Architecture Decision Record: Module Structure & Loading

Context

Arachne needs to be as modular as possible. Not only do we want the community to be able to contribute new abilities and features that integrate well with the core and with eachother, we want some of the basic functionality of Arachne to be swappable for alternatives as well.

ADR-002 specifies that one role of modules is to contribute schema to the application config. Other roles of modules would include providing code (as any library does), and querying and updating the config during the startup process. Additionally, since modules can depend upon each other, they must specify which modules they depend upon.

Ideally there will be as little overhead as possible for creating and consuming modules.

Some of the general problems associated with plugin/module systems include:

- Finding and downloading the implementation of the module.
- Discovering and activating the correct set of installed modules.
- Managing module versions and dependencies.

There are some existing systems for modularity in the Java ecosystem. The most notable is OSGi, which provides not only a module system addressing the concerns above, but also service runtime with classpath isolation, dynamic loading and unloading and lazy activation.

OSGi (and other systems of comparable scope) are overkill for Arachne. Although they come with benefits, they are very heavyweight and carry a high complexity burden, not just for Arachne development but also for end users. Specifically, Arachne applications will be drastically simpler if (at runtime) they exist as a straightforward codebase in a single classloader space. Features like lazy loading and dynamic start-stop are likewise out of scope; the goal is for an Arachne runtime itself to be lightweight enough that starting and stopping when modules change is not an issue.

Decision

Arachne will not be responsible for packaging, distribution or downloading of modules. These jobs will be delegated to an external dependency management & packaging tool. Initially, that tool will be Maven/Leiningen/Boot, or some other tool that works with Maven artifact repositories, since that is currently the standard for JVM projects.

Modules that have a dependency on another module must specify a dependency using Maven (or other dependency management tool.)

Arachne will provide no versioning system beyond what the packaging tool provides.

Each module JAR will contain a special `arachne-modules.edn` file at the root of its classpath. This data file (when read) contains a sequence of *module definition maps*.

Each module definition map contains the following information:

- The formal name of the module (as a namespaced symbol.)
- A list of dependencies of the module (as a set of namespaced symbols.) Module dependencies must form a directed acyclic graph; circular dependencies are not allowed.
- A namespace qualified symbol that resolves to the module's *schema function*. A schema function is a function with no arguments that returns transactable data containing the schema of the module.
- A namespace qualified symbol that resolves to the module's *configure function*. A configure function is a function that takes a configuration value and returns an updated configuration.

When an application is defined, the user must specify a set of module names to use (exact mechanism TBD.) Only the specified modules (and their dependencies) will be considered by Arachne. In other words, merely including a module as a dependency in the package manager is not sufficient to activate it and cause it to be used in an application.

Status

Proposed

Consequences

- Creating a basic module is lightweight, requiring only:
  - writing a short EDN file
  - writing a function that returns schema
  - writing a function that queries and/or updates a configuration
- From a user's point of view, consuming modules will use the same familiar mechanisms as consuming a library.
- Arachne is not responsible for getting code on the classpath; that is a separate concern.
- We will need to think of a straightforward, simple way for application authors to specify the modules they want to be active.
- Arachne is not responsible for any complexities of publishing, downloading or versioning modules
- Module versioning has all of the drawbacks of the package manager's (usually Maven), including the pain of resolving conflicting versions. This situation with respect to dependency version management will be effectively the same as it is now with Clojure libraries.
- A single dependency management artifact can contain several Arachne modules (whether this is ever desirable is another question.)
- Although Maven is currently the default dependency/packaging tool for the Clojure ecosystem, Arachne is not specified to use only Maven. If an alternative system gains traction, it will be possible to package and publish Arachne modules using that.