Animation of Self-Organising Resource Allocation using Presage2

Sam Macbeth, Jeremy Pitt, Julia Schaumeier and Didac Busquets

Department of Electrical & Electronic Engineering
Imperial College London, Exhibition Road, London, SW7 2BT, UK
Email: {samuel.macbeth06, j.pitt, j.schaumeier09, didac.busquets}@imperial.ac.uk

Abstract—This paper presents a demonstration of Presage2, a general purpose platform for developing animation and simulations of collective adaptive systems.

I. INTRODUCTION

This paper describes Presage2, a platform for developing animation and simulations of collective adaptive systems. It extends the original PreSage platform [1], adding support for declarative rule specifications and increasing modularity. Presage2 provides services to simulate large, heterogeneous populations, multiple different networks, inter-agent communication, policy modelling, the physical environment, event recognition, data logging and visualisation.

We describe the platform architecture and implementation (Section II), illustrate the process of constructing a simulator (Section III), and demonstrate an operational simulator (Section IV), used to perform experiments in self-organising resource allocation in an open multi-agent system [2].

II. ARCHITECTURE AND IMPLEMENTATION

A. System Architecture

Presage2 is designed to be as modular as possible. This enables reuse of components across systems, reduction in the complexity and interdependency of system components, and allows easy extension of the platform’s capabilities. Its functionality is determined by the composition of sets of modules:

- The state transformer function is composed of a set of action handlers which process the actions generated by agents and generate change sets to be applied to the environment and/or agent states.
- Agents’ perception of the environment and other agents is derived from services which read the raw system state and return data to the agent. This abstraction layer allows for simpler access for the agent designer, as the raw state data can be transformed into an easier to use format, as well as simulation of lossy perception or access limitations.

The platform does not impose many limitations on agents. Their agent function is invoked every timestep allowing them to submit zero or more actions. There are no constraints on computational complexity or memory usage, therefore any agent architecture can be used.

A multi-agent system in Presage2 is modelled as a discrete time driven simulation. A simulation run r consists of sequence of interleaved system states \( e_0 \rightarrow e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{n_1} \)

Agents generate actions from their observation of the environment and agents’ state. The set of all actions for a single timestep, combined with the system state, generates a new state for the next timestep. By enforcing this model on the execution of the simulation we ensure that system behaviour can easily be formalised in terms of the set of agent functions and the state transformer function.

B. Implementation

The platform is implemented as a Java library, providing interfaces and abstract classes for the user to extend. We also include libraries for common functions such as situated agents, ad-hoc networks and communication protocols which can be plugged into a simulation. Figure 1 shows the full set of available components.

![Figure 1. Presage2 components](image)

Every simulation timestep involves two stages. Firstly all agents are invoked and can submit actions. At this stage other modules may also be invoked if they have subscribed to run every timestep. Once all the agents have finished
gathering actions the state change sets are processed and applied to the system state.

As well as defining state transformers in pure Java code the platform is able to take declarative rules which modify state when certain conditions are true. We have an implementation which uses the JBoss Drools (www.jboss.org/drools) rule engine to store all system state as facts. These facts can be any Java object, as such complex data and relations can be stored and agents’ actions can be directly inserted into the engine. Rules can then be written in the form "when condition then action". The rules are then triggered every timestep to cause the insertion, modification and retraction of facts in the system state.

III. USAGE

A system implemented with Presage2 is minimally defined with:

- Simulation specification, including a set of parameters to initialise the simulation and the set of modules to load. These define the initial state of the simulation, the services which will be provided to agents in order for them to read the system state, and handlers for processing agent actions into state changes.
- One or more agent implementations. When loaded by the simulation these will be given the set of services available, then invoked every timestep and allowed to submit actions. There are no constraints on the internal architecture of the agent.

Once a valid simulation has been created it can be verified and executed with a set of parameters, either directly from the command line, or simulation runs can be defined and stored in a database to be loaded and executed later. This database integration extends to storing simulation data during simulation execution. The platform provides an API for storing key/value data and drivers for several database engines, including PostgreSQL and MySQL. This process is illustrated in Figure 2.

IV. SELF-ORGANISING RESOURCE ALLOCATION

In [2], we address the problem of resource allocation in an open system in an economy of scarcity. We use Ostrom’s principles of self-governing the commons [3] to define an institution, whose members use conventional rules to self-organise a resource allocation policy and ensure that outcomes of the policy are ‘fair’. Rescher reviews many canons of distributive justice, all of which have strengths and weaknesses [4]. His proposes instead legitimate claims: that justice consists of determining which canons are appropriate in context, how they are accommodated in case of plurality, and how they are reconciled in case of conflict.

We have used PreSage2 to construct an experimental platform to examine the self-organisation of ‘fair’ resource allocation. We define multiple clusters of agents. Each cluster performs resource allocation in a series of ‘rounds’, in each round each agent makes a provision of resources, an allocation is computed according to the policy, following which each agent makes an appropriation. As this is an open systems, agent may ‘cheat’, e.g. by not complying with the provision or appropriation rules.

We define a set of legitimate claims that prioritise resource allocation according to resource appropriation, resource provision, etc. Plurality is accommodated by treating each claim as a ‘voter’ in a Borda count protocol which implements the resource allocation policy. Conflict is reconciled by attaching a weight to each claim, and at the end of each round, the agents self-organise the weights on each claim.

This implementation supports controlled experimentation with different agent behaviour profiles, population sizes, population compositions, cluster configurations and allocation policies, as well as different sets of legitimate claims, different weights, and so on. PreSage2 can be run in batch mode to explore the effect of changes to different independent variables. The data logging and visualisation services of PreSage2 are used process the datasets generated by the experiments, as shown in Figure 2 (step 4: bottom right).

V. CONCLUSION

We have described the design, implementation and usage of the Presage2 platform, and the implementation of self-organising resource allocation in a collective, adaptive multi-agent system. The platform itself is open source and available under the LGPL license from www.presage2.info.

REFERENCES