Towards Knowledge-Intensive Software Engineering

Samuel Cauvin Derek Sleeman Wamberto Vasconcelos

Dept. of Computing Science, University of Aberdeen, Aberdeen, U.K.
s.cauvin@abdn.ac.uk

ICSOFT-EA 2015, Colmar, France

Outline

Content

Introduction

Motivation:

- There is a substantial disconnect between software and domain knowledge
- Often when developing a system, there already exists an ontology in the same domain
- Not making use of this ontology can create inconsistencies

Proposal:

- Combine code and ontologies with a matching system
- Matches shown to the user with reasons, allowing user input
- Developed a tool called Facilitator

Current Landscape

Attempts to integrate ontologies and software development:

- A tool, TwoUse combines UML and Ontologies
- A second tool, RDFReactor creates Java classes from an Ontology
- Another tool finds an ontology for code and then formally relates them together

Content

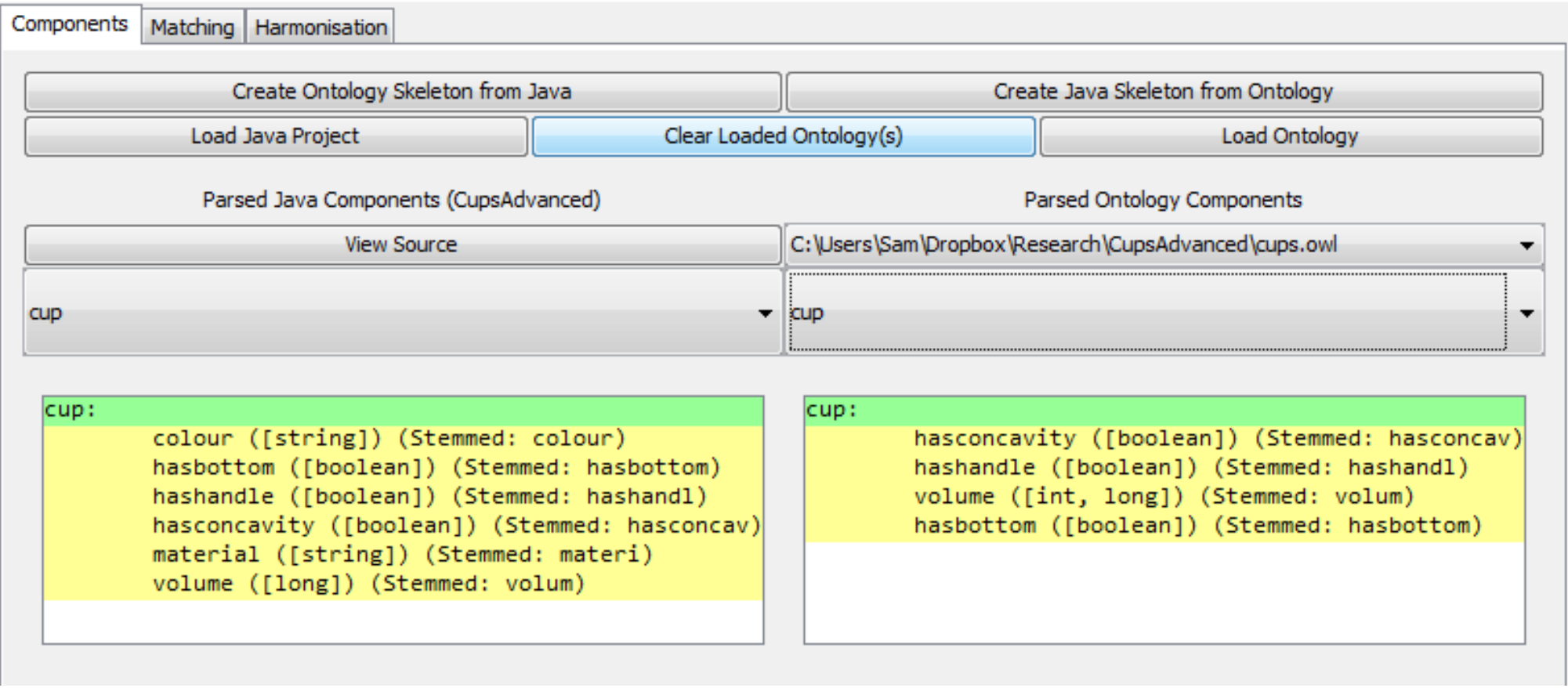
Facilitator Outline

Facilitator's main functionalities:

- Finds matches between a Java project and ontologies
- Displays matches in several ways to the user
- Creates Java projects from ontologies
- Creates ontologies from Java projects

Facilitator uses an ontology reasoner for some matches.

Screenshot (Parsed Components)



Screenshot (Match Results)

Components | Matching | Harmonisation

Run Matching | Clear Matches

Simple | Java Source Overlay | Report | Stats

Matching Results (CupsAdvanced versus cups)

Class **cup** (Java) might match Class **cup** (Ontology) because: Names Match, All Fields Match
- Field **hasbottom** [boolean] (Java) might match Field **hasbottom** [boolean] (Ontology) because: Names Match, Types Match Directly
- Field **hashandle** [boolean] (Java) might match Field **hashandle** [boolean] (Ontology) because: Names Match, Types Match Directly
- Field **hasconcavity** [boolean] (Java) might match Field **hasconcavity** [boolean] (Ontology) because: Names Match, Types Match Directly
- Field **volume** [long] (Java) might match Field **volume** [int, long] (Ontology) because: Names Match, Types Match Directly

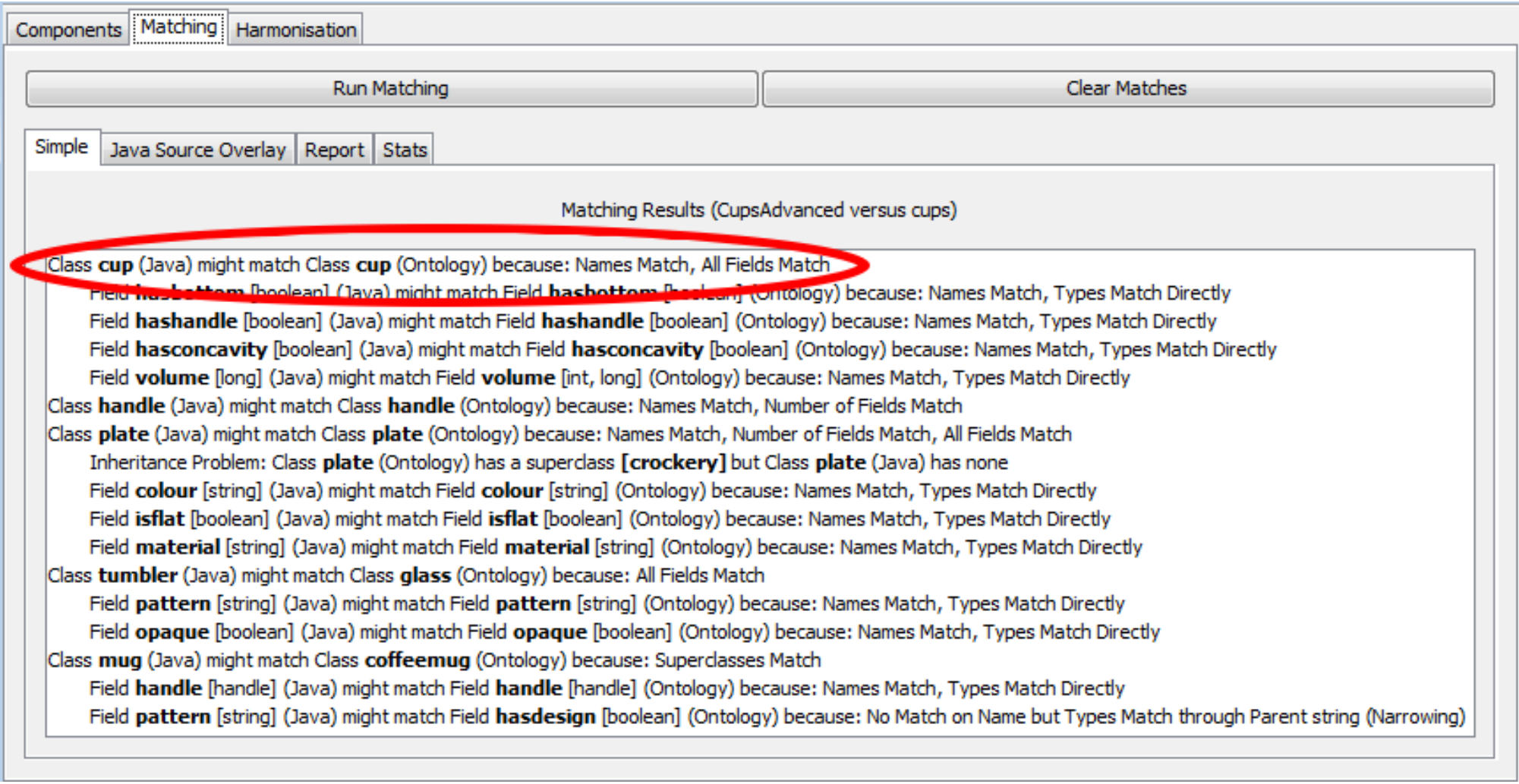
Class **handle** (Java) might match Class **handle** (Ontology) because: Names Match, Number of Fields Match

Class **plate** (Java) might match Class **plate** (Ontology) because: Names Match, Number of Fields Match, All Fields Match
- Inheritance Problem: Class **plate** (Ontology) has a superclass **[crockery]** but Class **plate** (Java) has none
- Field **colour** [string] (Java) might match Field **colour** [string] (Ontology) because: Names Match, Types Match Directly
- Field **isflat** [boolean] (Java) might match Field **isflat** [boolean] (Ontology) because: Names Match, Types Match Directly
- Field **material** [string] (Java) might match Field **material** [string] (Ontology) because: Names Match, Types Match Directly

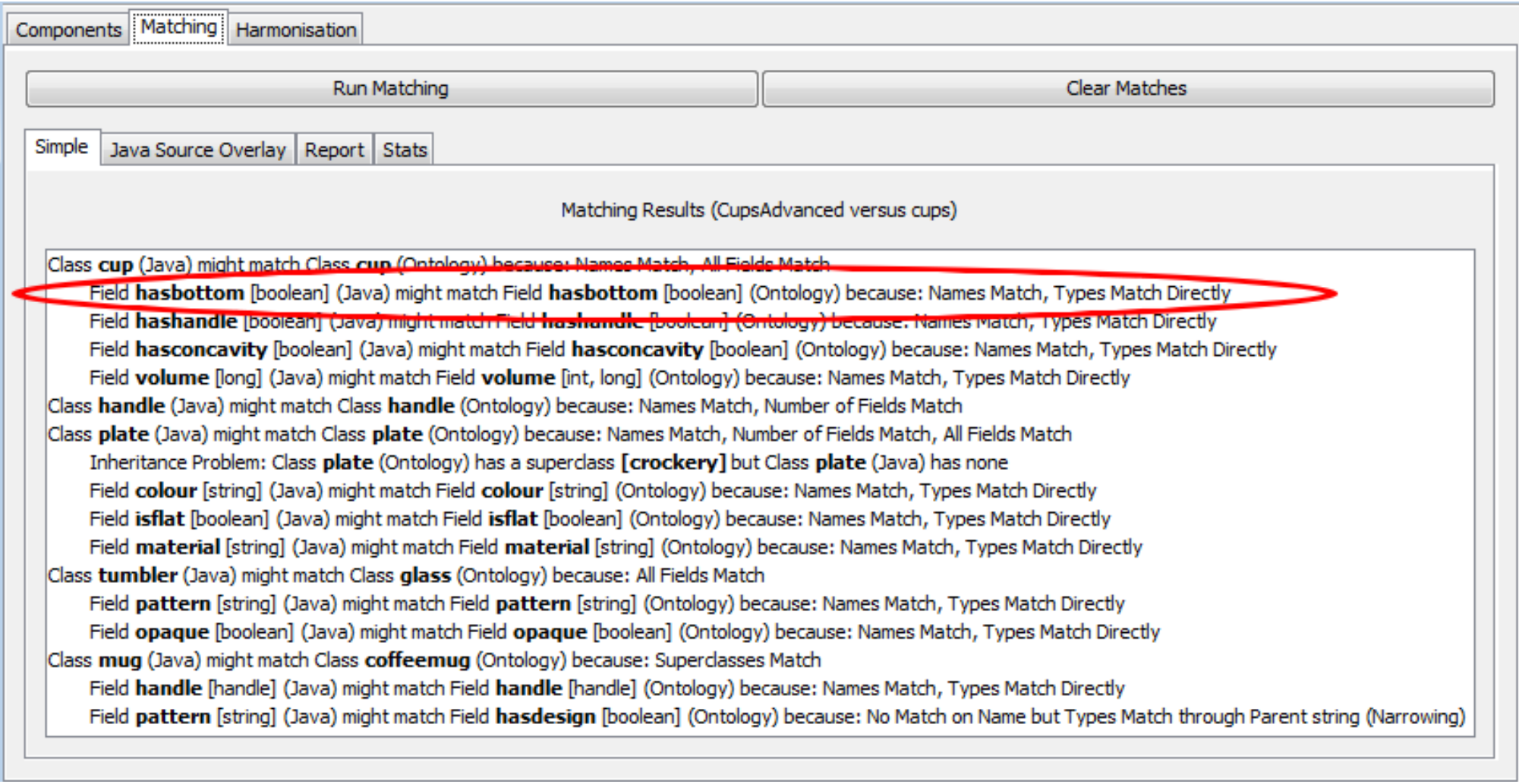
Class **tumbler** (Java) might match Class **glass** (Ontology) because: All Fields Match
- Field **pattern** [string] (Java) might match Field **pattern** [string] (Ontology) because: Names Match, Types Match Directly
- Field **opaque** [boolean] (Java) might match Field **opaque** [boolean] (Ontology) because: Names Match, Types Match Directly

Class **mug** (Java) might match Class **coffeemug** (Ontology) because: Superclasses Match
- Field **handle** [handle] (Java) might match Field **handle** [handle] (Ontology) because: Names Match, Types Match Directly
- Field **pattern** [string] (Java) might match Field **hasdesign** [boolean] (Ontology) because: No Match on Name but Types Match through Parent string (Narrowing)

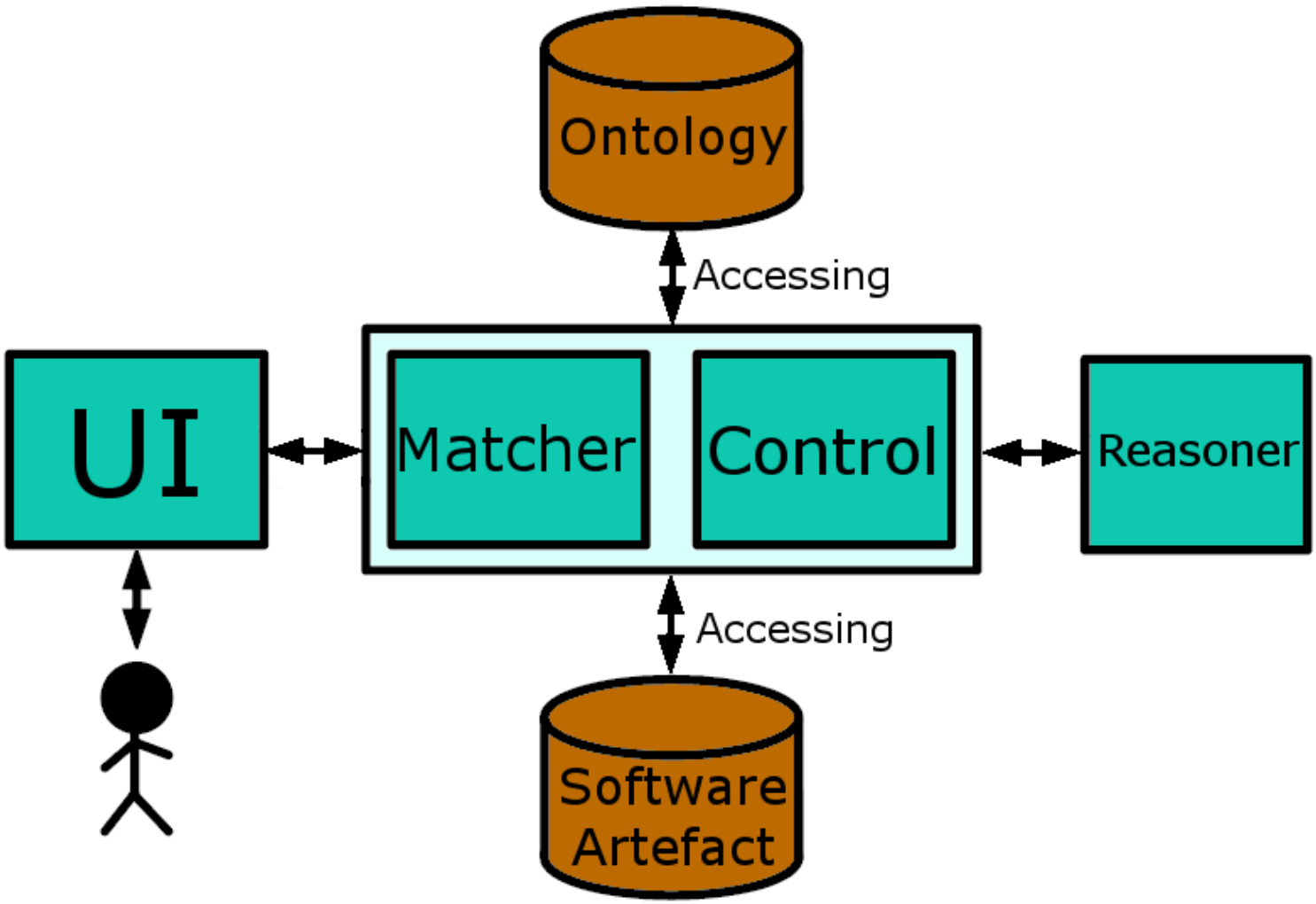
Screenshot (Match Results)



Screenshot (Match Results)

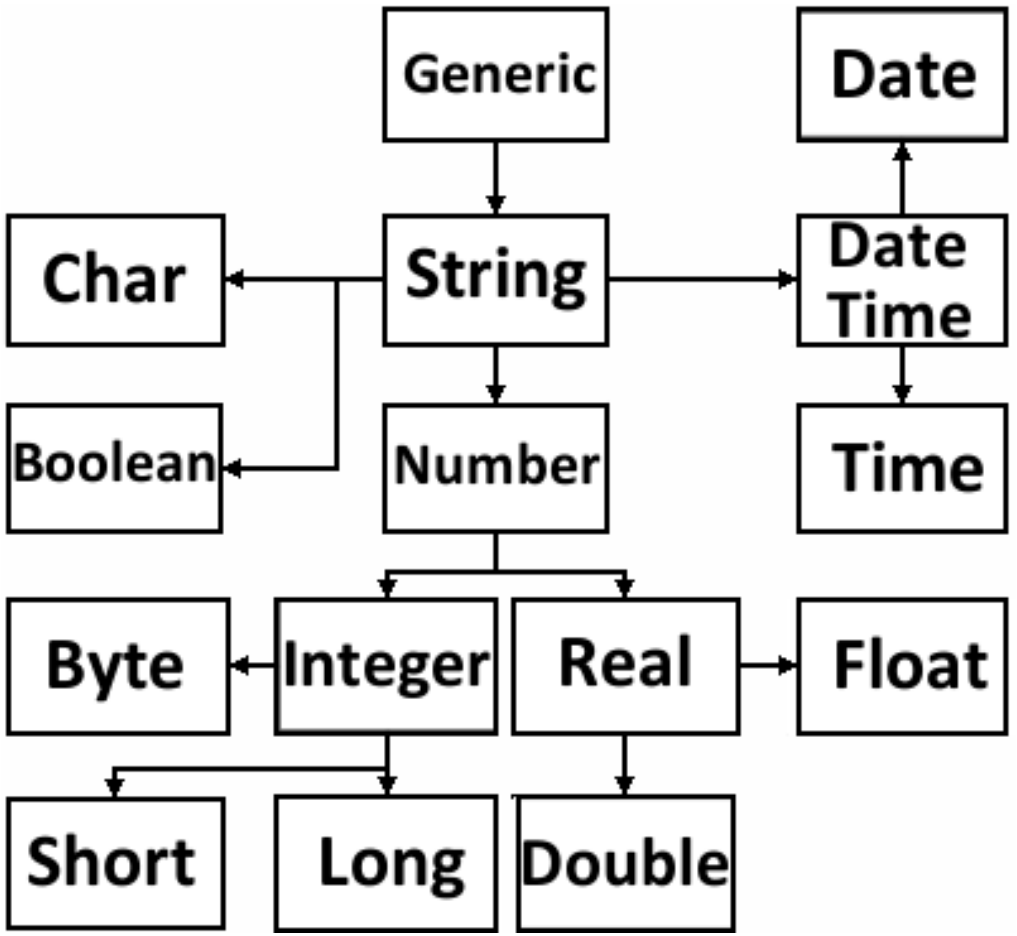


Architecture



Taxonomy of Types

Facilitator compares data types using a taxonomy of types



Content

Matching Algorithm

Matching algorithm has three main stages:

1) Detecting class matches between class and field names
2) Detecting class matches using superclass relationships
3) Detecting field matches, with three sub stages:
   a) Detecting field matches by name and type
   b) Detecting field matches using inferred fields
   c) Detecting field matches by type but not name, exclusively using previously unmatched fields

Illustrative Scenarios: Cars Example

Java	
Car	String colour
	int wheels
	Engine engine
Engine	int horsepower
	boolean turbo

Ontology	
Car	String colour
	int numberOfWheels
	int horsepower
	boolean turbo
	int doors

- Fields with different names but the same type and same class names will be loosely matched if they have not already been matched - Java *wheels* matched to Ontology *numberOfWheels*, *horsepower*, *doors*.
- Java fields with non-primitive types can have the their fields be inferred as fields of the class - fields of Engine inferred as fields of Car.

Illustrative Scenarios: Generic Matching

- If two classes match, any children of those classes are also likely to match.
- If two classes don't match but more than a number of **fields** match, then the classes are likely to match.
- Fields of a superclass can be inferred as fields of its children, and this combined list will be matched with.

Content

Performance

Times for i5 (quad core) 2.5 GHz processor

- Java Parser: Average ∼10,000 lines/sec
  - 19 classes (3,643 lines) in ∼0.001 secs
  - 62 classes (20,007 lines) in 2 secs
  - 707 classes (216,744) lines in 20 secs
- Ontology Parser: Average ∼800,000 concepts/sec
  - e.g. 2,358 concepts in ∼0.001 secs
- Matching: Average ∼20,000 combinations/sec
  - 4 classes (41 lines) matched to 6 concepts in ∼0.001 secs
  - 707 classes (216,744 lines) matched to 743 concepts in 26 secs

User Studies

- Subjects: Computing Science final year students
- Work in progress, nothing inferred yet as limited meaningful data produced

Studies:

- Study 1 is designed to test whether Facilitator forms the same matches as users over a given dataset.
- Study 2 is designed to test whether Facilitator aids in task completion by having users perform the same task with and without support from Facilitator.

Content

Contributions

- First time that code has been combined with ontologies
- Facilitator provides a range of integrated functionalities
- Finds and displays matches between code and ontologies
- Uses reasoning to form more complex matches
- Can create code from ontologies
- Can create ontologies from code

Future Work

Two important functionalities planned for Facilitator:

- Harmonisation: Carry out changes/corrections to *existing* code based on an ontology, and vice versa.
- Ontology/Matching Visualisation: Show ontology in a graphical format overlaid with matching information.

Thanks & Questions

Thanks for your attention!
Questions?