Database Data Loading for Orkit

Extend Orekit with database support to reap the benifits of modern database technology.

Bob Reynders

1 Introduction

In this section I hope to prove that I have properly looked at the proposal[1], the corresponding documents/code and that I have formed an intermediate understanding of the application.

Summary Orekit requires external data to function properly. This data is supplied by third parties such as the IERS (EOP, leap seconds history) and the JPL (planetary ephemerides). These datasets can grow to a decent size, the EOP included in the default orekit-data set[2] for example goes up to 2.8MB while the entire (unzipped) file reaches 17MB.

The datasets are provided as-is and are distributed in text format (e.g. EOP) or binary (e.g. ephemerides). When needed, Orekit loads and parses these files at runtime.

Current State The current state of data-loading is shown in the class diagram in Figure 1.

When certain data is required (e.g. EOP History for the `FramesFactory`) the appropriate `DataLoader` implementation is used. For the `FramesFactory` component the default loaders are `BulletinBFilesLoader`, `EOP08C04FilesLoader`, `RapidDataAndPrediction-ColumnsLoader` and `RapidDataAndPredictionXMLLoader`. These loaders are responsible for parsing their respective file formats and will all be used to retrieve a `Set<EOPData>`.

The `DataLoader`s are unaware of the data source themselves, they are fed by a `DataProvider` using the *Visitor* pattern. `DataProvider`s are aware of the data source (e.g. a directory-oriented provider `DirectoryCrawler`), these providers can feed (be visited by) `DataLoader`s. Loaders don't ask providers to be fed directly, they contact the `DataProvidersManager` singleton, which contains user configured (or default) `DataProvider`s.

An added constraint with the current design is that the factory classes (e.g. `FramesFactory`) cache their loaded data without providing an API to reload or replace this data.

[1] https://www.orekit.org/forge/projects/orekit/wiki/SOCIS
[2] https://www.orekit.org/static/downloads.html

Problem The problem with the current data-loading mechanism is that it is hard to scale (text files) and that a lot of pre-processing is duplicated (parsed information is not stored). A solution would be to stored parsed datasets in a (scalable) database, to do this the existing design should be modified to support database loaders.

Additionally, removing the constraint on data loading for the factory classes consists of exposing an proper API with enough power to (1) remove current data, (2) add new data or (3) reload data.

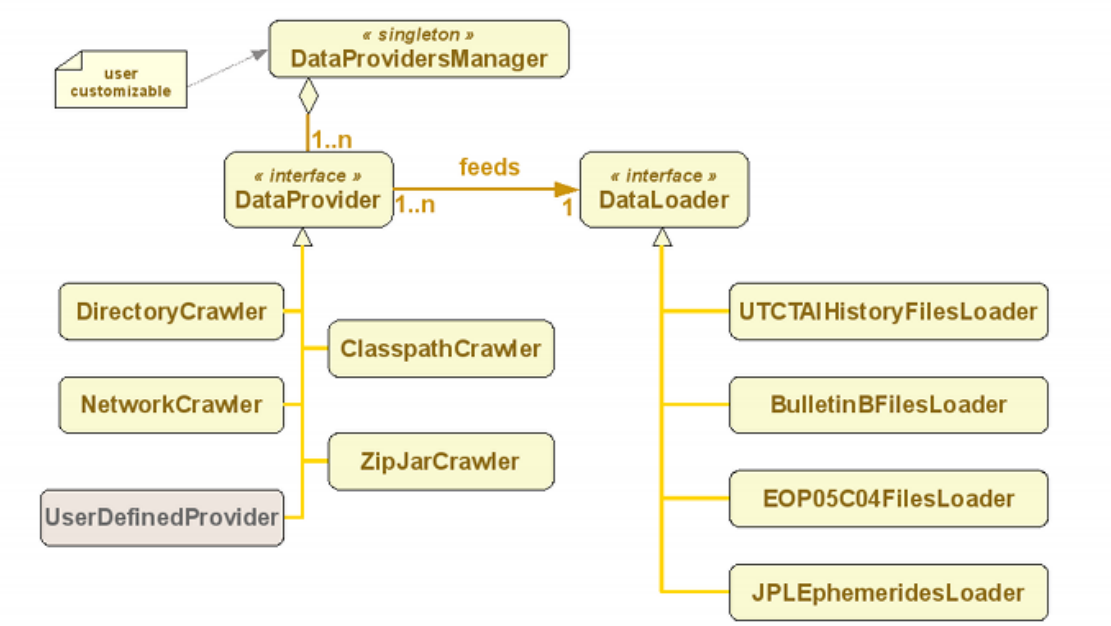

Figure 1: Data handling (https://www.orekit.org/forge/projects/orekit/wiki/Configuration)

2 Project Goals

My project goals for this SOCIS task would be:

- Implement an application that uses the existing Orekit features to parse raw datasets and store the results in a database.
- Add support to the existing Orekit feature set to load data from the aforementioned database.
- Extend the current API to support the reloading of data in factories.

3 Implementation

In Figure 2 I aim to show my current thought process/design towards database storage in the standalone application. For now I have focused primarily on the loading of `EOPEntry`s so these diagrams are not generalized.

The idea is to use JPA[3] to make the existing structure persistable. This includes the `EOPEntry` and `AbsoluteData` classes. I propose a parent entity for the existing `EOPEntry`s that contains the source for each stored entry. The source would be a hash of the `InputStream` that was used in the `EOPHistoryLoader`, this could be useful to pick up changes in the standalone application, if this proves not to be useful or worth it, the design will be simplified.

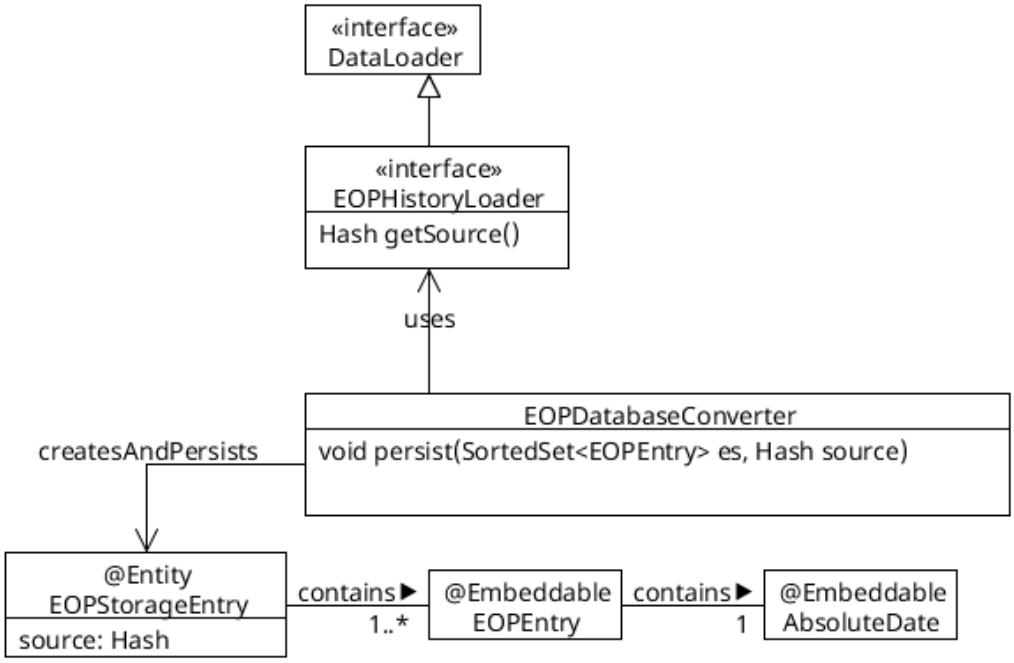

Figure 2: Database Converters

In Figure 3 I show my current design towards database loaders. I think it is best to create a database loader as just another subclass of `DataLoader`. Except this loader is not fed by the `DataProvidersManager` but by an `EntityManager`[4]. Instead of adding just the default data loaders in the required classes (e.g. `FramesFactory`) a check will be performed for an active database connection. If there is a database connection available, the database loader is added as a default loader as well.

[3] EclipseLink and OpenJPA implementations are Apache 2 compatible if their sources are not modified.
[4] The persistence manager.

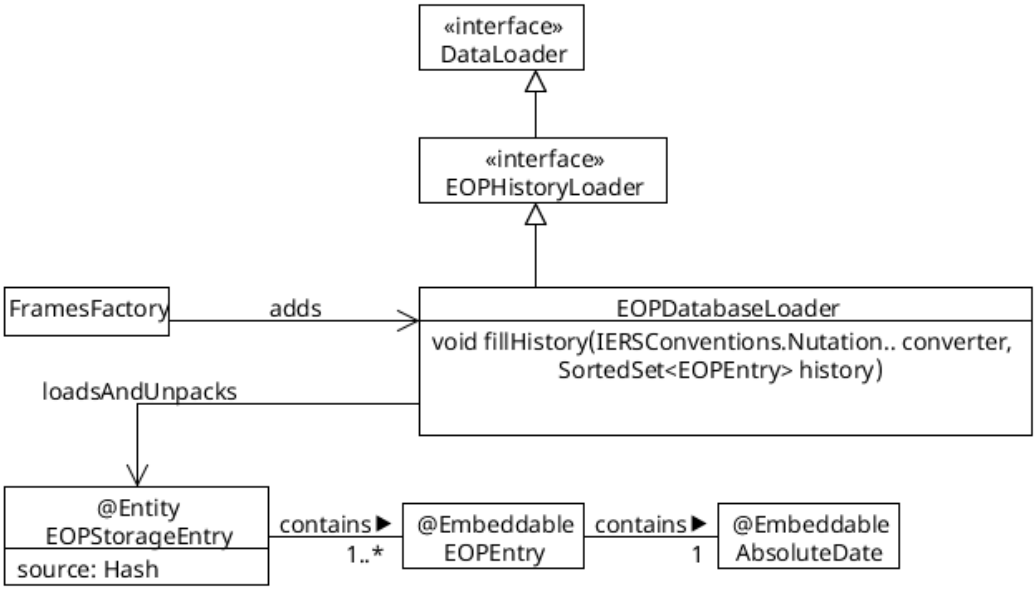

Figure 3: Database Loaders

In the example of EOP data, the database loader will fetch all `EOPStorageEntity`s and unpack the `EOPEntry`s.

Alternatives I propose JPA, an object-relational-mapper to ease development and lower database dependency while it should also be the most familiar tool for future Java developers. If there are strong objections against an ORM I can create a simple SQL scheme that does the same. I would however encourage you not to stick to SQL databases (JDBC), if you want to maximize the benefits (scalability, performance) you should consider moving to a NoSQL[5] database (JPA supports this). The reasoning behind this is that the kind of datasets in Orekit have a little amount of relationships which is the only vantage point of SQL vs NoSQL.

4 Timeline

My master thesis deadline is on June 6th so I will be preoccupied until then, my exam period starts on the 7th of June and ends on the 27th. During this period I will have limited time, it should however be possible to do preliminary work/reading!

In June I will work at limited hours a week (around 5) but as soon as I can (June 28th) I will treat this task as the job that pays my bills. I will be working full-time, 40 hour weeks, more if the project deadlines seem to be in danger!

[5]Read more about NoSQL on http://www.mongodb.com/nosql-explained.

I am confident that I can finish these goals in time. Based on the work packages in the proposal:

June	Learn more about Orekit and the purpose of all the datasets I will be working with while creating alternative designs for database support.
July 1 - July 21	Deliver a fully working prototype for EOP data using a local database (H2/JavaDB), this includes an EOP compatible standalone files → database converter.
July 22 - July 27	Extend support to other DataLoaders
July 28 - August 1	I *might* be allowed to give a talk at Scala2014[6], if I do I will also attend ECOOP which lasts until August 1st.
August 2 - August 14	Add support for a proper remote database (PostgreSQL).
August 15 - End	Cleanup, buffer period, implement data-reloading mechanism in factories and adjust existing tests to make use of it.

5 About Me

Most of the relevant information should be on my CV. To summarize, I'm a programming enthusiast who has been using Java extensively for the past 5 years both in hobby projects, at school and during an internship. I have worked on small scale (< 500 lines) to large scale (> 20000 lines) Java programs. Recently I have taken up more functional programming languages such as Scala (my current thesis).

I enjoy all things open source and am a passionate developer who likes to write code and I would be delighted to work with all of you!

[6] http://lampwww.epfl.ch/ hmiller/scala2014/