FLAME
Developer Manual

Simon Coakley

Unit - USFD

October 10, 2009

Abstract

The report presents a developer manual for the FLAME framework. The manual describes specifically the management of agent memory, agent execution, and agent communication.

Contents

1 Introduction

The FLAME framework is an enabling tool to create agent-based models that can be run on high performance computers (HPCs). Models are created based upon extended finite state machines that include message inputs and outputs. This information is used by the framework to automatically generate a simulation program that can run models efficiently on HPCs.

2 Memory

Allocation of memory for agents and messages should be in one continuous area of memory. This is so that the command *sizeof* can return the byte size of agent and message memory in one execution. This is important when sending agents and messages in parallel using MPI. The main problem is the use of dynamic arrays because the associated memory allocated is not going to be apart of the continuous memory allocated for agents and messages. It is possible to handle this but it is expensive. Because messages are continuously being sent and received they should not contain any dynamic arrays. It is only when agents are being load balanced that they need to be sent in parallel.

Currently user-defined data types are allocated as pointers in agent memory but this has been modified in a new version to be released. This means that instead of user using an arrow '->' to dereference variables, a dot '.' is used to access the data structure.

Dynamic array data structures are also not allocated as a pointer (but the actual dynamic array is) which means functions to interact with a dynamic array data structure need to pass a pointer. This means the use of the ampersand '&' to reference the data structure.

2.1 Agent Memory Management

Each agent has an associated memory data structure. Since early versions of the framework all agents have been managed in one list. This was so that the list could be randomised and therefore remove any artifacts of agents having priority over other agents by always being executed first. In essence the only effect this has is to randomise the message output and therefore the message input to agents. The current framework has a generic agent memory stucture that can point to any specific agent type.

With the introduction of the new message board library the action of randomising (or now also sorting and filtering) messages the need to randomise the agent list is redundant. Also redundant is the need to have a single list of all the agents. The generic agent memory structure is therefore not needed and each agent type can have it's only seperate list.

3 Execution

Agents have associated functions. The order that these functions are run are currently defined by dependencies on other functions. If they are within the same agent they are internal and if they are between agents, i.e. messages, they are communication dependencies. All functions of all agents are executed in this prescribed order.

But there are many instances when functions should not be run. By returning to the original basis of defining agents as extended finite state machines or X-Machines, this can be incorporated. Instead of defining function order using dependencies, functions are transitions between states. The order of functions is the transitions between states, from the start state(s) to the end state(s). There can be many paths through these transitions which are governed by conditions of their traversal.

Each transition function should be defined using:

- current state: the current state of the agent
- input: the inputs the function is expecting
- m_{pre}: the conditions on memory of executing the function
- name: the name of the function
- m_{post}: the changes in the memory (i.e. the function code)
- output: possible outputs of the function
- next state: the next state to move the agent to

By graphing all the possible transition routes of agents from the start state(s) to end state(s) the order of function execution can be created. This also provides a way to manage the processing of agents. By providing a link to an agent list for each possible agent state, agents can be moved between these agent state lists until they reach an end state.

NEW: check to see if agents being transitioned and not left behind in an old state

to start a new iteration, agents are moved from the end states into the start states.

4 Communication

Handling of communication.