Thursday, 10 September 2015

AGI Experimental Framework: A platform for AGI R&D

submit

By Gideon Kowadlo and David Rawlinson

Introduction

We've been building and testing AGI algorithms for the last few years. As the systems become more complex, we have found it ever more difficult to run meaningful experiments. To summarise, the main challenges are:

- testing a version of the algorithm repeatedly and over some range of parameters or conditions,
- scaling it up so that it can run quickly,
- debugging: the complexity of the 'brain' makes visualising and interpreting its state almost as hard as the problem itself!

Platforms for testing AIs already exist, such as the Arcade Learning Environment (http://www.arcadelearningenvironment.org/). There are also a number of standard datasets (http://homepages.inf.ed.ac.uk/rbf/IAPR/researchers/MLPAGES/mldat.htm) and frameworks for testing them. What we want is a framework for understanding the behaviour of an AI that can be applied successfully to any problem - it is supposed to be an Artificial *General* Intelligence, after all. The goal isn't to advance the gold-standard incrementally; instead we want to better understand the behaviour of algorithms that might work reasonably well on many different problems.

Whereas most AI testing frameworks are designed to facilitate a particular problem, we want to facilitate understanding of the algorithms used. Further, the algorithms will have complex internal state and be variably parameterised from small instances on trivial problems to large instances - comprising many computers - on complex problems. As such there will be a lot of emphasis on interfaces that allow the state of the algorithm to be explored.

These design goals mean that we need to look more at the enterprise and web-scale frameworks for distributed systems (https://en.wikipedia.org/wiki/Distributed_computing), than test harnesses for AIs. There's a huge variety of tools out there: Distributed filesystems (https://en.wikipedia.org/wiki/Apache_Hadoop), cloud resourcing (such as Elastic Compute (https://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud)), and cluster job management (e.g. many scientific packages available in Python (https://wiki.python.org/moin/ParallelProcessing)). We'll design a framework with the capability to jump between platforms as available technologies evolve.

Developing distributed applications is significantly harder than single-process software. Synchronization and coordination is harder (c.f. Apache Zookeeper (https://zookeeper.apache.org/)), and there's a lot of crud (http://docs.mongodb.org/master/crud/) to get right before you can actually get to the interesting bits (i.e. the AGI). We're going to try to get the boring stuff done nicely, so that others can focus on the interesting bits!

Foundational Principles

- **Agent/World conceptualisation**
  - For AGI, we have developed a system based around Experiments, with each Experiment having Agents situated in a World.
- **Reproducible**
  - All data is persisted by default so that any experiment can be reproduced from any time step.
- **Easy to run and use**
  - Minimal setup and dependencies.
  - No knowledge of the implementation is required to implement a custom module (primarily the intelligent Agent or World in which it operates).
- **Highly modular (Scalability)**
  - Different parts of the system can be customised, extended or overridden independently.

    - Distributed architecture (Scalability)
    - Modules can be run on physically separated machines, without any modification to the interactions between modules (i.e. the programmer's perspective is not affected by scaling of the system to multiple computers).
- **Easy to develop**
    - Code is open source.
    - Code is well documented.
    - All API's well documented and using standard protocols (at the moment RESTful (https://en.wikipedia.org/wiki/Representational_state_transfer), in future could be websockets (https://en.wikipedia.org/wiki/WebSocket) or other).
- **Explorable / Visualisable**
    - High priority placed on debugging and understanding of data rather than simply efficiency and throughput. We don't yet know what the algorithm should look like!
    - All state is accessible, relations are can be explored.
    - Execution is on demand (step-by-step) or automatic (until criteria, or batches of experiments completed).
    - It must be easy for anyone to build a UI client that can explore the state of all parts of the system.

Conceptual Entities

We have defined a number of components that make up an experiment. We refer to these components as Entities, and give them a specific interface (https://en.wikipedia.org/wiki/Application_programming_interface).

- **World**
    - The simulated environment within which all the other simulated components exist.
- **Agent**
    - The intelligent agent itself. It operates within a World, and interacts with that World and (optionally) other Agents via a set of Sensors and Actuators.
- **Sensor**
    - A means by which the Agent senses the world. The output is a function of a subset of the World state. For example, a unidirectional light sensor may provide the perceived brightness at the location of the sensor.
- **Actuator**

    - A means by which an Agent acts on the World. The output is a simulated physical action. For example, a motor rotating a wheel.
- **Experiment**
    - The Experiment Entity is a container for a World, and a set of Agents (each of which have a set of Sensors and Actuators), and an Objective Function which determines the terminating condition of the experiment (which may be a time duration).
- **Laboratory**
    - A collection of Experiments that form a suite to be analysed collectively. This may be a set of Experiments that have similar setups with minor parameter variations.
- **ObjectiveFunction**
    - The objective function computes metrics about the World and/or Agents that are necessary to provide Supervised Learning or Reinforcement Learning signals. It might instead provide a multivariate Optimization function. The ObjectiveFunction is a useful encapsulation because it is often easy to separate objective measurements from the AI that is needed to achieve them.

Architecture

To enforce good design principles, the architecture is multi-layered and highly modular. Multiple layers (also known as multi-tier architecture) allows you to work with concepts that are at the appropriate level of abstraction, which simplifies development and use of the system.

Each entity is a module. Use of particular entities is optional and extensible. A user will inherit the entities that they choose, and implement the desired functionality. Another modularisation occurs with the AGIEF Nodes. They communicate via interprocess conventions so that components can be split between multiple host computers.

Interprocess communication occurs via a central interface called the Coordinator, which is a single point of contact for all Entities and the shared system state. This also enables graphical user interfaces to be built to control and explore the system.

These concepts are expanded in the sections below.

Design Considerations

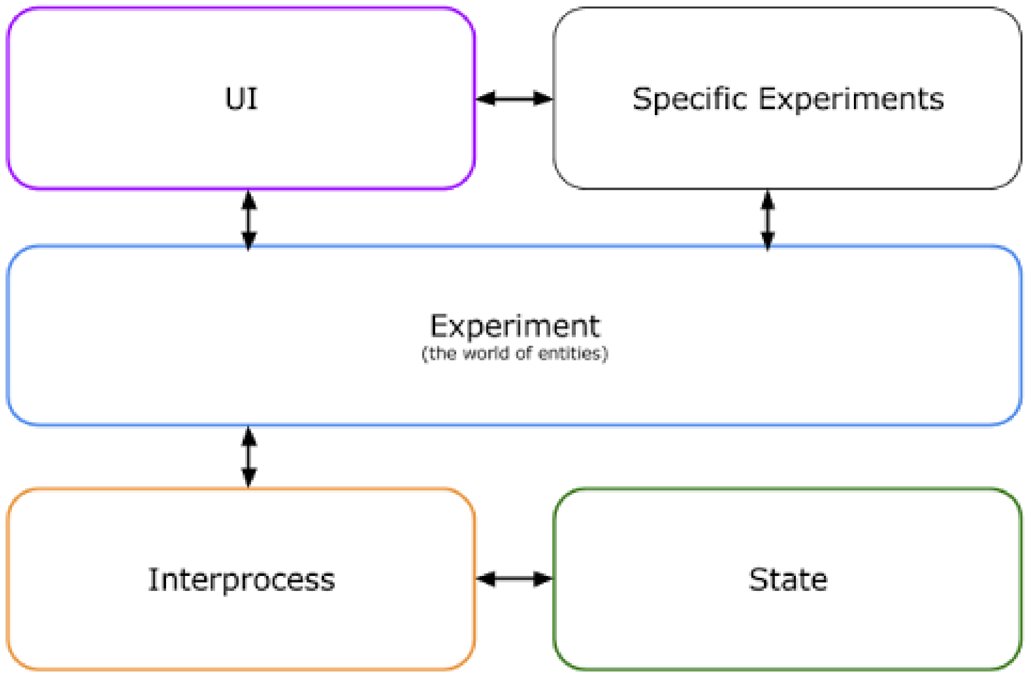
The various components of the system may have huge in-memory data-structures. This is an important consideration for persisting state, distributed operation, and ability to visualise the state.

Processing to update the state of Worlds and Agents will be compute-intensive. Many AI methods can easily be accelerated by parallel execution. Therefore, the system can be broken down into many computing nodes, each tasked with performing a specific computational function on some part of the shared system state. We hope to support massively parallel hardware such as GPUs in these compute nodes.

We will write the bulk of the framework and initial algorithm implementations in Java. Others can extend on this, or develop against the framework in other languages. We will also write a graphical user interface using web technologies that will allow easy management of the system.

Perspectives on the system design

The architectural layers are shown in the diagram below.

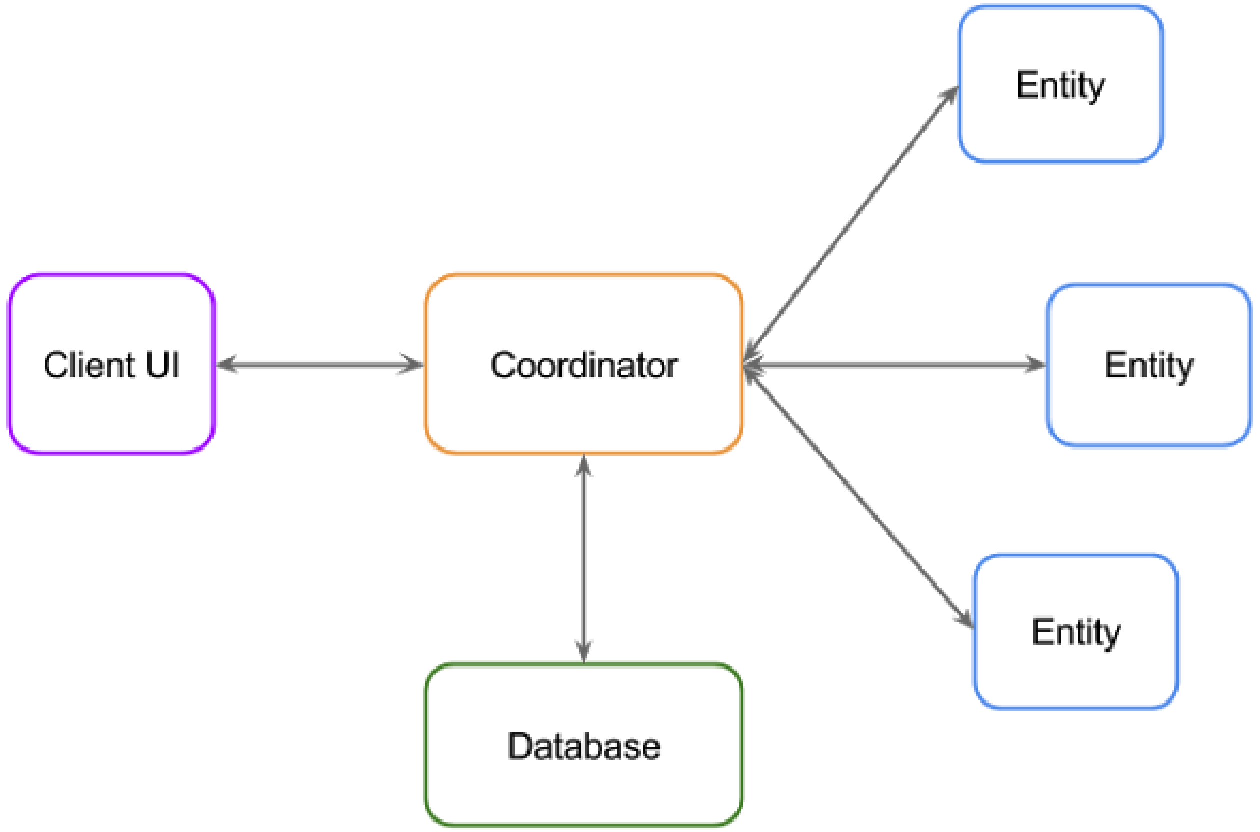


(http://1.bp.blogspot.com/-HcG6Yup0nsw/VfExsOOHDCI/AAAAAAAASRM/QFx4bEPqeUE/s1600/AGIEF%2BArchitectural%2BLayers%2B%25281%2529.png)
Figure 1: 'Architectural Layers'

Each layer is distinct, with strict separation. No layer has access to the layers above, which operate at a higher level of abstraction.

- **State:**
  - State persistence: storage and retrieval of state of all parts of the system at every time step. This comprises the shared filesystem.
- **Interprocess:**
  - Communications between all modules running in the system, locally and/or across a network.
  - Provides a single point of contact via a local interface, to any part of the system (which may be running in different physical locations), for both control signals and state.
- **Experiment:**
  - Provides all of the entities that are required for an experiment. These are expanded shortly.
- **UI:**
  - The user interface that an experimenter uses to run experiments, debug and visualise results.
  - The typical features would be:
    - set up parameters of an experiment,
    - run, stop, step through an experiment,
    - save/load an experiment,
    - visualise the state of any part of the experiment.
- **Specific Experiments:**
  - This is defined by the person experimenting with the system. For example, a specific Agent that seeks light, a specific World that contains a light source, and an objective function that defines the time span for operation.

Another perspective on the design is to view the Services and Entities and their lines of communication. The diagram is colour coded to indicate Layers, as per the diagram above.



(http://4.bp.blogspot.com/-rbR_VggsWsQ/VfEvIpGtm6I/AAAAAAAASQ4/BTH5yy6ejCU/s1600/Untitled%2Bdrawing.png)
Figure 2: 'Services and Entities'

The Coordinator and Database are services. The Coordinator is shown at the centre, as described earlier (Architecture section), being the primary point of contact for Entities and potentially other clients such as a Graphical User Interface.

A similar perspective is shown in an expanded diagram below that illustrates the Database API module and the distributed implementation of the Coordinator in the Interprocess layer, enabling Entities to run on separate machines. This is just one possible configurations; there can be multiple slaves, each with multiple entities.

Each bounding box indicates what we refer to as an AGIEF Node (or node for short). The node comprises a process that provides a context for execution of one or more entities, or other clients such as the GUI.

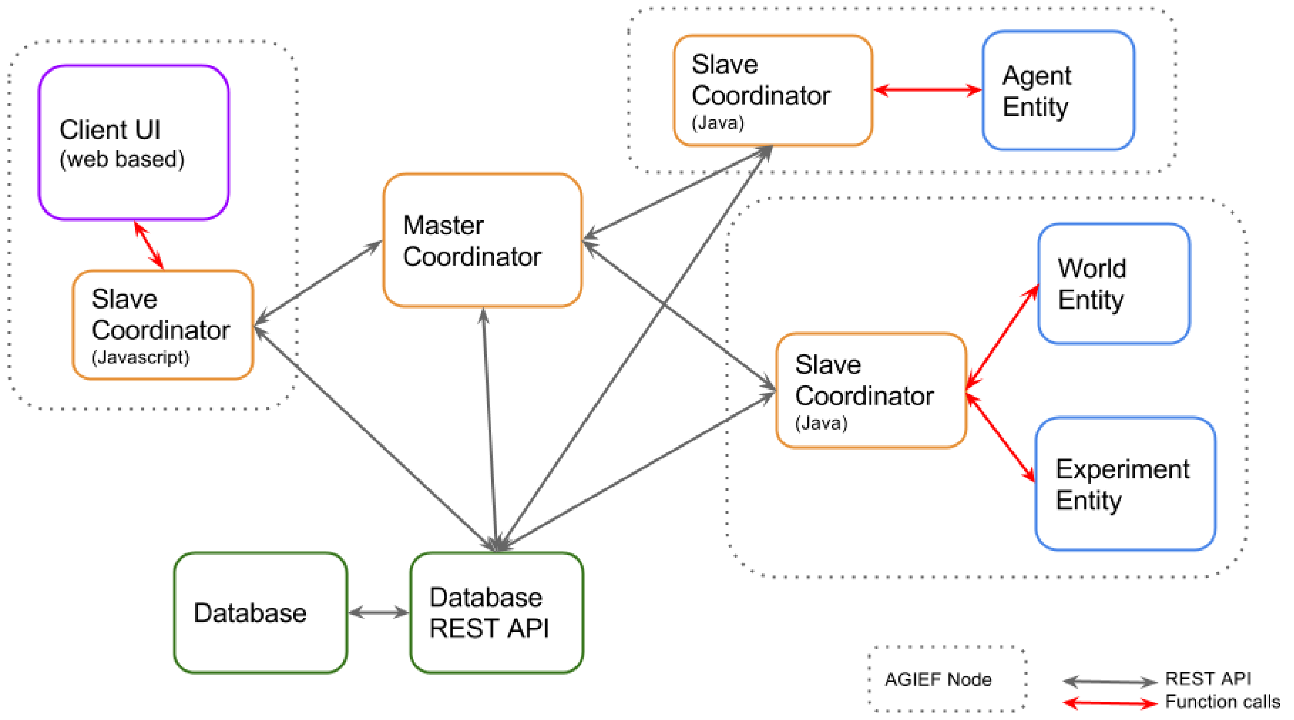
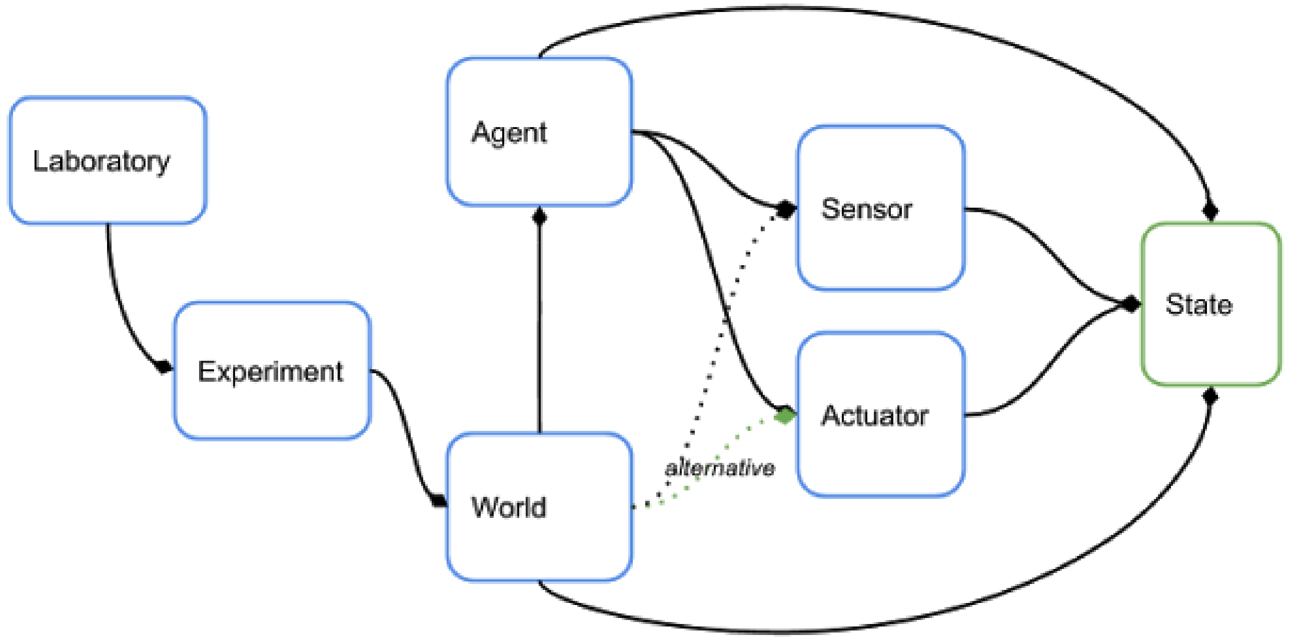


Figure 3: 'AGIEF Nodes'

We looked at popular No-SQL (https://www.mongodb.com/nosql-explained) web storage systems (basically key-value stores) which are very convenient and flexible due to the inherently dynamic, software-defined schemas (https://en.wikipedia.org/wiki/Database_schema) and HTTP interfaces. However, we have a relatively static schema for our data, on which we will build utilities for managing experiments and visualising data. In addition, relational databases (https://en.wikipedia.org/wiki/Relational_database) such as MySQL (https://www.mysql.com/) and PostgreSQL (http://www.postgresql.org/) are beginning to offer HTTP interfaces as well. Whether we pick a NoSQL or Relational Database, we will require a HTTP interface.

A third perspective is the data model that represents the system in its entirety. This is the model implemented in the database.



(http://2.bp.blogspot.com/-5I-TZIrVnSQ/VfEwIQ_19rI/AAAAAAAASRA/UoXzTYhb-P0/s1600/Data%2BModel.png)

Figure 4: 'Data Model'

The data model stores the entire system state, including hierarchy and relationship between entities, as well as the state of each entity. With a RESTful API (https://en.wikipedia.org/wiki/Representational_state_transfer) exposing the database, we have a shared filesystem accessible as a service, essential for distributed operation and restoring the system at any point in time.

Future Work

We will shortly be releasing an initial version of our framework and we'll post about the technology choices we've made, and some alternatives. We'll include a demonstration problem with the initial release and then start rolling out some more exciting algorithms and graphics, including lots of AI methods from the literature (we have hundreds in our old codebase ready to go).

Posted by Gideon Kowadlo (https://www.blogger.com/profile/06783501071538911513) at Thursday, September 10, 2015 (2015-09-10T17:08:00+10:00) (http://blog.agi.io/2015/09/agief-artificial-general-intelligence.html) (https://www.blogger.com/post-edit.g?blogID=1180536024131440638&postID=2948004019509866258&from=pencil)

G+1 Recommend this on Google

Labels: AGI (http://blog.agi.io/search/label/AGI) , AGIEF (http://blog.agi.io/search/label/AGIEF) , Architecture (http://blog.agi.io/search/label/Architecture) , Experimental Framework (http://blog.agi.io/search/label/Experimental%20Framework)

No comments :

Post a Comment

(https://www.blogger.com/comment-iframe.g?blogID=1180536024131440638&postID=2948004019509866258&blogspotRpcToken=5843234)

Enter your comment...

Comment as: Gideon Kowadl

Sign out

Publish Preview

Notify me

Newer Post (http://blog.agi.io/2015/10/quantum-computing-via-sparse.html)HomeOlder Post (http://blog.agi.io/2015/07/reading-list-july-2015.html)
(http://blog.agi.io/)

Subscribe to: Post Comments (Atom) (http://blog.agi.io/feeds/2948004019509866258/comments/default)