University of Southern Denmark
Department of Mathematics and Computer Science,
IMADA

Master Thesis

Building a Package Manager for Jolie

Author:
Dan Sebastian Thrane

Supervisor:
Fabrizio Montesi

April 17, 2017

Contents

Introduction

Introduction goes here.

Background

Background goes here.

Jolie Packages

Packages

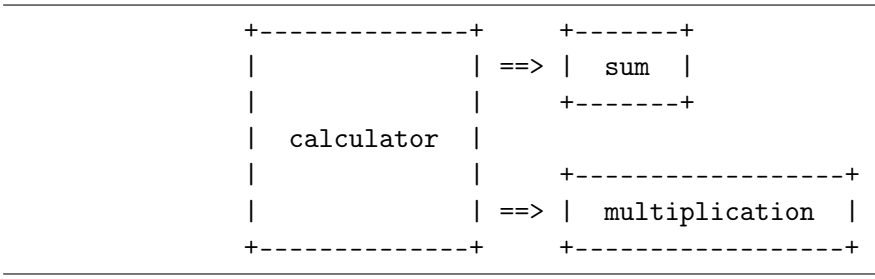
A Jolie Package is an extension of a Jolie Module. Recall that a Jolie Module was defined as a collection of resources, a name, and optionally an entry-point for the module. A package extends this concept by adding information required for package management.

A Jolie Package is described by a package manifest. The package manifest is a JSON file, which is always placed at the root of the package, and must be called `package.json`. The fixed location allows for the package manager to easily identify a package. The JSON format was chosen as it plain-text, and easy to both read and write for both humans and machines.

In listing 3 we show a simple package manifest. This manifest showcases the most important features of the manifest.A complete specification of the package manifest format can be seen in Appendix A. The service that this manifest describes is shown in figure 4.

```
{
    "name": "calculator",
    "main": "calc.ol",
    "description": "A simple calculator service",
    "authors": ["Dan Sebastian Thrane <dathr12@student.sdu.dk>"],
    "license" "MIT",
    "interfaceDependencies": [
        { "name": "math", "version": "1.0.0" }
    ],
    "dependencies": [
        { "name": "sum", "version": "1.2.X" },
        { "name": "multiplication", "version": "2.1.0" }
    ]
}
```

Listing 1: A Simple Package Manifest



Listing 2: A Calculator Service

Lines 2-3 take care of the module definition. The remaining attributes, however, are entirely unique to packages. Some attributes included in the manifest are there for indexing and discoverability purposes, examples of such attributes are shown in lines 4-6.

Package Manager

This is the chapter on package managers.

Packages

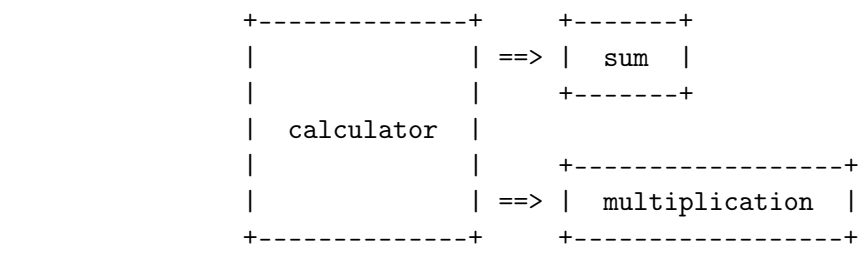
A Jolie Package is an extension of a Jolie Module. Recall that a Jolie Module was defined as a collection of resources, a name, and optionally an entry-point for the module. A package extends this concept by adding information required for package management.

A Jolie Package is described by a package manifest. The package manifest is a JSON file, which is always placed at the root of the package, and must be called `package.json`. The fixed location allows for the package manager to easily identify a package. The JSON format was chosen as it plain-text, and easy to both read and write for both humans and machines.

In listing 3 we show a simple package manifest. This manifest showcases the most important features of the manifest.A complete specification of the package manifest format can be seen in Appendix A. The service that this manifest describes is shown in figure 4.

```
{
    "name": "calculator",
    "main": "calc.ol",
    "description": "A simple calculator service",
    "authors": ["Dan Sebastian Thrane <dathr12@student.sdu.dk>"],
    "license" "MIT",
    "interfaceDependencies": [
        { "name": "math", "version": "1.0.0" }
    ],
    "dependencies": [
        { "name": "sum", "version": "1.2.X" },
        { "name": "multiplication", "version": "2.1.0" }
    ]
}
```

Listing 3: A Simple Package Manifest



Listing 4: A Calculator Service

Lines 2-3 take care of the module definition. The remaining attributes, however, are entirely unique to packages. Some attributes included in the manifest are there for indexing and discoverability purposes, examples of such attributes are shown in lines 4-6.

Architecture

The entirety of the ecosystem around JPM is written in Jolie, using the features that the Jolie Module System provides, along with the features that JPM itself provides.

At the ten-thousand foot view of the architecture, it consists of three core services, as shown in figure 5:

1. **Registry**: Responsible for serving packages known to the registry.

2. **JPM**: Provides the back-end of JPM. This includes communication with one, or more, registries, for example to download packages.
3. **CLI**: Provides the front-end of JPM. The front-end is responsible for displaying a user-facing interface, and will communicate with the back-end to perform the actual work.

```
+-----------+     +-------+     +------------+
|  jpm-cli  | ==> |  jpm  | ==> |  registry  |
+-----------+     +-------+     +------------+
```

Listing 5: Ten-thousand foot view of the JPM architecture

Integrity Checks

This is a section about integrity checks in JPM.

Appendix

Appendix A: JPM Manifest Specification

This document covers the specification of the file which defines a package. The format used for this document will be JSON, but the format and whether or not to allow for several documents is still up for discussion. For now we should avoid using any features which the generic Jolie value cannot support.

Purpose

The purpose of the package document is to define what a package is. Every Jolie package will contain such a document, and it describes several important properties about the package. These properties are described in the section "Format and Properties".

Table of Contents

Format and Properties

name

Name: `name`

Optional: false

Type: `string`

Description: The `name` property uniquely defines a package in a registry. Every registry must only contain a single package with a given name.

Rules:

- The name of a package is *not* case-sensitive
- The length of a name is less than 255 characters
- Names are US-ASCII
- Names may only contain unreserved URI characters (see section 2.3 of RFC 3986)

If any of these rules are broken the JPM tool should complain when *any* command is invoked. Similarly a registry should reject any such package.

version

Name: `version`

Optional: false

Type: `string`

Description: This property describes the current version of this package.

Rules:

- The version string must be a valid SemVer 2.0.0 string (see http://semver.org/spec/v2.0.0.html)

license

Name: `property_name`

Optional: false

Type: `string`

Description: Describes the license that this package is under.

Rules:

- Must be a valid identifier. See https://spdx.org/licenses/

authors

Name: `authors`

Optional: false

Type: `string|array<string>`

Description: Describes the authors of this package

Rules:

- The array must contain at least a single entry
- Each entry should follow this grammar:

```
name ["<" email ">"] ["(" homepage ")"]
```

private

Name: `private`

Optional: true

Type: `boolean`

Description: Describes if this package should be considered private. If a package is private it cannot be published to the "public" repository.

Rules:

- By default this property has the value of `true` to avoid accidential publishing of private packages.

main

Name: `main`

Optional: true

Type: `string`

Description: Describes the main file of a package.

Rules:

- The value is considered to be a relative file path from the package root.

dependencies

Name: `dependencies`

Optional: true

Type: `array<dependency>`

Description: Contains an array of dependencies. See the "dependency" sub-section for more details.

Rules:

- If the property is not listed, a default value of an empty array should be used

dependency

Type: `object`

Description: A dependency describes a single dependency of a package. This points to a package at a specific point on a specific registry.

name **Name:** `name`

Optional: false

Type: `string`

Description: Describes the name of the dependency. This refers to the package name, as defined earlier.

Rules: A dependency name follows the exact same rules as a package name.

version **Name:** `version`

Optional: false

Type: `string`

Description: Describes the version to use

Rules:

- Must be a valid SemVer 2.0.0 string
- (This property follows the same rules as the package version does)

registry **Name:** `registry`

Optional: true

Type: `string`

Description: This describes the exact registry to use. If no registry is listed the "public" registry will be used.

Rules:

- The value of this property must be a valid registry as listed in the `registries` property.

registries

Name: `registries`

Optional: true

Type: `array<registry>`

Description: Contains an array of known registries. See the registry sub-section for more details.

Rules:

- This property contains an implicit entry which points to the public registry. This registry is named "public".

registry

Type: `object`

Description: A registry describes a single JPM registry. A JPM registry is where the package manager can locate a package, and also request a specific version of a package.

name **Name:** `name`

Optional: false

Type: `string`

Description: This property uniquely identifies the registry.

Rules:

- A name cannot be longer than 1024 characters

- The name cannot be "public"
- No two registries may have the same name

TODO:

- Encoding of name
- Should the length limit be dropped? There is no technical reason for the limit

location **Name:** `location`

Optional: false

Type: `string`

Description: Describes the location of the registry.

Rules:

- Must be a valid Jolie location string (e.g. "socket://localhost:8080")