[Guix] Clojars importer for Guix

Leandro Doctors
<ldoctors@gmail.com>

March 31, 2020

1 Summary

1.1 Objective

Add an importer for Clojars, the *de facto* repository for Clojure packages.

1.2 Benefits

- In the short term, the Clojars importer will add to Guix a potential universe of at least 500K new potential users.

  This is because Clojure (a modern LISP dialect) is the third most popular language of the JVM platform, which is used by millions of people around the world.

- In the long term, supporting Clojure would have the additional benefit of bringing potential Guix contributors.

  This is because Guix is also written in a LISP dialect (Scheme), lowering the skill barrier for making the transition from one language to the other.

1.3 Deliverables

- An interface/wrapper for a Clojure API for transitive dependency graph expansion (most likely, `clojure.tools.deps.alpha`; alternatives being `boot` and `leiningen`).
- A full, testable Clojars importer, integrated with GitHub for source retrieval (and, if time permits, with other forges).

2 Motivation

Guix supports importing packages from multiple, diverse sources. They range from metadata from the GNU System, repositories such as the Python Package Index and the Comprehensive R Archive Network, down to plain JSON metadata files. Supporting a broad offering of package sources does not only provide Guix with a broad catalog of packages, it also enriches it with a plethora of languages to choose from. All this offering of software platforms enables Guix as a competitive option to a wide variety of users.

Whereas the array of supported Guix sources is indeed vast and diverse, there is a major source yet to be supported: JVM packages[1]. The Java Virtual Machine is one of the most popular platforms for software development: as of 2013, Java alone (by far, the most popular language of this platform) comprises a reported 9 million of developers worldwide[2]. Whereas initially intended for a single language (Java), the JVM has been used as a platform to develop new languages. The only requirement that any language hosted on the JVM platform has to comply with is to generating bytecode artifacts compliant with the JVM specifications. In this way, as of March 2020, Wikipedia reports more than 60 languages that run on the JVM[3].

All JVM packages are distributed via `.jar` files -compressed files containing bytecode files. Across packages, there is dependency graph. Whereas some niches and / or languages use their own package repository, the main *de facto* repository for JVM packages is “The Central Repository” (formerly known as “Maven Central”)[4]. All these repositories host thousands of JVM packages. They also maintain the dependency graph that relates them. As with any client-server architecture, any tool that wants to fetch packages from any JVM package repository has to take care of implementing their own traversal heuristics.

Whereas supporting most JVM languages at this point may be contrary to Guix values (TCR does not support either license compliance analysis and repository linking metadata), nothing prevents supporting any other JVM package repository that supports both those features. Fortunately, Clojars[5], the *de facto* repository for Clojure packages, supports the features Guix requires. Also, as Clojure (a libre, modern LISP dialect) is among the most popular JVM languages, Clojars would be a very valuable addition to the list of Guix importers. Finally, as Guix is also written in a LISP dialect (Scheme), bringing together these two kindred functional programming communities would have the additional benefit of bringing new potential contributors to Guix.

[1] Guix does include many Java packages, but none of them support their dependencies

[2] https://web.archive.org/web/20181202112953/https://www.oracle.com/technetwork/articles/java/afterglow2013-2030343.html

[3] https://en.wikipedia.org/wiki/List_of_JVM_languages

[4] The Central Repository Search Engine: https://search.maven.org/

[5] Clojars: https://clojars.org

Stage	Deliverable	Phase
-1	Minor patch sent for review.	Review
0	Patch reviewed/reworked	Bonding
1	Wrapper/Interface for `clojure.tools.deps`	Partial Evaluation #1
2	Basic Clojars importer integrated with GitHub	Partial Evaluation #2
3	Advanced Clojars importer integrated with GitHub	Final Evaluation
(4)	Possible extensions, time permitting: other forges.	(Extra)

Table 1: Stages and Deliverables

3 Solution Overview

There are many tools used to access JVM package repositories and solve the dependency graph traversal problem. Depending on the programming language used, the most popular tools include Ant, Maven, Gradle, Ivy... In the specific case of Clojure, the main tools used are: `clojure.tools.deps`[6], `boot`[7], and `leiningen`[8]. Whereas they all support Clojars as their main package source, each one has their own strategy for satisfying dependencies: while `clojure.tools.deps` implements its own heuristics, the other two heavily depend on Maven's.

Considering that the dependency resolution is a problem so complex to become a potential threat for the success of this project[9], my strategy in this regard is to leverage on the existing tools that already solve the problem. In this regard, `clojure.tools.deps` seems the best candidate for this. Whatever the tool that ends up being used, charting the problem domain via visualization is a complementary tactic to be considered for this. Clearly, this is one of the first problems to solve. In the unlikely case that none of the existing tools can successfully solve the dependency problem within Guix, potentially limited or simplified heuristics would have to be developed; support would then be restricted to smaller and/or acyclic dependency graphs.

A very important Guix guiding principle is reproducibility. Of course, this requires the ability to rebuild the package from source. For any given Clojure package, aside from obtaining its full dependency graph, reproducibility would require two features already part of Guix: compiling the package itself and and obtaining the corresponding source code for all its Clojure dependencies. With respecto to the first item, Clojure itself is already packaged into Guix. With respect to the last item, the new Clojars importer will make use (and extend, if needed) the GitHub Guix updater. This is because GitHub has become the *de facto* go-to place when it comes to looking for a free software project. Once the importer works, other forges could be added as sources, such as GitLab and BitBucket.

Considering that in may cases Clojure packages are replicated across Clojars and TCR, the importer could work in the following way[10]: when requesting a package, first, it would try importing it from Clojars. If this fails, it would import the package from TCR, without source nor licensing information. In short, to try multiple repositories, from the most desirable to the least desirable. In all cases, it would use the existing dependency information.

4 Implementation Plan

4.1 Stages & Deliverables

I plan to divide the project into three main stages, to be validated on each evaluation. If time permits, there may be extensions. See 1.

4.2 Timeline & Milestones

Notes:

- I have considered 5-days weeks for all periods, so there can be slack time if needed.
- There are also three planned slack week, to be able to catch up with delayed work if needed.
- The number of activities / deliverables grows in each phase: over time, I expect to become more productive.

In 2 is the timeline for the expected Activities and Deliverables, on a weekly basis. In 3 you will find the specific dates for each work week.

5 Communication

My timezone: UTC -3 (minus three).

I plan to communicate with my mentor(s) in the following ways:

- Mainly:
  - A weekly one-on-one videocall for PMC (planning, management, and control).
  - Via email, both through the Guix-devel mailing list and privately. In both cases, I estimate a maximum reply delay of 24 working hrs.
- Exceptionally:

[6] clojure.tools.deps: https://github.com/clojure/tools.deps.alpha.git
[7] boot: https://github.com/boot-clj/boot.git
[8] leiningen: https://github.com/technomancy/leiningen
[9] Re: [GSoC 2020] Clojure importer for Guix? (Pjotr Prins) https://lists.gnu.org/archive/html/guix-devel/2020-03/msg00294.html
[10] Thanks to Julien Lepiller for the first draft of this mechanism.
[11] The first Review week is actually *half* a week.
[12] Coding #1 includes Partial Evaluation #1.
[13] Coding #2 includes Partial Evaluation #2.
[14] Coding #3 includes Partial Evaluation #3.

Activities	Deliverables	Stage	Applic.		Review					Bonding				
		Week	A0	A1	R0	R1	R2	R3	R4	B0	B1	B2	B3	C0
Start flicking through Guix's code. [done]			X	X										
Set up a development environment. [done]			X	X										
Learn about Guix's internal processes and culture.			X	X										
Start reading Guix documentation. [in progress].			X	X										
Start exploring possible approaches to implement proposed features [in progress].			X	X	X	X	X			X	X			
	Minor patch sent for review.						X							
Finish reading introductory material.					X	X	X	X	X					
Experiment with possible approaches to implement proposed features.					X	X	X	X	X	X	X	X		
Engage with the Community and develop possible features not initially considered.					X	X	X	X	X					
Continue hacking into Guix's codebase to get to know it better.					X	X	X	X	X	X	X	X	X	
Explore options to implement proposed features.										X	X	X	X	
Re-assessment of implementation difficulty of proposed features.										X	X	X	X	
	Patch reviewed/reworked								(X)		X			
Research possible solutions to dependency solving.										X	X	X	X	X
Test Wrapper/Interface in a subset of "simple" packages.														X
	Wrapper/Interface for `clojure.tools.deps`													
Integrate GitHub importer.														
Test Clojars importer in a subset of "simple" packages.														
	Basic Clojars importer integrated with GitHub													
Test Clojars importer in a subset of "complex" packages.														
	Advanced Clojars importer integrated with GitHub													
Test Clojars importer in a subset of non-GitHub packages.														
	Possible extensions, time permitting: other forges.													

Table 2: Timeline for Activities and Deliverables (weekly).

Stage	Week	Start	End
Application	A0	March 16th	March 31st
	A1		
Review	R0[11]	April 1st	April 3rd
	R1	April 6th	April 10th
	R2	April 13th	April 17th
	R3	April 20th	April 24th
	R4	April 27th	May 1st
Bonding	B0	May 4th	May 8th
	B1	May 11th	May 15th
	B2	May 18th	May 22th
	B3	May 25th	May 29th
Coding #1[12]	C0	June 1st	June 5th
	C1	June 8th	June 12th
	C2	June 15th	June 19th
	C3	June 22th	June 26th
	(S0)	June 29th	July 3rd
Coding #2[13]	C4	July 6th	July 10th
	C5	July 13th	July 17th
	C6	July 20th	July 24th
	(S1)	July 27th	July 31th
Coding #3[14]	C7	August 3rd	August 7th
	C8	August 10th	August 14th
	C9	August 17th	August 21th
	(S2)	August 24th	August 28th
Extra Time (for emergencies/ extra work)	(S3)	August 31st	September 4th
	(S4)	September	September 8th

Table 3: Dates per week

- In case of urgency, I may use Matrix or a similar messaging app - TBD with my mentor(s).
- In case of emergency, I may use phone/Internet calls - TBD with my mentor(s).

A About the Applicant

A.1 Qualification and Background

A.1.1 Free Software Involvement

I am a GNU/Linux user since 1999, and I was very involved into Free Software advocacy in my hometown until 2006. During those years, I co-founded and chaired the local Free Software Users Group, and spearheaded the inclusion of Free Software as a subject of discussion into the local University. Specially worth mentioning is the organization of the first editions of the FLISoL (Latin American Free Software Install Fest) between 2004 and 2006.

I have contributed documentation and code to at least fifteen different Free Software projects. Last year, I was accepted as a core member of perun, a static web generator written in Clojure.

A.1.2 Work & Education

I have a Masters in Information Systems. I worked as a developer for a short time, but realized I am quite fond of Academia. I worked as a Researcher, working mainly on Evolution and Quality of Free Software Projects, and got two short, peer-reviewed papers published. Currently, I am a Lecturer at UNTREF (Caseros, Buenos Aires, Argentina). I strive to broaden my knowledge beyond IT: I am currently pursuing a B.A. in Tango Dance at the National University of Arts.

A.1.3 Functional Programming and Me

My current language of preference is Clojure, after I completely rejected OO's limitations. I started with Functional Programming five years ago, after having to deal with OO's limitations for simply too long. Whereas I am mainly focused on Clojure, I'm interested in LISPs in general -this being the main reason I applied to Guix to work with Scheme.

A.2 Online Profiles

- GitHub: allentiak
- LinkedIn: LeandroDoctors
- Google Scholar: Leandro Doctors

A.3 Availability

I am aware that GSoC requires an availability of *around* 30 hours per week. As with any other project, there will be weeks that may require slightly more, and weeks that may require slightly less.

I will be able to allocate the required time for GSoC-related activities. I am able to keep up with more than one project at the same time. Specially in this case, as it will involve two very complementary projects: my part-time

Algorithms teaching job (from which I will have a break in July and August), and coding for GSoC. There is a chance that I may have to travel for around a week; If this happens, I will discuss this in advance with my mentor(s), so we can plan ahead to minimize any possible interference.