Creating an Abstract Definitive Machine Plugin for JS-EDEN

Ruth King - 1037559

The aims of this project will be to develop a prototype Abstract Definitive Machine (ADM) plugin for JS-EDEN, and then develop an example model using this. A language for ADM will be defined and then translated using Javascript to be used within JS-EDEN. This will be the main section of the project. The usefulness of ADM will then be demonstrated through direct use as a computational model, as a simulation of a systolic array. Hopefully a further example use will be provided by another student who has already experimented with similar work.

The motivation of ADM is to provide a truly multi-agent architecture for Empirical Modelling (EM). In a multi-agent system computation should obviously take place in parallel, as agents carry out actions simultaneously. The state of the system should be definitive and dependencies between things in the environment should be redefinable by all agents within the system.

An ADM program consists of a group of entities, each having a set of definitions and guarded actions. The way of defining these entities is the language of the ADM, to be translated. A computational step consists of evaluating the guards for the actions of all entities then executing these actions in parallel. Each action can be a redefinition of any observable or creating or deleting another entity. Computation continues in this way until no guards evaluate to true. If a conflict occurs, where two actions try to concurrently change the same observable, computation halts or pauses and can wait for user instruction of how to resolve the conflict. Throughout computation all definitions are stored in the same definition store, so they are not encapsulated within entities.

In this way entities roughly correspond to objects - they intuitively group together definitions of a single conceptual “thing”. Each computational step changes state within the system, in a way well defined by the actions of entities. Humans can interact with the ADM in different ways, perhaps by changing the entities themselves or by adding new entities, and by resolving conflicts. Entities should be created through templates. Each machine step should be slow enough for the human agent to recognise the states of the system, to give them the possibility of interacting and understanding.

There are three uses for ADMs. The first of these, which will be demonstrated in this project, is directly with a computational model. This allows the modelling of deterministic automated behaviour similar to that of a traditional computer. The second use is to model concurrent systems using Empirical Modelling, for example the Clayton Tunnel model where many agents act within the system concurrently. The third is as the use of Eden - all EM models should be expressible in ADM form.

The model chosen to demonstrate the uses of ADM for the first use mentioned is as a simulation of a systolic array. This is mentioned as an example by Beynon, Slade and Yung in 1988 in the first paper describing the ADM model for parallel computation. A simulation of a systolic array was used to illustrate the ADM as it well fits the model - each processor can be represented as an entity, where at each computational step if it has input (this condition is provided by the guard on observables defined by the processor) it performs an assignment.

References to be used:
W.M.Beynon. Parallelism in a definitive programming framework. Parallel Computing 89, Advances in Parallel Computing Vol 2, North-Holland, 425-430, 1990.
W.M.Beynon, M.D.Slade and Y.W.Yung. Parallel computation in definitive models. CONPAR'88, British Computer Society Workshop Series CUP, 359-367, 1989.
These papers describe the motivation for ADM as well as some example uses.
M D Slade Parallel Definitive Programming MSc thesis, Warwick Univ 1989
Slade’s master thesis describes his implementation of the ADM for Eden.
I will also look at the different translators from ADM to Eden whose source code is available.
The notes on ADM from previous years of the CS405 course will also be studied.

As the model for this coursework would be fairly simple and most implementation work within the plugin there will probably be more weighting with the report, which will explain the motivations behind the ADM and the implementation choices made.