impact – A Modelica® Package Manager

Michael Tiller
Xogeny Inc., USA
michael.tiller@xogeny.com

Dietmar Winkler
Telemark University College, Norway
dietmar.winkler@hit.no

Abstract

To manage complexity, modern programming languages use organizational units to group code related by some common purpose. Depending on the programming language, these units might be called libraries, packages or modules. But they all attempt to encapsulate functionality to promote modular code and reusability. For the remainder of this paper, we will simply refer to these organizational units as *packages* (as they are called in Modelica).

Also common to many modern programming languages are tools to manage these packages. These tools are generally called *package managers* and they allow developers to quickly "fetch" any packages they may need for a given project. The main functions of package managers are to allow developers to search, install, update and uninstall packages with a simple command-line or graphical interface. In the Java world, the most common package manager is `maven`. For Python, tools like `easy_install`[1] and `pip`[2] are used for managing packages. For client-side web development, `bower` is used. For server-side JavaScript, the tool of choice is `npm`[3]. For compiled languages, these package managers often include some additional build functionality as well.

This paper introduces *`impact`*, a package manager for Modelica. Using `impact`, Modelica users and developers can quickly search for, install and update Modelica libraries. In this paper, we will discuss the functionality provided by `impact`. In addition, we will discuss how the functionality was implemented. As part of this we will discuss the importance of collaborative platforms, like GitHub[4] in our case, for providing a means for collecting, curating and distributing packages within a community of developers.

The `impact` package manager is provided to the Modelica community as a free, open-source tool. Furthermore, the protocols involved are all documented and we encourage tool vendors to integrate them into their own tools so they can provide the same searching, updating and installation capabilities that the command-line tool provides.

Keywords: *Modelica, package manager, GitHub, dependency management, Python*

1 Introduction

It is increasingly the case that the adoption of new technologies hinges on automating away the tedious tasks required to learn and adopt these new technologies. For programming languages or frameworks, this means streamlining the process by which libraries can be found and installed.

For nearly all modern languages, this issue of "package management" has reached the point where it is almost an element of language design. The Java world has the `maven` tool, Scala has `sbt`, Node.js has `npm` and the Go language includes built-in support for package management via its the command line `go` compiler.

In the Modelica world, this issue has been largely overlooked. Although there have been proposals for formats to list network accessible libraries, these efforts have remained mere proposals without any concrete functionality. The `impact` project was inspired by the Bower[5] project's approach. This lightweight, `git`-centric approach (discussed in Section 3) turned out to be relatively straightforward to implement and provides functionality otherwise unavailable in the Modelica world.

The goal of the `impact` project is to provide the same basic package management features found in most package managers. These features will be discussed in detail in subsequent sections of this paper (see Section 2). Our goal is to lower the barrier for users to find, install and update libraries. At the same time, we expect that the `impact` tool itself will be just as easy to install as the libraries it supports.

The contribution of the `impact` project is making installation of Modelica packages as easy as possible. There are actually three important aspects in our approach. The first aspect is the one most apparent to the user, a command-line interface that can be used to easily install not just a given Modelica library, but **also its dependencies**. However, such a command line tool must rely on the second aspect which is the avialability of a centrally served, up to date index of packages. The final aspect is making it easy for library developers **to publish** their libraries in such a way that they are available to other Modelica users through the `impact` package manager. Each of these aspects will be discussed as part of this paper.

It is worth noting that while `impact` can handle dependences, it does not solve some of the problems currently inherent in Modelica. At the moment, dependencies between packages are described by individual versions. The result is that these dependencies can create brittle chains which are not always possible to satisfy. The logic for unconvering dependencies in `impact` is very simple. It merely identifies any dependencies explicitly listed by the library and then attempts to find that version of those packages within the `impact` package index. Hopefully the Modelica annotations to express dependencies will be refined to support a richer set of relationships. If so, the logic used by `impact` to identify and install these dependencies can be extended to support this improved expressiveness.

2 Command Line Interface

2.1 Installation of `impact`

The `impact` tool is available from “PyPI”[6] and can be installed by running either

```
$ pip install impact
```

or

```
$ easy_install impact
```

As an alternative one can also download the sources from `https://github.com/xogeny/impact` or `https://pypi.python.org/pypi/impact`, unpack and run

```
$ python setup.py install
```

in order to install it.

2.2 Searching

Searching for librararies is done by executing:

```
$ impact search <search term>
```

This will print a list of all package whose names and/or description strings contain the “`<search term>`”. The returned list also contains the URL where the Modelica package is hosted.

The output can also be made more verbose with:

```
$ impact search -v <search term>
```

which, in addition, will return the description string and the available versions.

2.3 Installing packages

Once a package of interest is found using `search`, it can be installed by executing:

```
$ impact install <package name>
```

This will then fetch not only the package itself and extract it in a configurable target directory but **it will also fetch the dependencies of the packages** as long as those are available to `impact`.

If several versions of a package are available, `impact`will choose the latest one. If this is not desirable then one can also specify the version explicitly. For example, in order to install Modelica version 2.2.2 one would execute the command:

```
$ impact install Modelica#2.2.2
```

Just like for the `search` sub-command there is also a verbose switch "`-v`" available for `install` which will give information on what versions are available, which version is going to be installed, what the dependencies are and where the version will be downloaded from.

In addition there is also a "`[-dry-run|-d]`" option available which does not download or extract any files but will simply report what `impact` *would* do. Users will generally use the "dry-run" option in combination with the "verbose" option.

3 Candidate Packages

The question that might arise now is, how does `impact` know what packages and which versions of those packages are available.

3.1 Making packages visible

The Modelica Association (MA) has always maintained a list of available Modelica libraries on their website under `https://modelica.org/libraries`. Initially, the list was a static web page which listed the different packages and their latest version as submitted to the webmasters of the MA. Keeping the list of packages up-to-date was a manual job for both the website maintainers and package developers.

In spring 2013, all the free packages listed on `https://modelica.org/libraries` were moved to individual repositories on GitHub. Third-party packages can be found under `https://github.com/modelica-3rdparty` and packages by the MA under `https://github.com/modelica` This had the following benefits:

- Package developers can access their package repository directly without having to involve the webmasters of the MA thus submitting updates any time.
- All packages now have an individual issue tracker and version control service in place.
- The webmasters of the MA can now collect the latest information on all the packages automatically in order to generate an up-to-date listing on `https://modelica.org/libraries` (more on this later in Section 3.3).

3.2 Semantic versioning

One thing that is important when trying to build up a package manager that can also handle version dependencies is the need for a proper approach to version numbering.

We decided to base our package manager on a system called Semantic Versioning[7]. As a result, package developers are strongly encouraged to use semantic versioning so that they are compatible with `impact`. This has the additional benefit of being a well-documented and logical approach.

Semantic versioning has the simple rule of:

Given a version number MAJOR.MINOR.PATCH, increment the:

1. *MAJOR version when you make incompatible API changes,*
2. *MINOR version when you add functionality in a backwards-compatible manner, and*
3. *PATCH version when you make backwards-compatible bug fixes.*

In addition, pre-releases (*e.g.,* beta releases, release candidates) and build metadata (*e.g.,* version control hashes) are also taken care of in this system, details can be found in the manual page[7].

3.3 GitHub API

Having all packages available as GitHub repositories means that we can use the *GitHub API v3*[8] in order to collect data about those packages. All API access is over HTTPS, and accessed from the `api.github.com` domain. All data is sent and received as JSON[9].

For example if one visits: `https://api.github.com/users/modelica/repos` a verbose list containing a series of information of all repositories that exist under the user `modelica` is returned in JSON format. This includes also a new API-url for retrieving the tags of a specific repository. For example, by visiting `https://api.github.com/repos/modelica/Modelica/tags` we get a list of all tags for the Modelica Standard Library repository including download links for a zipped version of the tagged source code.

As mentioned before, all data is returned as JSON according to a proper schema. This makes it easy to pull out all the information we need. Initially, this information was used to generate an up-to-date listing of all MA and third-party packages to be displayed on `https://modelica.org/libraries` but we also noticed quite quickly the possibilities such an API opens up. It was offering the very information we needed in order to build a catalog of available packages including the different tagged versions for download.

3.4 GitHub only?

The mechanism described so far seems to depend a lot on GitHub's API. So one might wonder is there a danger of locking us to GitHub.

The answer is actually no. We chose GitHub just as **one** possible data source. It is possible to enhance `impact` with other "connectors" to other existing package hosting solutions (private or public). As long as the schema is known to `impact` it can then pull its data from basically all possible places. For example, it would also be possible to use the API of the GitLab project[10] to extract the same information.

4 Package Index

Package information is maintained in an index file. This index file is also generated by `impact` but the process of building an index is not normally used by the user or tool vendors so all discussion about the creation of index files is presented later in Section 5. The index file is stored in JSON format and has the following structure:

```
{
  "<LibraryName>": {
    "homepage": "<URL>",
    "description": "<description string>",
    "versions": [
      "<version number>": {
        "version": "<version number>",
        "major": <major version number>,
        "minor": <minor version number>,
        "patch": <patch version number>,
        "tarball_url": "<URL to tarball>",
        "zipball_url": "<URL to zipball>",
        "path": "<path to library>",
        "dependencies": [
          {"name": "<DepLibName1>",
           "version": "<version string>"},
          {"name": "<DepLibName2>",
           "version": "<version string>"},
          ...
        ]
      }
      ...
    ]
  }
}
```

All quantities listed within angle brackets, `<...>`, are library specific details. The `<LibraryName>` is the name of the `package` in Modelica. Generally speaking, all version numbers follow the semantic versioning approach. However, since not all Modelica libraries currently follow semantic version conventions, indices can include semantic duplicates (*e.g.,* `1.0` and `1.0.0`) which reference the same underlying version. Therefore, any non-semantic conforming versions (*e.g.,* `1.0`) will act as "redirects" to the semantic version (*e.g.,* `1.0.0`).

The `homepage` field is a URL to a web site that contains additional information about the library. The `zipball_url` and `tarball_url` fields point to archives that can be downloaded, in the `zip` and `tar` formats, respectively. The `dependencies` field lists all the library's dependencies. These are the libraries that will also be installed when installing the specified library version they are listed under. The `path` field specifies the name of the directory or file representing the Modelica library within the specified archive.

Note, we have not currently defined a schema for this format. To promote interoperability we recognize that a formal schema would be the next logical step. We have added the creation of a JSON schema for the index file format to our list of next steps to promote interoperability with other implementations. Our hope is that such a schema would further encourage tool vendors to support this format as a means of publishing information about available Modelica libraries.

The Modelica Association index of publicly available libraries can be found at `https://impact.modelica.org/impact_data.json`.

5 Private Packages

5.1 Using Private Indices

As mentioned in Section 4, there will be a package index hosted on `modelica.org` that lists any packages connected to special Modelica related GitHub accounts. This provides a means for library developers to quickly add their libraries to the set of libraries that are publicly indexed.

However, we recognize that many users will depend on libraries that cannot be hosted publicly. At the same time, we would like for those users to be able to benefit from the same kind of package management features for finding and installing their privately hosted libraries.

For this reason, users can create a special configuration file that lists the indices to be searched. By default, `impact` will use only the index hosted on `modelica.org`. But through custom configurations, users can specify any collection of indices (public or private) they wish to use.

To specify an alternative list of indices, a user would simply edit their user configuration file and add the following text:

```
[Impact]
indices=<url1>,<url2>
```

where the value of `indices` is a list of URLs pointing to index files. The default value for the `indices` variable is `https://impact.modelica.org/impact_data.json`. In cases where private index are files used, the URLs for private index files should be listed first and the URL to the index file hosted on modelica.org should be last.

Note, the location of the user's configuration file will depend on the platform their are using. Information about the location of the configuration file and current settings is generated by the following command:

```
$ impact info
```

5.2 Generating Private Indices

In order for users to include a private index file in the list of indices to be searched (as discussed 5.1), it is necessary to also have the capability to easily generate such private indices. This functionality is also available using the `impact` command line although we did not discuss it previously because it is not functionality that a typical user would require.

To generate an index file, the following `impact` command line syntax should be used:

```
$ impact refresh <source1> \
      <source2> ... <sourcen> \
      -o <output file>
```

where each source is a URL that encodes information about a potential source. For example, the default sources are `github://modelica-3rdparty/.*` and `github://modelica/.*` (in other words, all repositories belonging to the GitHub user `modelica-3rdparty` and all repositories belonging to the GitHub user `modelica`, respectively). Note that later sources have a higher priority than earlier sources. Also, at the moment the only types of repositories supported are GitHub repositories although by using a URL based approach it is easy to extend the possibilities to include Git, Subversion or other types of repositories as long as the information required for the index file is available.

The output file generated from this command should then be made accessible to users so they can incorporate it into the set of indices they search (see Section 5.1 for more details).

6 Source Code and Licensing

The `impact` project started off as a simple script, then a gist and eventually a complete repository. The repository for the source code is hosted on GitHub at `https://github.com/xogeny/impact`. Potential contributors are invited to fork the repository and add more functionality. Contributions to improve `impact` are very welcome.

The software is distributed under an MIT license. As such, there are no significant restrictions on using the code in open-source, closed-source or commercial projects. In fact, we welcome vendor support and adoption. In addition to making the complete source code for the package manager available and documenting the functionality in this (freely downloadable) paper, we are also making the index data freely available from the `modelica.org` domain. We hope that all these measures will lead to the highest possible chance of adoption.

7 Conclusion

Inspired by the approach taken in Bower, we've created `impact`, a package manager for the Modelica eco-system. Like Bower, this approach is relatively light and relies heavily on the GitHub API to aggregate and index publicly available libraries. Also like Bower, our approach relies on semantic versioning, a widely adopted approach for associating concrete meaning to the various elements of a version. We use these meanings to help validate and organize the tags associated with Modelica libraries.

The `impact` tool also provides one of the key elements of a package manager, the ability to automatically pull in dependencies during installation. With this feature, users can list the libraries they directly depend on and `impact` will automatically install any additional dependencies. The dependency information is constructed automatically from the `version` annotation already present in Modelica libraries.

The index of publicly available libraries is hosted on `modelica.org`. But `impact` can also be used to index and install private libraries as well. All `impact` functionality (installing, searching, *etc.*) is available for both public and private libraries.

References

[1] easy_install. *Easily download, build, install, upgrade, and uninstall Python packages*. 2014. URL: `https://pypi.python.org/pypi/setuptools`.

[2] pip. *A tool for installing and managing Python packages*. 2014. URL: `http://www.pip-installer.org`.

[3] npm. *Node Packaged Modules*. 2014. URL: `https://npmjs.org/`.

[4] GitHub. *Build software better, together.* 2014. URL: `https://github.com/`.

[5] Inc. Twitter. *Bower – A package manager for the web*. 2014. URL: `http://bower.io/`.

[6] PyPI. *The Python Package Index*. 2014. URL: `https://pypi.python.org/pypi`.

[7] Tom Preston-Werner. *Semantic Versioning 2.0.0*. 2014. URL: `http://semver.org/`.

[8] GitHub Developers. *GitHub API v3*. 2014. URL: `http://developer.github.com/v3/`.

[9] JSON. *JavaScript Object Notation*. 2014. URL: `http://www.json.org/`.

[10] GitLab developers. *GitLab API*. 2014. URL: `https://github.com/gitlabhq/gitlabhq/blob/master/doc/api/README.md`.