Creating an Abstract Definitive Machine Plugin for JS-EDEN

1037559

Abstract

The Abstract Definitive Machine (ADM) is a method of modelling and simulating concurrent systems using states and transitions within a definitive environment. Entities are defined and instantiated with collections of variable definitions and guarded actions which depend on and redefine the system state. These describe the state changing privileges of the entity. At each computational step a selection of actions with true guards is processed. This paper describes an attempt to develop a prototype ADM plugin for JS-EDEN, taking a unique human perspective with the modeller at the centre, and illustrates some uses of the plugin through example models. The paper will then discuss the results of the attempt, including possible future work and extensions.

1 Introduction

The concept of the ADM was one of the main themes during the early work of the Empirical Modelling (EM) Project. The motivation for this was to create a "machine model with which to express context-sensitive parallel redefinition" (Beynon, 1990), and to give a more complete picture of a concurrent system than previously used approaches. This style attempts to avoid the problems of state consistency encountered with more traditional methods through explicitly defined state and dependency. With this the handling of side-effects from redefinitions is automatic.

The computational body of the ADM consists of a collection of instantiated entities, which are defined statically. Each entity consists of a list of variable definitions and a collection of guarded action sequences depending on system state. Each action in a sequence may redefine variables within the system or instantiate and delete other entities. Within this framework entities represent groups of definitions and actions which share a "common extent in time" (Beynon et al., 1988), similar to objects in object oriented languages. At each "step" of computation, the guards of all actions are evaluated and actions with true guards are executed simultaneously in parallel. Execution terminates when none of the guards evaluate to true. The ADM represents a state-transition model, where the state, originally defined by the entity definitions, is evaluated and redefined at each step. The pattern of state transitions defines the behaviour of the system being modelled.

Actions being executed concurrently may cause conflicts if both try to redefine the same state variable in different ways or through circular dependencies. There are various ways to solve these conflicts. One method is to execute until a conflict arises and then prompt the user as to how to resolve the conflict. Another method is to introduce a human as the central "agent", who selects a set of (non-conflicting) actions to execute at each step. This human perspective gives the human user complete control over system state and allows empirical modelling and experimentation to take place more freely. It also improves the ease of arbitrary intervention as the user has control over the speed of the computation, as actions are being executed at their command.

There have been three uses defined for ADMs. The first of these is directly through a computational model. This allows the modelling of deterministic automated behaviour similar to that of a traditional computer, as detailed in Beynon et al. (1988). 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[1]. The third is as the use of EDEN - all EM models should be expressible in ADM form.

2 Previous Work

A key motivation for the ADM is as an animator for LSD accounts. LSD accounts describe the roles of a collection of state changing agents within an environment, where each agent's perceptions and capabilities are explicitly defined (Beynon et al., 1990). LSD is an aid to designing concurrent systems, and takes into account the belief that the designer conceives the behaviour of the system as interactions between different agents. Each of these agents is privileged to

[1]`http://empublic.dcs.warwick.ac.uk/projects/claytontunnelChanHarfield2005/`

carry out different actions under given circumstances, as described by its protocol. The consequences of an agent's actions are redefinitions of state within the system.

Multiple attempts have been made at implementing versions of the ADM in the past. The first of these was developed by Slade (1990) as an attempt to animate LSD accounts. The animation of LSD is important in linking the theory of the components and behaviour of the system to the actual results of the agents in the system acting together. However there are a few key differences between LSD accounts and ADM models. One of these is that agents within LSD accounts act sequentially, whereas entities within the ADM are allowed to interleave sequences asynchronously with other available actions. Another is that the varying speeds of entity reactions and actions is additionally open to interpretation in LSD accounts, whereas in the ADM this must be well defined in order for the entity to be implementable. In these ways the "spirit" of agents in LSD and entities in the ADM differs slightly.

Since the initial work by Slade a number of translators from "ADM" languages to EDEN have been created. Notably the ADM was linked to EDEN notation for the purposes of graphical animation by Simon Yung. However there were a number of flaws with these translators and they have become out of date with modern computers and EM tools.

One example model that was created using these translators is the five-a-side football model developed by Turner[2]. This model was originally defined using LSD notation and then implemented in an ADM language, with each football player as an entity within the system. However entities could not be instantiated using parameterised templates so Turner had to resort to a lot of repetition defining each football player entity.

3 Aims of the Prototype Development

The main aim of developing the prototype was to bring the ADM up-to-date with modern EM tools. Currently the a large proportion of EM development is done using JS-EDEN, a tool created by Tim Monks for his MSc project that allows the creation of models using EDEN notation through web browsers[3]. Following the work of previous ADM interpreters the prototype also aims to add slightly different functionality and allow more experimentation with different flavours of the ADM.

3.1 Human Perspective

The current state of the ADM defines which actions instantiated entities are *privileged* to act given the current system state. It is not necessary that all these actions be performed, as the behaviour being modelled may not be deterministic. In particular LSD accounts do not enforce that all actions be carried out and rather that the choice between commands with true guards is nondeterministic (Beynon et al., 1990). For these reasons it was decided to incorporate a human perspective into the prototype. Within this at each step the human user is presented with a list of available actions for each instantiated entity given the current state and they may make a selection of what is to be performed in the current step. In this view the user can be seen as a "prototypical agent" who chooses the state transitions at each stage (Beynon, 1990). It is also useful empirically to step through the state transitions in this way, as it becomes clear how the choices of each agent affect the state of the system.

3.2 Parameterised Templates

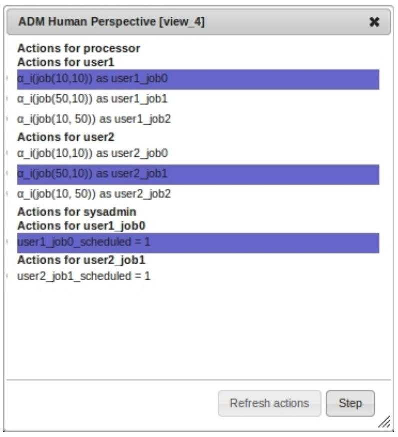
A second aim of the prototype was to allow the user to specify generic template entities and then instantiate these using parameters. This was inspired by Turner's football example, which required the creation of many very similar entities with slightly different definitions. This also allows the prototype to more easily support a wider range of models, as it is fairly common for a system to incorporate a large number of similar agents. A further example is the systolic array as described in Beynon et al. (1988), where each processor is a separate parameterised entity.

4 Discussion

The design and implementation of the prototype are discussed in detail in the attached model documentation, and will be described here briefly. Views exist for creating generic templates and instantiating these with parameters described above, or alternatively templates and entities can be created through a view to input code using a custom ADM language. A list of currently instantiated entities along with their definitions and guarded actions is available. Through

[2]http://empublic.dcs.warwick.ac.uk/projects/footballTurner2000/

[3]http://jseden.dcs.warwick.ac.uk/master/



```
ADM Human Perspective [view_4]
Actions for processor
Actions for user1
α_i(job(10,10)) as user1_job0
α_i(job(50,10)) as user1_job1
α_i(job(10, 50)) as user1_job2
Actions for user2
α_i(job(10,10)) as user2_job0
α_i(job(50,10)) as user2_job1
α_i(job(10, 50)) as user2_job2
Actions for sysadmin
Actions for user1_job0
user1_job0_scheduled = 1
Actions for user2_job1
user2_job1_scheduled = 1
```

Figure 1: Human perspective view, showing actions selected for execution.

the JS-EDEN symbol viewer the status of all agent definitions and guards can be seen (guards are created as EDEN variables with a dependency on their boolean definition). The human perspective view presents the user with currently allowed actions for each entity given the current state and multiple actions can be selected at each computational step. The existence of additional actions in a sequence is shown visually. EDEN animations through definition can be interleaved and included within the ADM as both share the same definition store.

The prototype ADM gives a new human perspective on the ADM which hasn't previously been experimented with. Within it entities can be created with nondeterministic behaviour, as controlled by the experimenter or modeller. This allows a wider range of models to be created that were not possible in previous ADM implementations (without some degree of pseudo randomness), and allows the exploration of a much greater range of concurrent multi-agent systems. Within the prototype the user is able to act as a kind of super entity with control over all the observables in the system. This is possible as all the protocols for the instantiated entities depend on observables in the JS-EDEN environment. By changing these the user is able to "dynamically impose appropriate scenarios for action and interaction upon agents" or entities (Beynon, 1994), as well as having full choice over the currently privileged actions of all entities. The user may also instantiate or delete entities at any time using the interface provided by the prototype ADM, and redefine the guards of any actions.

4.1 Example Models

To illustrate the functionality of the prototype example models were created. These are described in the paragraphs below.

Firstly an initial model was constructed to experiment with the prototype and test the functionality. Within this model entities representing users within a system can submit jobs to be processed. These jobs are instantiated as new entities with parameterised requirements, and remove themselves after they have successfully scheduled and used their required resources. If too many jobs are submitted simultaneously the processor can break and require replacement through the actions of a sysadmin entity. This model illustrates the usefulness of entities being able to instantiate and remove other entities and gave an insight into the thought processes needed to define an ADM model. Initially the design required thinking of possible ways in which each entity within the model could change the state of the system, and then developing these iteratively through experimenting with the model and finding appropriate actions at each stage in the process.

To illustrate the expressiveness of the human perspective an LSD account for a cat flap was animated using the ADM prototype as described by Bridge[4]. The creation of this model also showed the ease of converting simple LSD accounts, as only minimal changes were required to convert the notation. In particular the protocol is directly translatable into actions with the state and derivates forming the definitions belonging to the entity, using dependency for the derivates. However the oracles and handles, relating to observables that are visible and changeable by the entity, are not translatable, although systems should still be functional without this encapsulation. Bridge's cat flap account includes a cat agent whose behaviour is nondeterministic, for example the cat may intend to travel through the cat flap and then change his mind halfway through. Using the human perspective it is possible to model this and explore the consequences of different decisions of the cat. Although this is a very simple model it illustrates the power of the human perspective on less deterministic situations. This idea is extendible - a specific example of this is that in creating models of famous railway accidents a number of situations could be explored. By choosing various combinations of actions for all entities involved many possible consequences can be experienced.

[4]http://www2.warwick.ac.uk/fac/sci/dcs/research/em/teaching/cs405-0708/concsys/concsyslecture4/lsdexamples/catflapiblsd

The prototype is also capable of modelling more machine like behaviour. In this paradigm, as the classical ADM, all actions with true guards are executed at each step. This is equivalent to selecting all actions in the human perspective, so the current prototype could easily be extended to incorporate this behaviour. However conflicts between actions would not be recognised. It is arguable that this is similar to the way machines act in the real world, where after the program has been created any conflicts may cause undefined behaviour. This idea was illustrated through the creation of a systolic array model, as described in Beynon et al. (1988). A hexagonal array of processors which perform matrix multiplication is represented by instantiating an entity for each processor, with time modelled by a time entity. At each computational step a number of processors work in parallel to multiply the elements of two input arrays and adapt an element in the output array. The human perspective also aids understanding this computational model, as the modeller is exactly aware of the actions carried out at each step. This allows experimentation, for example through observing the consequences of a single processor failing.

4.2 Issues

Implementing the human perspective aspect of the plugin using JavaScript caused some issues. In the current prototype the refreshment of available actions relies on a step being executed. If the definitions are then changed within the JS-EDEN input window and the value of guards is changed by dependency this won't automatically propagate to relevant changes in the lists of available actions. To alleviate this problem a refresh button was created to recheck the state of all guards. This is not a problem when using solely ADM input as the only redefinitions that occur will be through steps, although it is likely users experimenting with models in the ADM will also input their own redefinitions, as a type of superuser within the system.

The use of JavaScript also made certain aspects of the UI more difficult to implement. A key example of this is the illustration that actions are part of a sequence and there are more actions to follow. If the human perspective was implemented using JS-EDEN the following actions could more easily dynamically appear as computation progressed.

5 Future Work and Extensions

As well as generally improving the UI of the various views and providing more input validation and useful error messages there are a variety of extensions and improvements that could be made to the prototype. Ideas for these future developments will be discussed in this section.

Currently existing entities are not redefinable, and template refinements and instantiation of these would be required. This functionality is probably fairly key to using the ADM as a way of empirically creating a multi-agent model, as through experimentation the user may notice relevant actions and wish to redefine existing entities to incorporate these. This is supported by EM teaching literature, which suggests that "complex specifications for systems cannot be constructed without building and experimenting on components"[5]. Adding the ability to edit entities would give a greater scope for system development through systematic refinement of the model in an attempt to identify important behaviour defining factors (Beynon, 1994). Methods for redefining entities have already been created so this work would just consist of modifying the UI in such a way to allow this and ensuring all edit changes are propagated.

As discussed above available actions in the human perspective should react automatically to any redefinitions that affect the guards. Sequences of actions could also be displayed to the user more effectively, for example as a horizontal list. This would give the user a greater understanding of the consequences following their action selection in the future. To these ends future work could be suggested to reimplement the human perspective view using JS-EDEN code rather than JavaScript code. In this way the actions being displayed could depend on the state of the guards and update with this dynamically.

It could be suggested that if an action is part of a sequence the next action should only be available if the state allowing the original action to occur still causes the guard to be true. Alternatively a step could consist of multiple actions within the same sequence. However this doesn't seem to correspond to the idea of a single atomic step of computation. These possibilities would need to be explored and the best solution may depend on the application so it may be best to make a number available.

The implications of deleting an entity within the ADM are currently unclear and need further explo-

[5]`http://www2.warwick.ac.uk/fac/sci/dcs/research/em/teaching/cs405-0809/concsys/concsyslecture4/`

ration. When deleting an entity, should all guards and actions which refer to the definitions of that entity within other entities also be deleted? This makes sense when thinking of an entity as "associat[ing] variables and actions sharing a common extent in time" (Beynon et al., 1988), as when an entity is removed from the ADM its associated variables can also be inferred to be no longer relevant. However it might be the case that the entity is re-instantiated in the future, in which case physically deleting related actions and definitions would be incorrect. Possibly the best course of action would be to disable these definitions in some way. They could be redefined as $undefined$, however any action changing these would still function. All corresponding actions could also be disabled in some way, however they may also be part of a sequence of actions where some of which are still valid. The current prototype simply removes the entity instantiation and leaves its definitions within the definition store as their final value. This represents possible intended behaviour, for example where an agent is instantiated just to bring a certain definition into the system.

The ADM model for parallel computation creates a multi-agent environment where actions may be executed simultaneously and "atomically" in parallel. However currently it is only the single human user of the system who decides which actions are executed at each step. This could be extended to a more distributed model where different users take "control" over different entities and decide their actions at each step without knowledge of the actions being taken by other entities in the system outside their control. This takes into account the idea of oracles as defined in LSD, where the environment is not fully-observable to all entities. Steps would still take place in parallel, where each user makes a decision before the actions of any entity are evaluated. In this way conflicts would occur that were not foreseen by users, and potentially a "superuser" would have to also be defined to resolve any conflicts that arise. A weaker version of this multi-agent environment is already possible, as users could just decide amongst themselves which entity to make decisions for, although they would be aware of the actions of other entities and the consequences of these. This would be an ambitious extension as it would require using the JS-EDEN tool in a distributed way where multiple users work on the same environment using separate browsers.

When developing example models to implement within the prototype ADM plugin it was realised that a lot of applications have an autonomous component as well as components that require a user to make decisions for their actions. An example of this is the Cat Flap example, where it would make sense for deterministic entities such as the cat flap to act autonomously. The current ADM plugin could be extended such that when creating or instantiating templates it is possible to define whether these will be autonomous or human controlled. At each step autonomous guards will also be evaluated and any that evaluate to true automatically processed alongside any actions selected by the modeller. Any conflicts that arise due to autonomous actions would have to be resolved by the human user. However it is useful experimentally for these autonomous changes to still be visible to the user, so possibly these could just be autoselected within the human perspective. This would also retain the possibility of the user modelling the failure of one of these autonomous components.

6 Conclusion

Creating an ADM prototype for JS-EDEN has asked many design questions and provides the possibility of a wide range of models, both with deterministic and nondeterministic entities. It is hoped that the prototype can be further developed in future to improve the plugin and introduce more modellers to the ADM. Lessons that have been learnt when developing this plugin can be used to inform the design of any future work and have brought to light a great number of possible extensions.

The creation of this prototype has made it possible to experiment with a human perspective of a parallel multi-agent system and hopefully models will be created to take advantage of this. The creation of these models is likely to reveal further work and ask further questions about the design of the ADM, allowing it to be developed iteratively through experience in a similar way to EM models themselves. The creation of and interaction with these models will hopefully give the human user a deeper empirical understanding of the system being modelled and the ways in which multiple entities interact and the consequences of this on system state. It also allows experimentation with the consequences of the choice of actions of nondeterministic agents, as was not previously possible.

Acknowledgements

I would like to thank Meurig Beynon for his support and guidance throughout the development of the project, as well as for all his enthusiasm and suggestions.

References

WM Beynon. Parallelism in a definitive programming framework. *Proc Parallel Computing*, 89, 1990.

WM Beynon. Agent-oriented modelling and the explanation of behaviour. In *Proc. International Workshop Shape Modelling Parallelism, Interactivity and Applications*, 1994.

WM Beynon, MT Norris, and MD Slade. Parallel computation in definitive models. 1988.

WM Beynon, MT Norris, RA Orr, and MD Slade. Definitive specification of concurrent systems. In *UK IT 1990 Conference*, pages 52–57. IET, 1990.

M Slade. *Definitive Parallel Programming*. PhD thesis, MSc Thesis, Department of Computer Science, University of Warwick, 1990.