SqueakSave

Thomas Kowark Robert Hirschfeld Michael Haupt

July 7, 2009

Abstract

SqueakSave is an object-relational (O/R) mapping framework that provides the benefits of relational database storage without the need for extensive user defined mappings between object structures and relational schemas. The API is kept as simple as possible and blends in seamlessly with Smalltalk programming paradigms. Only minimal configuration is required in order to add relational persistence to existing or newly created applications

1 Project Description

Keywords: Persistence, Object-Relational Mapping, Metaprogramming and Reflection

1.1 Programming with SqueakSave

Using the SqueakSave framework in order to add persistence to an application is intended to be as simple as possible. The only thing required to use the framework is the specification of valid connection data for accessing the underlying relational database. Once this has been done, all application objects can be saved with a simple call of the save method. But not only storing objects is meant to feel as natural as possible within a Smalltalk environment - and what feels more natural than telling an object to store itself? Retrieving objects from the database is also seamlessly integrated into the language because the query API closely emulates the Smalltalk collection protocol.

1.2 Extending SqueakSave

SqueakSave is, however, not only built with the goal of simple usage, but an emphasis has also been put on the extensibility of the framework. Therefore, abstract base classes exist that provide simple guidelines for the development of new ways to describe the mapping between objects and relational database schemas (e.g. Pragmas, XML documents, etc.) or implement adapters for different relational database management systems.

1.3 Outlook

The first iteration of the framework is currently under heavy usage by the students of a university lecture held at the Hasso Plattner Institute[1] in Potsdam, Germany. All feedback that is gathered during those projects is recorded and will be incorporated into future versions of the framework. Additionally, we are also working on the extension of the framework with new features. This includes the ability to read or write O/R mapping descriptions of other O/R mapping frameworks like GLORP[2], but also some improvements regarding caching mechanisms, eager loading, or accumulation of storage operations.

[1] `http://www.hpi-web.de`
[2] `http://www.glorp.org`

1.4 Developers and Acknowledgements

The framework has been developed by Thomas Kowark during the course of his master's thesis at the Hasso Plattner Institute (HPI) in Potsdam, Germany. Helpful advise has been contributed by the members of HPI's Software Architecture Group[3]. While the predominant part of the framework has been developed from scratch, the query mechanism is based on the work carried out by William Harford and Eric Hochmeister for the ReServe project[4]

1.5 System Requirements

As the name implies, SqueakSave has been developed with and for the Squeak[5] Smalltalk dialect. While porting the framework to other dialects and VMs should be possible without any major effort, we are currently not planning to do so because of the increased maintenance overhead the emerges with offering the framework and especially fixes for all available platforms.

With regards to Squeak, supported versions include 3.9 and 3.10 images. A good starting point for (web) development with Squeak are the development images built by Damien Cassou[6].

2 Download and Installation

Currently, the framework is only available from the Squeak Source Monticello Repository of the SWA group:

- `http://www.hpi.uni-potsdam.de/swa/squeaksource/SqueakSave`

The repository is set-up to be readable without any special permission, thus, username and password can be left blank. We are currently also working on combining all required parts into a package that can be conveniently downloaded with the SqueakMap Package Loader or through the Universe Browser. But as of now, the following packages have to be loaded in the following order:

- SqSave
- The RDBMS client of your choice. For example, the `PostgreSQL Client` from Universe or SqueakMap or the `MySQL Client` from the SqueakSave repository.
- The driver package corresponding to the chosen client. This means, if you chose the PostgreSQL Client, you have to download the `SqSavePostgresDriver`. For MySQL you need the `SqSaveMySqlDriver` package, respectively.
- OPTIONAL: If you want to use SqueakSave's dynamic finder methods for searching, you also have to download the `VB-Regex` package. It is also available in Universe or via the SqueakMap Package Loader.

3 First Steps: Adding Persistence to an Application

The following introduction will provide only a short description of the required steps to get your application up and running with SqueakSave. For a more detailed description please refer to the paper or the master's thesis about the framework. They contain much more detailed information about the usage and especially the inner-workings of the framework. A usage introduction is also available in the wiki of the SqueakSource homepage[7] of the project

In the following we will use a simple weblog application for our example. The application's object model is fairly simple:

[3] `http://www.hpi.uni-potsdam.de/swa/`
[4] `http://www.squeaksource.com/REServe.html`
[5] `http://www.squeak.org`
[6] `http://damiencassou.seasidehosting.st/Smalltalk/squeak-dev`
[7] `http://www.hpi.uni-potsdam.de/swa/squeaksource/SqueakSave.html`

```
MyAppSqsConfig class >> connectionSpecification
   ↑ SqsPGConnectionSpecification
      user: 'test'
      password: 'test'
      database: 'my_blog'
```

Listing 1: Specifiying the Configuration

- *User*s are separated by inheritance into *Admin*s and *Author*s.
- Each user has a username, password and email adress.
- Authors have a *Blog* with 0 or more *BlogEntries*.
- Each BlogEntry has 0 or more *Comment*s attached to it.
- Blogs, BlogEntries, and Comments have a title. BlogEntries and Comments additionally a body.
- An Admin is responsible for a number of Blogs - administratedBlogs.

3.1 Creating a configuration

SqueakSaves uses a naming convention based approach to determine the configuration for your application. This configuration basically tells the framework, where to store your objects. In order to create a configuration, all that has to be done is to create a subclass of *SqsConfig* that is named after the category of your application. If all you application's model classes for example reside in a category named MyBlog-Models then you can either name the configuration class *MyBlogSqsConfig* or *MyBlogModelsSqsConfig*. The latter, however, will only be valid for all model classes, so if you try to persist instances of classes in let's say MyBlog-ExtendedModels, the framework will trigger an error. No all that's left to do is creating a method named `connectionSpecification` on the class side of this configuration class. Depending on the chosen RDBMS driver, this method has to return an instance of the respective connection specification class. For PostgreSQL this method would look like shown in Listing 1.

3.2 Basic Persistence Operations

Now that the framework knows where to store your data, you can start to store created objects in the database and perform queries on the persistent space. Storing is as simple as it gets, since all you have to do is call `save` on any object of your application. All required table structures will be automatically generated, or, if already present, altered to reflect the current structure and data types of the saved object. So if you add an instance variable to one of your classes, the framework accordingly alters the table structure. So no need for you to write any O/R mapping descriptions by yourself - the framework takes care of this and you can alter them if you have to. Listing 2 shows you in a nutshell what you can do with SqueakSave and your objects.

3.3 Searching

While simple storing is of course an important part of persistence, retrieving objects from persistent space is just as viable for any application. SqueakSave closely emulates the Smalltalk collection protocol, so you can almost write your queries as if you were operating on the collection of all instances of a class within your image. The examples in Listing 3 show some of the possibilities of the SqueakSave query mechanism.

As you can see, simple comparisons of direct attributes, such as username, with a given value are possible. Additionally, you can also follow multiple references from an instance variable and thus only select authors with a blog that has more than 10 entries or only blogs with at least one blog entry that has been commented at least once. The only limitations of this algorithm are:

```
author := Author new
    username: 'hemingway';
    password: 'secret';
    email: 'ernest@hemingway.net'.

author blog: (Blog new title: 'My Blog').
author save.

blogEntry := BlogEntry new
    title: 'My first Blog Entry';
    body: 'Just testing ... '.

blog blogEntries add: blogEntry.
blog save.

"now we want to delete the blog entry from our database"
blogEntry destroy.
```

Listing 2: SqueakSave Basic Operations

```
(SqsSearch for: Author) detect: [:anAuthor | anAuthor username = 'hemingway'].

(SqsSearch for: Author) select: [:anAuthor | anAuthor blog blogEntries size > 10].

(SqsSearch for: Blog) select: [:aBlog | aBlog blogEntries anySatisfy: [:aBlogEntry |
    aBlogEntry comments size > 1]].
```

Listing 3: SqueakSave Search Operations

- Method calls to the search block objects are limited to instance variable accessor methods
- operations on Collections, Integers, etc. have to be implemented within SqueakSave since they are mapped to SQL queries

3.4 Summary

The previous examples have shown that SqueakSave provides persistence for applications in a manner that can hardly be simplified anymore. One `save` call is enough to store an object and associated ones without caring about table structures, O/R mappings, and the like. Searching is also made as simple as possible and if it wasn't for the *`SqsSearch`* part, would be no different from searching for any object within the image itself. However, this functionality is only the tip of the iceberg, so please try the framework for your application and take a look at the detailed communication to explore the possibilities offered by SqueakSave.

4 License Information

The SqueakSave framework is licensed under the MIT License. The MySQL bindings that are available in the Monticello repository of SqueakSave are licensed under the GPL. However, they are not an integral part of the framework and distributed separately.