RODI: Technical User Manual

Christoph Pinkel[1] and Ernesto Jiménez-Ruiz[2]

[1] fluid Operations AG, Walldorf, Germany
[2] University of Oxford, UK

Abstract. This manual is addressed to users of the *RODI* benchmark suite [**?**], who intend to run the benchmark for evaluating some mapping generation system. *RODI* is designed to test the quality of (semi-) automatically generated mappings between relational schemata and ontologies. It includes a benchmark framework, test scenarios, a scoring function and reporting, as well as an extension mechanism to add new scenarios.
In this manual, we address preliminaries, framework setup and usage for standard evaluation, as well as a brief discussion of default scenario contents and on usage of the extension mechanism. For details on the purpose and design of this benchmark, please refer to our original paper [**?**]. Also, when referring to this work, please cite as [**?**].

1 Setup and Usage

1.1 Overview

In the following, we describe how to setup and use the *RODI* benchmark suite. After required libraries have been added, some configuration parameters need adjustment. Also, the environment needs to be setup accordingly (e.g., database server). Finally, the benchmark suite can be used to interact with and to test mapping generation tools.

1.2 General Environment Setup

Java, Libraries, Framework The benchmark suite is written in Java and and requires a Java runtime. In addition, a number of common, open source libraries (such as Apache Commons, the PostgreSQL JDBC connector and Sesame) are required. Table **??** shows a list of libraries that are being used by the benchmark. Please make sure that all required libraries are placed into `./lib` folder.

The *RODI* distributions is composed by the following folders and files:

```
data/
        * Database, queries, ontology and mappings for each scenario.
lib/
        * Libraries required by RODI
ontology/
        * Default folder for input ontologies generated by a system
```

Table 1: Required libraries – version numbers indicate reference versions that we used for testing and for initial experiments.

Library	Version	Web Site
Commons IO	2.1	http://commons.apache.org/proper/commons-io
Commons Logging	1.1	http://commons.apache.org/proper/commons-logging/
Sesame "OneJar"	2.7.13	http://sourceforge.net/projects/sesame
DB2Triples	1.0.2	https://github.com/antidot/db2triples/
Slf4J API	1.7.4	http://www.slf4j.org/download.html
PostgreSQL JDBC	9.3-1100	https://jdbc.postgresql.org/download.html
OWLAPI	3.4.8	http://sourceforge.net/projects/owlapi
OWLlink	1.2.2	http://owllink-owlapi.sourceforge.net/download.html
HermiT	1.3.8	http://www.hermit-reasoner.com/
LogMap	2.3	http://www.cs.ox.ac.uk/isg/tools/LogMap/

```
r2rml/
        * Default folder for input mappings generated by a system
reports/
        * Stores output results produced by RODI
sesame_db/
        * Required for runtime
src/
        * RODI source codes
tools/
        * Mappings and ontologies for BootOX, IncMap and D2RQ
config.prop
        * RODI configuration file
CHANGELOG.txt
        * Record of changes implemented in the benchmark
bootox_example.sh
        * Unix script to run RODI for BootOX
bootox_example.bat
        * Windows script to run RODI for BootOX
incmap_example.sh
        * Unix script to run RODI for IncMap
incmap_example.bat
        * Windows script to run RODI for IncMap
d2rq_example.sh
        * Unix script to run RODI for D2RQ
d2rq_example.bat
        * Windows script to run RODI for D2RQ
LICENSE
        * RODI license file
manual.pdf
        * This user's manual
```

```
pom.xml
        * Maven configuration details (e.g. library dependencies)
RODI_benchmark.jar
        * Application file. Use --help for command line options
setup_databases.sh
        * Sets up the databases required for the example scripts
```

The framework JAR (i.e., `./RODI_benchmark.jar` in the distribution) is runnable. Provided that all necessarry libraries are available in the expected location, running `java -jar RODI_benchmark.jar` should trigger the benchmark workflow (see below). Use `--help` for command line options. You may also use the `.sh` and `.bat` scripts as example. For use by non-developers, there is a complete pre-packaged distribution[3] of the latest RODI release available.

Development requires a clone of RODI's git repository.[4] You may either use the pre-configured Eclipse project, in which case several external libraries need to be added manually. As an alternative, you may use the included Maven to automatically download dependencies. To generate a RODI JAR file from the command line using Maven, run: `mvn package`.

Configuration A configuration file, `config.prop`, will be read from the current working directory while the benchmark is running. It needs to be setup for the particular environment (e.g., database connectivity) and can be used to alter the framework's behavior in several ways. A copy of `config.prop` with a standard configuration and explanatory comments is included in the distribution. In particular, you will have to specify connection details for the PostgreSQL database. Other parameters may be optional.

PostgreSQL Relational Database The benchmark requires a PostgreSQL database to be accessible by both the framework and tested tools. Required connection details include the host, port, user name, password and path to the PostgreSQL command line utility psql could be set in `config.prop`.

In default configuration, the framework attempts to connect to localhost on port 5432 with user and password postgres/postgres, and assumes the psql utilities is available in `/usr/bin/psql`. Note that Windows users must change this default configuration with the correspondent location of the psql utility (e.g. ``C:/Program Files/PostgreSQL/9.5/bin/psql'').

The user must have sufficient permissions to create and drop schemata in the database, as different scenarios are reflected by different schemata in the benchmark.

Sesame RDF Database Optionally, you may provide an RDF database management system to be jointly accessible by the benchmark framework and tested systems.

[3] `http://www.cs.ox.ac.uk/isg/tools/RODI/`
[4] `https://github.com/chrpin/rodi`

Setting up a database is not required, though. The benchmark can run its own lightweight version of Sesame internally. However, when using this internal database, transferring data between the framework and tested tools may become slightly more difficult, depending on your environment: if you access the framework programmatically from Java (see below) you may reuse this internal database for the tested tools. In all other control flows, however, you may need to import/export data between the tools and the framework. Still, using the internal database is usually the recommended method and is configured by default.

If you decide to instead use an external database instance, this database must be accessible for writing to the benchmark as a Sesame repository. Make sure to configure it accordingly in `config.prop`.

Ontology Alignment Ontology alignment is required when the evaluated systems do not produce R2RML mappings targetting the vocabulary of the ontology provided in the scenario. Instead, these systems create an intermediary ontology called putative or bootstrapped ontology. Thus, since the benchmark queries are based on the vocabulary of the ontologies provided in each scenario, the vocabulary of the putative or bootstrapped ontology should be aligned with the vocabulary of the scenario ontology. RODI currently relies on the ontology alignment systems LogMap.[5]

Reasoning RODI allows different reasoning modes during evaluation. Could be one of "none", "simplified" (simplified rule based reasoning), "structural" (structural reasoner that covers most of the needs of the default scenarios), "owllink" (full reasoning using an OWLlink reasoning server; tested with Konclude, HermiT, FaCT++). When ontology alignment is required, reasoning should be set to "structural" or "owllink". Make sure to configure it accordingly in `config.prop`. Full reasoners using OWLlink must be made available as an OWLlink Server on localhost:8181.[6]

1.3 Control Flow Options

Each benchmark run consists of three steps per benchmarked scenario. First, the scenario needs to be set up, i.e., prepare the data and environment. Second, the benchmarked system can run on it and produce mappings. Third, the benchmark framework evaluates those mappings.

There are different options to trigger those steps and to pass on required information.

Manual Control Flow A straight-forward and non-intrusive way to use the benchmark is to manually trigger each individual step of the control flow. Optionally, calls can be scripted. A benchmark run then consists of at leat two benchmark

[5] `http://www.cs.ox.ac.uk/isg/tools/LogMap/`
[6] See `http://owllink-owlapi.sourceforge.net/` for details

invocations, `--setup` and `--eval`. Between those invokations, the tested tool can run its evaluations. In addition it may be necessary to import/export data between the benchmark and tested tools (that is, unless you share data between the benchmark and tested tools through an externally setup Sesame database).

Use the following parameters when invoking the benchmark for manual control:

- † `--setup` prepares the configured scenario, cleaning repositories and loading data.
  - ‡ `--setup-ontology` uses a given ontology (e.g. placed in './ontology' folder) as scenario ontology. The given ontology may be the result of a prior ontology alignment step. Prepares the configured scenario, cleaning repositories and loading data, as the regular 'setup'.
- † `--eval` runs provided mappings, applies reasoning to derive additional facts, produces benchmark results. *Alternatively*, you may trigger the sub steps of evaluation manually. This gives you the opportunity to export/import the data between those steps and perform additional operations on them outside of the framework:
  - ‡ `--eval-r2rml` runs the provided R2RML mappings on the database and imports triples as generated by the mappings.
  - ‡ `--eval-reasoning` triggers the reasoner on already imported data to derive additional facts and adds them to the imported data.
  - ‡ `--eval-queries` runs the query based evaluation of the data, as already imported.
- † `--debug-queries` outputs a detailed report of the results for each individual query (e.g. missed expected results and unexpected provided results). Thanks to this option one can point to individual issues and bugs in the evaluated systems. This option must be used in conjunction with `--eval` or `--eval-queries`.
- † `--alignment` performs ontology alignment between the scenario ontology and the bootstrapped/putative ontology provided by a system. It requries the scenario and bootstrapper IDs as additional input (e.g. `--scenario=SCENARIO_ID --bootstrapper=BOOTSTRAPPER_ID`)."
- † `--import-base` imports data from a file import.rdf, which must be present in the current working directory in RDF/XML format. Replaces all existing data with the newly imported data. Overwrites both the base data (which remains unchanged during evaluation) and temporary copy. This is necessary in manual control, if a tool executes its mappings autonomously but does not share its repository with the benchmark.
- † `--export-base` exports base data to a file export.rdf and places it in the current working directory as RDF/XML. This is an alternative to sharing a ready and setup scenario with a tested tool without directly sharing the Sesame repository. Note, that base data represents the exact, unchange copy of most recently imported data (i.e., either manually imported or through setup). To retrieve changes that have been introduced during evaluation (e.g., mapping execution results, reasoning artifacts), use `--export-temp`.

`--import-temp` Exports base data to a file export.rdf and places it in the current working directory as RDF/XML. The base copy of data represents " the unchanged state of data, as previously imported, either manually or through –setup.

† `--export-temp` Imports data from a file import.rdf, which must be present in the current working directory in RDF/XML format into the temporary, changeable evaluation repository without tampering with the base data repository. Note, that temporary data will be changed or replaced with a base copy of the data at several stages of the process.

In addition, you may use `--scenario=SCENARIO_ID` to set the specified scenario, overriding general configuration, or `--title=SOME_TITLE` to set an explicit title to the current benchmark run, which would be reflected in generated reports. Titles may be used to annotate individual runs of the benchmark or iterations, such as "run1", "at5" or similar. For more information, try `--help`.

Programmatic Control Flow In more advanced settings, the framework's API can be directly accessed from Java, triggering each step from an controller application. For developers of Java based mapping generation systems this may be the most convenient option. Programmatic access offers more possibilities to fine-tune the control flow and also offers additional insights into evaluation results through the API. It is then also possible to run the framework and tested application within the same Java instance.

Programmatic access works by linking the provided benchmark JAR file, then using a small number of static key methods for triggering the first and last steps of the control flow:

- For initializing, `RdbmsUtil.loadPostgresDump(...)`, `SesameAdpter.reset()` and `SesameAdapter.getInstance(RepoType.MAIN).loadFromFile(...)`.
- For evaluation, `Evaluator.runR2Rml()`, `Evaluator.runReasoning()` and `Evaluator.evaluate(...)`.

Please refer to the source code or JavaDocs for details on any of those or any additional methods.

In between, the tested application should generate mappings. For iterative scenarios, this step can be taken multiple times, and reevaluation can be triggered after each iteration.

Automatic Control Flow In addition, a fully automatic mode is available. It will execute all three basic steps automatically and additionally supports to benchmark a sequence of different scenarios. However, to do so it puts strict constraints on when and how information exchange between the framework and tested tools should happen, partially depending on framework configuration. The sample `config.prop` in the benchmark distribution contains explanations of each available configuration parameter and its effects. Please refer there to find out whether the automatic mode can be tuned to be compatible with your setups.

1.4 Information Exchange

Information needs to be exchanged between the framework and tested tools after setup and mapping creation, while benchmark results need to be accessible for eventual external access.

Data All relational data will be shared through a dedicated PostgreSQL database (see above for configuration essentials). During setup, the framework will create one schema per scenario and fill it accordingly. The database should then be accessed by tested tools to try and create mappings. Eventually, the framework will connect back to the same database to evaluate created mappings against the same database.

RDF data may also need to be shared, though: during setup, the repository populates the RDF database with initial data. Usually, this would be only the target ontology, which could also be imported from its original RDF file provided with the scenario. However, in composite scenarios featuring data integration from different sources, additional (A-box) data may be provided as well. Also, while the framework does not need to receive any RDF data from tested tools if they submit materialized `R2RML` mapping files for evaluation, some tools may need to execute mappings themselves for various reasons. In those cases, mapping results need to be stored in the framework's RDF database for evaluation first. Two alternatives exist to achieve this sharing of RDF data: either, you install an external Sesame database server and configure it to work with both the benchmark framework and tested tools (also see above). Or, you use the `--import` and `--export` commands of the framework to import/export data between steps. In programmatic control, if the benchmark framework and tested tool run in the same Java instance, a third alternative is also viable: the framework's internal database repository is also available from the API and could thus be shared with the tested tool, if applicable.

Mappings After a tool has created mappings, it needs to provide them to the framework. There are two ways of doing this: either, the tool materializes the mappings as one or several R2RML mapping files and provides them in a preconfigured directory (by default, ./r2rml). Or, it passes no mappings at all, but executes the mappings itself and stores the results. The latter requires the tool to actively write materialized results in a shared RDF database repository, either by using the same database instance as the framework or by using the framework's `--import` command before evaluation.

Results The benchmark framework always provides evaluation results as *reports*. Reports are, by default, stored per scenario in directory `./reports` under the current working directory. For each query, a report notes the calculated precision, recall, and F1-measure.

2 Scenarios

RODI ships with scenarios that are readily including data. Those are in the domains of conferences, geographical data and oil and gas exploration.

2.1 Conferences Scenarios

As our primary domain for testing, we chose the conference domain: it is well understood, comprehensible even for non-domain experts but still complex enough for realistic testing and it has been successfully used as the domain of choice in other benchmarks before.

They are based on three different ontologies in the same domain. For each, there are scenario variants with different corresponding relational database variants, which focus on different mapping challenges. For instance, scenario *cmt_structured* is based on conference ontology *CMT* and uses the schema variant *structured* (see [**?**] for details). In addition, there are cross-matching scenarios, which map a database variant derived from one ontology to an entirely different ontology, e.g., *cmt2sigkdd*. These contain the most realistic (but also toughest) tests.

In the current version, the following scenarios from the conference domain are considered default scenarios and should be included in every evaluation while others are optional: cmt_renamed, conference_renamed, sigkdd_renamed, cmt_structured, conference_structured, sigkdd_structured, sigkdd_mixed, conference_nofks, cmt_mixed, cmt2conference, cmt2sigkdd, conference2cmt, conference2sigkdd, sigkdd2cmt, sigkdd2conference.

2.2 Geographical Scenarios

As a second application domain, *RODI* ships scenarios in the domain of geographical data.

The Mondial database is a manually curated geographical database. Based on Mondial, we have developed a number of benchmark scenarios.

Among those, we defined one default scenario, *mondial_rel*.

2.3 Oil & Gas Scenarios

Finally, we include an example of an actual real-world database and ontology, in the oil and gas domain: The Norwegian Petroleum Directorate FactPages.

We have constructed two scenarios that feature a different series of tests on the data: first, there are queries that are built from information needs collected from real users of the FactPages and cover large parts of the dataset. Those queries are highly complex and require a significant number of schema elements to be correctly mapped at the same time to bear any results. In additon, we have generated a large number of small, atomic query tests for baseline testing. These are similar to the ones used with the conference domain, i.e., they test for

individual classes or properties to be correctly mapped. A total of 439 such queries have been compiled in scenario *npd_atomic_tests* to cover all of the none-empty fields in our sample database.

A specific feature resulting from the structure of the FactPages database and ontology are a high number of 1:n matches, i.e., concepts or properties in the ontology that require a UNION over several relations to return complete results. 1:n matches as a structural feature can therefore best be tested in the *npd_atomic_tests* scenario.

Both *npd_atomic_tests* and *npd_user_tests* are default scenarios.

3 Adding Custom Scenarios

Extending *RODI* by adding scenarios is easy. Basically, you only need to add an ontology, corresponding database and query tests. To add a scenario, follow these steps:

1. Create a scenario folder under `./data` – the name of this folder must correspond to the name/identifier of your scenario.
2. Add a PostgreSQL database dump (format: plain) of your source database as `dump.sql`. The database must be stored in a schema with the name of your scenario (i.e., the same name as the scenario folder). Make sure that the dump is configured to first drop the database schema (if exists) and sets the search path to the schema of your scenario (e.g., "SET search_path = your_scenario, pg_catalog").
3. Add the target ontology (T-Box, only) to this new folder as `ontology.ttl|rdf|n3` (i.e., in Turtle, OWL/RDF or N3 format)
4. Add a sub folder `queries` to your scenario folder.
5. Adding other files (e.g., a `README.txt`) is optional. You can already now run your scenario to test whether it loads and executes without errors.
6. Add any number of query tests to the `queries` sub folder. Query tests have the file suffix `.qpair` (query pair). See below for details of their contents.

You can run custom scenarios just as any built-in scenarios: either by configuring them in `config.prop` for the automatic workflow, or by invoking them explicitly using the `scenario=...` parameter.

Writing Query Tests (.qpair) Query tests (.qpair) are formatted like Java property files, i.e., they contain named parameters in the form `param=value`.

There are three mandatory parameters for each query test: *name* (a string, used for identifying the test in logging and evaluation), *sql* (a SQL query) and *sparql* (the corresponding SPARQL query, meant to produce equivalent results as the SQL query when executed on the results of a correct/reference mapping). Queries should be tested manually to ensure that they produce the same results when used on the source database and target ongology after being filled with A-Box facts by a reference mapping, respectively.

Additional recommended parameters include *orderNum* (a weight factor to guarantee execution of queries in a particular order or semi-order) and *categories* (a comma-separated list of category tags; we calculate aggregate scores for each test category in evaluation).

Please have a look at the existing scenarios for examples.

References

1. Pinkel, C., Binnig, C., Jimenez-Ruiz, E., May, W., Ritze, D., Skjæveland, M.G., Solimando, A., Kharlamov, E.: RODI: A Benchmark for Automatic Mapping Generation in Relational-to-Ontology Data Integration. In: ESWC (2015)