations/sets

Summary

- Database for storing and manipulating sets of tuples of variable bindings
- Access by JDBC

MARS Demonstrator

http://www.semwebtech.org/mars/frontend/

Thank You

Questions?