Managing Resources in Constrained Environments with Autonomous Agents

C. Muldoon¹, G. M. P. O’Hare², M. J. O’Grady²

¹ Practice and Research in Intelligent Systems and Media (PRISM) Laboratory, School of Computer Science and Informatics, University College Dublin (UCD), Belfield, Dublin 4, Ireland.
² Adaptive Information Cluster (AIC), School of Computer Science and Informatics, University College Dublin (UCD), Belfield, Dublin 4, Ireland.
  conor.muldoon@ucd.ie
  {gregory.ohare, michael.j.ogrady}@ucd.ie

Abstract. In the future electronic devices will permeate the environment where they will work invisibly and autonomously to deliver new and enhanced services that go far beyond the mandate of the desktop era. Intelligent agents will form the basis of many applications in this emergent ubiquitous domain. Agent Factory Micro Edition (AFME) is a framework that facilitates the construction of agent-based applications for computationally constrained devices, this paper outlines three enhancements introduced to AFME to enable resources to be managed more effectively, namely a new threading model, an extended rational decision making infrastructure, and a syntactic modification to the agent programming language that improves efficiency. The extended reasoning capabilities of AFME enable agents to choose the most appropriate course of action with respect to their finite resources in a social context.

1 Introduction

The true potential for information technology is in making it an integral, invisible part of the way people live their lives [1]. Ubiquitous computing prescribes a new model of computation, one that takes into consideration the natural human environment and removes computers from our direct focus into the objects that surround us. In the ubiquitous computing era intelligent agents will operate on ‘smart’ devices where they will manage and control dynamic ad-hoc networks, working both reactively and pro-actively to achieve individual and common goals. This paper concerns three features introduced to Agent Factory Micro Edition (AFME) [2], a framework for the construction of agents that operate on resource constrained mobile devices.

A new threading model, embedded within AFME, ensures that agents are sensitive to absolute time. This enables agent response time values to be specified accurately. Rather than each agent creating their own thread agents share

³ When the application begins to execute a clock value is recorded in the scheduler. Subsequent timing values are relative to the initial clock value.
a thread pool whereby they are scheduled to execute at regular intervals. To reduce the number of computational bottlenecks, and prevent agents from synchronising with one another, response time values are altered such that they are prime numbers.

To be able to solve problems collectively social structures must be present to enable agents to interact with each other. For agents to act as a team more is required than the synchronisation of individual isolated events. The group must act as a single agent that adopts beliefs, desires, and intentions of its own [3]. The manner by which collective decisions are made, however, is ultimately governed by the choices made by, the desires of, and the goals of the individuals that form the group. The extended rational decision making capabilities introduced to AFME enable agents to choose the most appropriate course of action with respect to their finite resources.

Belief labeling has been introduced to AFME to improve the efficiency of the reasoning process and to reduce development time. With belief labeling common sequences of predicates are only encoded and evaluated once.

The paper is organised as follows. Section 2 provides a broad overview of AFME. Section 3 describes the threading model. Section 4 describes the rational decision making infrastructure. Section 5 discusses belief labeling. An evaluation is provided in section 6. Some related work and a discussion that focuses on the social aspects of the system is provided in section 7.

2 AFME

AFME is loosely based on Agent Factory [4], it uses a subset of the Agent Factory Agent Programming Language (AFAPL) [5] and augments it with a number of features specific to AFME. AFAPL is founded on a logical formalism of belief and commitment. Rules that define the conditions under which agents should adopt commitments are used to govern and encode agent behaviour. AFME is described elsewhere [6] [2]. This section provides a broad overview of AFME to put the work in context but the primary focus of this paper is on the three features introduced to AFME to enable resources to be managed more effectively.

An AFME platform comprises a scheduler, several platform services, and a group of agents (see figure 1). The scheduler is responsible for the scheduling of agents to execute at periodic intervals. Rather than each agent creating a new thread when they begin operating, agents share a thread pool.

AFME delivers support for the creation of BDI agents that follow a sense-deliberate-act cycle. The control algorithm performs four functions. (1) Preceptors are fired and beliefs are resolved within the belief resolution function. (2) The beliefs are used within the deliberation process to identify the agent’s desired states. Agents are resource bounded and will be unable to achieve all of their desires even if their desires are consistent. (3) A subset is chosen, within the intention selection process, that maximises their self-interest with respect to their finite resources. (4) The final function of the control algorithm con-
cerns commitment management. Depending on the nature of the commitments adopted various actuators are fired.

AFME has been used in several applications [7] [8]. A discussion of these applications is beyond the scope of the paper. A key requirement for the development of an agent platform for constrained environments is that the platform shall manage resources efficiently and effectively. This is primary focus of this paper.

3 Threading Model

3.1 Thread Management

Agents in AFME are scheduled to execute at periodic intervals. In the original Agent Factory threading model when an agent was constructed its controller created a new thread and the agent began executing. A sleep time parameter was passed to the controller and was used to determine the responsiveness of the agent and to facilitate cooperative multi-threading. The original system was somewhat limited in that all of the threads operated in an independent ad-hoc manner; there was no management or scheduling of the threads involved. Consider the use of the sleep time to indicate the responsiveness of the agent. The original system did not support the concept of fixed-rate execution, whereby subsequent executions take place at approximately regular intervals that are sensitive to absolute time. In the AFME threading model the agent’s true response time is determined by a combination of the time taken to perform background activities (such as garbage collection), the agent’s execution time, and the agents’ sleep time.

The sleep time parameter specified the amount of time the agent’s thread would sleep in between iterations of the control algorithm.
The objective of the new threading model is to introduce thread management and scheduling into AFME. The system is composed of a scheduler, a thread pool, and a task buffer. The threading model can be used to schedule any task but in AFME the task is typically an agent. The threading model is used in the collaborative agent timing framework [9]. Figure 2 illustrates the architecture of the new threading model. Rather than starting a new thread when an agent is created, the agent’s control algorithm is incorporated into a thread task object that is added to the scheduler. When the task is added the desired response time is specified. The task is then scheduled to be performed at approximately regular intervals with respect to absolute time. The amount of sleep time is varied so that the response time will be consistent. The accuracy of the timing is dependent on the clock underlying the Object.wait() method. The scheduler synchronises all response times with an initial clock value that is recorded when the application begins to operate. Task sleep times are constantly adjusted to be in synchronisation with the universal clock value so that an agent’s response time value will be regular and consistent with respect to (1) its previous execution times and (2) the relative execution times of the other agents.

The threading model is not limited to the execution of agents. Other internal platform or application tasks, such as IO operations, are scheduled to be performed at some point in the future either on a once off basis or periodically. These tasks are subject to the same timing criteria and scheduling algorithm. An example of where this is used is within the message transport service, which is required to periodically connect to an external server to receive incoming messages [2]. This is possible because the management logic of the thread pool is decoupled from the functionality that the thread is executing. A thread is simply viewed as a process. The management logic is not concerned about the specific task that the thread is performing.

The Scheduler contains an internal binary search tree to schedule tasks efficiently. The tree ensures that tasks are ordered in accordance to their scheduled execution time. When a task is added to the tree the scheduler thread is notified. The minimum node is obtained from the tree and its execution time is calculated. The scheduler thread waits until the execution time of the minimum node and then places it in the task buffer. If the node requires periodic execution it is rescheduled otherwise it is removed.

When placing tasks into the task buffer the scheduler first checks an active set of tasks. If the task is not in the active set it is added. The task buffer is composed of a set of tasks to be executed within one of the threads in the thread pool. The pooled threads extract and execute tasks from the task buffer. Once a task has completed execution it is removed from the buffer.

---

5 It is possible to obtain a universal clock value as the threading model is only concerned with agents on the local platform. Agents on other platforms will be using different CPUs and therefore their computational load will not overlap with local agents.
3.2 Primal Scheduling and Phase Shifting

To prevent the agents from synchronising with one another the agents’ response times are altered to be prime numbers. For example consider two agents, one with a response time of 500 milliseconds, and the other with a response time of 1000 milliseconds. Both agents begin to execute at time 0 and finish executing at some later point. At time 500 the first agent begins to operate again and then finishes at some later point. At time 1000 because both agents are sensitive to absolute time they will both begin to execute at the same point because any variance incurred due to background tasks or executing times will have been removed. The cycle will repeat itself and both agents will begin executing at the same time point at 2000, 3000, 4000, . . . milliseconds. The harmonics of the agents’ response times cause the agents to synchronise with one another, the agents are literally in tune. This causes a computational bottleneck. It is undesirable, from a computational efficiency perspective, to have two or more agents beginning to execute at exactly the same time. The time between the last agent finishing to execute and the next agent or the next two agents beginning to execute is effectively wasted.

To prevent this problem agents’ response times are altered to be prime numbers. Consider the case when the first agent’s response time is changed to 499 and the second agent’s response time is changed 1013. Rather than the first agent synchronizing with the second agent once every two iterations and the
second agent synchronizing with the first agent on every iteration the first agent will synchronize with the second agent once every 1013 iterations whereas the second agent will synchronize with the first agent once every 499 iterations. The worst case scenario, whereby both agents begin executing at exactly the same time, only occurs once every 505487 (499*1013) milliseconds rather than once every 1000 milliseconds. The system ensures that the agents are out of tune with respect to each other.

The effect that this has is to average the agents’ computational load over the available time range. The agents’ computational overhead will still sometimes overlap but the time between agents completing execution and subsequently beginning to execute will not always be wasted. It should be noted that 1013 was chosen rather than a prime value, such as 997, closer to the original because it would cause the agents to come out of synchronization faster since there is a greater difference between the harmonics of 1013 and 499 than that of 997 and 499. The primes chosen are those that have the greatest harmonic difference within a particular threshold value. The threshold value is intended to keep the values close to the originals specified. If there are no primes within this range then the primes closest to the originals are chosen.

To further reduce computational bottlenecks the threading model phase shifts thread tasks with equal response times. Consider two agents with response times of 499. Rather than altering their responsiveness to be different primes such 491 and 509 the system phase-shifts one of the agents by 180 degrees. So the first agent will begin executing at time 0 whereas the second agent begins executing at time 249. Because the agents are sensitive to absolute time and have equal responsiveness values they will never begin executing at exactly the same point, there will always be approximately 249 milliseconds between their execution times. The number of degrees that the agents are phase shifted is equal to 360 divided by the number of agents with equal responsiveness multiplied by the agent’s arbitrary ordinal number. Thus in the previous example the first agent is phase shifted by 249 * 0, i.e. it is not phased shifted, whereas the second agent is phase shifted by 249 * 1. The choice as to which agent is first, second, third... is capricious.

Another improvement to the efficiency of the threading model is that it randomizes start times. When applications begin to operate rather than having all tasks or agents begin to execute at time 0 the system staggers agents’ and other scheduled tasks’ start times.

4 Rational Decision Making

The requirement for agents to be rational necessitates that they act in a manner that maximises their self-interest or utility. The term self-interest was first popularised by Adam Smith [10] during the enlightenment period. In AFME Agents are rational. Their self-interest enables them to act in a consistent manner to achieve their objectives. For some agents their objectives necessitate selfish behaviour for others they do not.
AFME is consistent with the BDI model of agency [11]. The BDI model acknowledges that agents are resource bounded and thus will be unable to achieve all of their desires even if their desires are consistent. The agent must fix upon a subset of desires and commit resources to achieving them. This subset of desires is the agent’s intentions. To be able to fix upon a subset of desires the agent must make a decision over a number of options. There must be some concept of utility or preference for an agent to make a decision 6. Otherwise the agent would be unable to choose between the various alternatives. Even a random or arbitrary choice requires the concept of utility in that all utility values must be equal, thus no preference is specified. A random choice is rational in such circumstances because it returns the maximum possible benefit from the various alternatives.

This illustrates the difference between utility and preference. Utility specifies the benefit of an option whereas preference specifies a greater than relation between utilities. To say that the taste of a cake is preferred to the taste of some other cake does not imply that they both taste equally as nice. It implies that one tastes better than the other. If you prefer something, you believe it to be better than something else. This differs from weak stipulative definitions of preference [12] whereby two items can be equally preferred to each other. In the context of this work if the utility values of two items are equal, no preference is specified between those two items. Stipulative definitions cannot be considered correct or incorrect because they are not propositions. A definition can only be considered correct or incorrect when discussing usage, in such a case the definition is lexical.

The intention selection process has been realised within AFME by extending the Agent Factory Agent Programming Language (AFAPL) with additional constructs for rational decision making. In AFME the subset of commitments chosen maximises the agent’s utility for a particular course of action. In AFAPL rules that define the conditions under which agents adopt commitments are used to govern an agent’s behaviour. These rules are based on the agent’s current model of the world, namely the agent’s beliefs. In AFME’s extended version of AFAPL the decision as to whether to adopt a commitment is contingent on the amount of resources an agent has available to it and the previous commitments that it has made. For example if an agent has a significant number of commitments it will decide not to adopt an additional commitment that it otherwise would have adopted if it did not already have such a heavy workload. If the agent does not have the requisite resources available the commitment is not adopted. If the agent can free additional resources by dropping previous commitments and the benefit of adopting the new commitment is greater than the loss of dropping the previous commitments then the agent drops the older commitments to make the requisite resources available. At a given point in time an agent will have, based on its model of the world, a number of commitments that it wishes to adopt. The agent chooses to espouse the subset of commitments that maximises its utility with respect to its finite resources. If an agent fails to achieve a commitment

---

6 The utility is usually determined in the interpreter and is not specified within the logic components.
they must still incur the costs of the attempt. This is true for both human and computational agents.

In BDI logics the concept of desire is a qualitative representation of utility. The intentions are chosen, within the interpreter, from among the desired states, using some metric that represents the actual utility value. In the AFAPL extension the metric for determining the utility is removed from the intention selection algorithm and is replaced within perceptors that generate beliefs about the costs and benefits of certain actions 7. This is useful because it enables different metrics to be used for different commitments. The values generated by the perceptors are only potential utility. The chosen commitments must still be from among the desired states as determined by the commitment rules within the agent design. If a commitment is desired then the potential utility value will represent its actual utility value within the intention selection algorithm otherwise the commitment is not considered for selection. The desires are still a qualitative representation of utility.

Decoupling the utility metric from the intention selection algorithm has other advantages. It enables the developer to more easily alter system behaviour within the agent design. The benefit and cost of certain actions will be dependent on context. Variable beliefs provide a natural way of representing such data. The beliefs are not exclusive to the intention selection process and will sometimes be used by the agent to drive other system behaviour.

In the AFAPL extension the total amount of resources available to the agent is specified so that the agent is aware of its limitations or constraints. The following is an example of an agent design written with the addition constructs:

```
resources: ?res;

BELIEF(a) & BELIEF(costX(?cx)) & BELIEF(benefitX(?bx))
=> COMMIT(Self, Now, BELIEF(true), doX, ?cs, ?bx);

BELIEF(b) & BELIEF(costY(?cy)) & BELIEF(benefitY(?by))
=> COMMIT(Self, Now, BELIEF(true), doY, ?cy, ?by);

BELIEF(c) & BELIEF(costZ(?cz)) & BELIEF(benefitZ(?bz))
=> COMMIT(Self, Now, BELIEF(true), doZ, ?cz, ?bz);
```

In AFAPL terms preceded by the question mark symbol are variables. For illustrative purposes consider the case where the variables ?cx, ?bx, ?cy, ?by, ?cz and ?cz assume the values 30, 11, 25, 5, 10, and 2 respectively. These variables represent the cost and benefits of adopting the commitments. Assume the resources variable, ?res, has a value of 20. If at a given point in time the agent adopts either BELIEF(a), BELIEF(b), or BELIEF(c), along with the beliefs for costs

---

7 The utility values do not necessarily have to be determined by a perceptor. They may be hard coded by the developer in the agent design but this will not usually be the case because it will often be difficult to determine utility values a priori.
and benefits, the commitments for doX, doY, doZ will be adopted respectively. The agent compares the commitment’s cost to the available resources and if the cost is lower the commitment is adopted. If the agent adopts all three beliefs at the same discrete time point all three commitments are adopted because their combined cost is 18 which is less than the available resource allocation. Commitment resolution is non-deterministic, it does not make any difference what order the commitment rules are specified. The subset of commitments that maximise utility with respect to available resources at a particular time point will be chosen to be adopted.

Consider the case when all three beliefs are adopted but when the resource constraint is set at 15 rather than 20. The agent wishes to adopt all three commitments but does not have the resources to do this and must therefore make a decision as to what commitments to espose. Considering the available resources the agent’s options are (1) doX and doZ, or (2) doY and doZ. The agent chooses doX and doZ because it yields a greater utility.

The costs and benefits of temporal commitments will sometimes vary over time. If this is the case agents adopt beliefs about the cost and benefits of maintaining the commitments. Variables within the maintenance condition of a commitment are matched with the variables for cost and benefit.

The resource constraints specified are an abstract representation of resources rather than a direct reference to the CPU’s computational overhead. There are a number of reasons why this approach has been adopted. The agent does not need to be aware of its low level processing performance to reason about its potential actions in much the same way that a person does not need to be aware of the exact number of joules they are going to use to perform a particular task. People have a more abstract notion of the amount of work involved and use this abstract concept, along with other factors such as their spatial and temporal constraints, previously adopted commitments etc., within their decision making process. The abstraction effectively hides such low level data. The tasks typically performed by intelligent agents are at a higher level of granularity than CPU scheduling. An agent’s current load will only be required when an agent is making a decision as to whether or not to perform a future action. Processor performance is not useful in determining the overhead of future events because it can only be measured after the event has occurred. An agent’s abstract representation of the cost of its commitments would be more useful in such situations. A developer could profile a particular application and use the data as a pre-emptive indication of a task’s overhead. The data must still only be used as an approximation to overhead and be specified abstractly because it is dependent on the platform on which profiling was carried out. The abstract representation of resource usage facilitates the development of generic agents whose functionality is not coupled or dependent on particular processor capabilities. Java is a platform independent language and hides processor specific information from the developer. J2ME does not support the Java native interface so platform dependent code cannot be written to obtain such data in any case.
The metric for determining the amount of resources available could of course include the residual power of an embedded device along with its computational constraints. In such a case the power of the device would have an effect on the nature and degree of the commitments adopted.

The original AFAPL is a particular instance of the extended language specification whereby the benefit of adopting commitments is 1, the cost of adopting commitments is 0, and the resource allocation is non-negative. In this case the agent will adopt all of its potential commitments because each commitment will increase its utility for free. In the extended version of the language if the values for a commitment’s benefit and cost are not specified their default values are 1 and 0 respectively. If the resource allocation is not specified it has a default of 0. In this respect, it is consistent with the original specification. An agent written in the standard AFAPL will act in the same manner within an AFME environment as it does in the standard framework.

In reality when someone is considering adopting a non-trivial commitment their beliefs about the costs and benefits of the commitment along with an abstract concept of the amount of resources available to them form an integral part of their reasoning process. Over time their beliefs about the costs and the benefits of adopting certain commitments change because they are dependent on context.

5 Belief labeling

Belief labeling has been introduced to improve the efficiency of the reasoning algorithm and reduce development time. It is syntactically different from AFAPL but the underlying semantics are equivalent. The following commitment rules

\[
\text{BELIEF}(x) \& \text{BELIEF}(y)\text{ is somelabel;}
\]
\[
\text{BELIEF}(w) \Rightarrow \text{COMMIT}(\ldots)\text{ requires somelabel;}
\]
\[
\text{BELIEF}(z) \Rightarrow \text{COMMIT}(\ldots)\text{ requires somelabel;}
\]

are an equivalent alternative to

\[
\text{BELIEF}(x) \& \text{BELIEF}(y) \& \text{BELIEF}(w) \Rightarrow \text{COMMIT}(\ldots);
\]
\[
\text{BELIEF}(x) \& \text{BELIEF}(y) \& \text{BELIEF}(z) \Rightarrow \text{COMMIT}(\ldots);
\]

similarly

\[
\text{BELIEF}(z) \Rightarrow \text{COMMIT}(\ldots)\text{ requires !somelabel;}
\]

is equivalent to

\[
!\text{BELIEF}(x) \& \text{BELIEF}(y) \& \text{BELIEF}(z) \Rightarrow \text{COMMIT}(\ldots);
\]
\[
\text{BELIEF}(x) \& !\text{BELIEF}(y) \& \text{BELIEF}(z) \Rightarrow \text{COMMIT}(\ldots);
\]
\[
!\text{BELIEF}(x) \& !\text{BELIEF}(y) \& \text{BELIEF}(z) \Rightarrow \text{COMMIT}(\ldots);
\]

The second case is not logically equivalent to the single commitment rule

\[
\text{BELIEF}(z) \Rightarrow \text{COMMIT}(\ldots);
\]
because in that case if both $\text{BELIEF}(x)$ and $\text{BELIEF}(y)$ were adopted the commitment would still be espoused, this is not what we want. In this example the agent will only ever adopt one commitment because each rule is mutually exclusive. Its clear, in this case, that considerably less reasoning is required than the original AFAPL approach. Additionally, the developer need only write one line of code rather than three (provided some label is already declared). This improves efficiency as the belief sequence is only evaluated once. The behaviour encoded is a negated logical and.

Agent designs that contain belief labels may be compiled into a format that is equivalent to the original AFAPL syntax so that the agent can run on the standard platform. When an agent migrates from an AFME environment it is converted to the alternative format.

The developer need not worry about writing the optimal agent design for performance. The compiler converts the code to the optimum automatically. Nevertheless, the developer can save time by doing so and improve the maintainability of their code by minimising redundancy and increasing reuse. Standard AFAPL design files can be compiled into the belief label format using the AFME compiler.

Belief labels can be embedded to further reduce redundancy. The some label belief set is embedded in the other label belief set in the following example.

$\text{BELIEF}(x) \land \text{BELIEF}(y)$ is some label;
$\text{BELIEF}(w) \land \text{BELIEF}(z)$ is other label requires some label;

It should be noted that cycles are prohibited. If the developer encodes cyclical dependencies the compiler will throw an error.

6 Evaluation

6.1 Evaluation of Threading Model

To evaluate the effectiveness of the threading model we conducted four experiments. In the first two experiments the unmanaged approach\(^8\), the scheduled (timed) approach, the phase shifting approach, and the random start time approach were compared. Agents with equal average response time values\(^9\) and random computational overheads were used. The second two experiments evaluated primal scheduling.

The agents summed the integers from 0 up to a pseudo random number chosen between 0 and 100000. Arithmetic series rules were not used. The same seed was used to generate the pseudo random numbers to ensure that the results were consistent in each case. In the new threading model agents share a thread pool rather creating their own thread. The number of threads in the pool is three. The results of the computational overhead experiment are illustrated in figure 3.

---

\(^8\) The unmanaged (original) approach refers to when scheduling is not used.

\(^9\) The term response time cannot be used for agents using the unmanaged approach.

In that case we are referring to the sleep time of the agent.
Fig. 3. Average time taken to process tasks with random computational overheads.

The scheduled (timed) approach is the least efficient. This is as we would expect in that by ensuring that agents' response time values are consistent we ensure that agents are in tune with respect to each other. The random start time approach is sometimes better than the unmanaged approach other times it is not. The phase shifting approach is the most efficient. This is because it ensures that the agents never begin operating at the same time.

The experiment was repeated but this time the responsiveness of the agents was recorded rather than their computational overhead. The results of the responsiveness experiment are illustrated in figure 4.

Fig. 4. Average processing duration for sixteen iterations for tasks with random overheads.

The results indicate that Phase shifting is the most desirable. The problem is that phase shifting can only be used when the agents' response time values are equal. To improve the efficiency of agents that have different response times primal scheduling was developed.

To evaluate primal scheduling we conducted an experiment using four homogeneous agents with response times of 1000, 500, 250, and 125 milliseconds. Each
of the agents has an equal computational overhead. The agents sum the integers from 0 to 100000. Again, arithmetic series rules were not used. The experiment was conducted with and without random start time values. The results for the computational processing time and responsiveness are given in table 1. These results illustrate that randomising the start times improves efficiency with the standard scheduling approach.

<table>
<thead>
<tr>
<th>Random Time</th>
<th>Random Duration</th>
<th>Time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>125</td>
<td>2064</td>
<td>76.77</td>
<td>2639</td>
</tr>
<tr>
<td>250</td>
<td>3900</td>
<td>106</td>
<td>3866</td>
</tr>
<tr>
<td>500</td>
<td>7828</td>
<td>119.44</td>
<td>7619</td>
</tr>
<tr>
<td>1000</td>
<td>1558</td>
<td>115.33</td>
<td>15120</td>
</tr>
<tr>
<td>Average</td>
<td>71.33</td>
<td>104.39</td>
<td>7324</td>
</tr>
</tbody>
</table>

**Table 1. Standard Scheduling Results**

The experiment was repeated but with the response times altered, by the scheduler, to be 991, 509, 241, and 131. The results are given in table 2.

<table>
<thead>
<tr>
<th>Random Time</th>
<th>Random Duration</th>
<th>Time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>131</td>
<td>2106</td>
<td>60.44</td>
<td>2222</td>
</tr>
<tr>
<td>241</td>
<td>3804</td>
<td>60.55</td>
<td>3806</td>
</tr>
<tr>
<td>509</td>
<td>7933</td>
<td>69.77</td>
<td>7851</td>
</tr>
<tr>
<td>991</td>
<td>15338</td>
<td>71.77</td>
<td>15068</td>
</tr>
<tr>
<td>Average</td>
<td>65.61</td>
<td>7323</td>
<td>65.64</td>
</tr>
</tbody>
</table>

**Table 2. Primal Scheduling Results**

To enable the developer to specify the responsiveness of an agent accurately a scheduling procedure must be adopted. The accurate scheduling of agents causes problems because agents with harmonically similar response times become synchronized. These experiments indicate that by altering an agent’s response time value to be a prime number the efficiency of the platform is improved.

### 6.2 Algorithm Analysis of the Rational Decision Making Extension

The task of determining the subset of commitments to espouse is a classic 0-1 knapsack problem. Given n items, with corresponding values and weights, the knapsack problem concerns the packing of some of these items in a knapsack of a specified capacity C, such that the profit sum of the included items is maximised. This is equivalent to the problem of an agent attempting to adopt the subset of commitments that maximise its utility with respect to its finite resources. It is called the 0-1 version of the knapsack problem because there are no fractions.
The whole item must be either included in the knapsack or left out. Similarly, an agent cannot adopt half a commitment.

Since the pioneering work of Dantzig [13] the knapsack problem has been studied extensively in practice as well as in theory. No polynomial-time solution is known for the general case; it is non-deterministic polynomial-time hard [14]. The solution adopted in AFME uses a standard dynamic programming approach [15] and operates in pseudo-polynomial time. It has a run-time complexity of $O(nC)$. The complexity of dynamic programming solutions assure a much faster running time than other techniques, such as backtracking or brute-force. As noted in [14], “A pseudo-polynomial-time algorithm... will display ‘exponential behavior’ only when confronted with instances containing ‘exponentially large’ numbers, [which] might be rare for the application we are interested in.” It is not anticipated that ‘exponentially large’ numbers will be encountered in AFME, nevertheless, future work will investigate the use of a greedy approximation algorithm.

6.3 Efficiency of Belief Labeling

Belief labeling reduces redundant processing. It enables developers to encode common sub sequences of predicates, which are only evaluated once. A depth first search is still used to match variables but the search space is considerably reduced. A depth first search has a runtime complexity of $O(b^m)$, where $b$ is the branching factor\(^{10}\) and $m$ is the maximum depth. To evaluate the efficiency of belief labeling we consider the worst case, in which the developer encodes a negated logical and. A negated logical and specifies the conditions under which a commitment will not be desired. Under all other circumstances it will be desired. To model this type of situation with the original approach the system would exhibit exponential behaviour. The developer would have to write, and the system would have to process, $2^n-1$ rules for $n$ conditions. With belief labeling the developer need only write, and the system need only process, 1 rule.

7 Discussion and Related Research

7.1 Agent Platforms

There have been several agent platforms developed for resource constrained environments reported in the literature. The LEAP [16] (Light Extensible Agent Platform) is a FIFA compliant agent platform capable of operating on both fixed and mobile devices. LEAP extends the JADE (Java Agent DEvelopment) architecture by using a set of profiles that allow it to be configured for various Java Virtual Machines (JVMs). The platforms supported are J2SE, Personal Java, and CLDC/MIDP. The architecture is modular and contains components for managing the life cycle of the agents and controlling the heterogeneity of communication protocols. The LEAP add on when combined with JADE replaces certain components of the standard JADE runtime environment to form a modified kernel that is referred to as JADE-LEAP or JADE powered by LEAP.

\(^{10}\) The branching factor is the average number of children.
3APL-M [17] provides a platform that enables the fabrication of agents using the 3APL language for internal knowledge representation. It provides a scaled down version of the pre-existing language infrastructure, which was designed for a desktop environment. The 3APL-M architecture contains sensor and actuator modules, the 3APL machinery, and the communicator module. The sensor and actuator modules enable the agents to sense and to act upon their environment respectively. The 3APL machinery is a BDI reasoning engine. The communicator module provides the support for inter-agent communication. mProlog was developed as a subcomponent of the 3APL-M project. It is a reduced footprint Java Prolog engine, optimized for J2ME applications.

Agilla [18] is an agent platform that has been designed specifically for wireless sensor networks where power consumption is an issue. Agilla agents are modelled as genetic algorithms and are not reflective.

MicroFIPA-OS is a minimised footprint version of the FIPA-OS agent toolkit [19]. The FIPA-OS was developed as an agent middleware environment to enable the creation of FIPA compliant agents. MicroFIPA-OS was constructed because the original FIPA-OS employed software engineering techniques, such as excessive object creation and XML parsing, that did not scale down well. The MicroFIPA-OS improves the efficiency of the system by avoiding or removing some of the additional overhead, such as mandatory XML parsing. It manages resources better and introduces thread and other resource pools that are shared among agents.

Though sharing the same broad objectives of these projects, AFME differs in a number of key ways, JADE-LEAP, and MicroFIPA-OS are frameworks for the development of agent technology but they do not contain reasoning capabilities. Any intelligence required must be written by the application developer. These systems therefore do not adhere to the same definition of agency as AFME. It is sometimes claimed that intelligent frameworks that have been developed for JADE will work with JADE-LEAP without making alterations to the code but this is in fact not the case. This would only work for the J2SE and perhaps Personal Java versions of JADE-LEAP. If an application were developed for JADE without considering the possibility of porting it to a CLDC/MIDP environment it would contain dependencies on standard Java classes and APIs not present or supported in the CLDC/MIDP specification.

3APL-M provides support for the construction of cognitive rational agents but it is an API not an agent development framework as such. This differs from AFME whereby the agent functionality is specified in a platform independent AFAPL design file and the Java code generated from the design through the use of the AFME compiler. The AFME development process supports the development of agents written in AFAPL through the use of visual debugging tools, a development methodology, and an integrated development environment.

The threading infrastructure differs from other systems in that agents are phase shifted and their response times are altered to be prime numbers so as to prevent computational bottlenecks.
The AFME migration process distinguishes itself from other approaches in that it hides platform idiosyncrasies. Generic agent designs are combined with platform specific functionality enabling agents to move between heterogeneous environments. This process is handled autonomically and is transparent from the agent’s perspective.

7.2 Social Structure

Following on from section 4, for an action to be performed jointly more is required than just the union of simultaneous individual coordinated events. When a group decides to collaborate a team is formed that acts as a single agent with beliefs, desires, and intentions of its own above those of the individual team mates. [3] gives a formal model of the mental properties of teams and how joint intentions act, are affected by, and are reduced to the mental states of the team members.

In AFME agents are rational. Sometimes they collaborate other times they compete depending on the context. The decisions agents make are governed by their self interest11. If the developer wishes to guarantee that agents will always collaborate they encode utility values or use metrics such that it is always in an agent’s interest to collaborate. Individually rational altruistic agents adopt common goals and help other agents at their own expense because they believe there is value in doing so even though it might never improve their personal welfare. The utility values adopted do not necessarily represent the personal welfare of the agent. In AFME some agents are selfish. Selfish agents benefit themselves but they also benefit society because to gain profit off one’s own labours in an open market something must be provided that others value [10].

The manner by which collective decisions are made is ultimately governed by the choices made by, the desires of, and the goals of the individual agents that form the team. This is not to say that responsibility should end with the individual agent, but that it must start with the individual agent. Individuals will choose a course of action that maximises their self-interest. To illustrate the influence of variable utility values on the collective behaviour of agents consider a wireless sensor network with a number of interconnected nodes. Transmission in a wireless sensor network is quite a costly operation, for example the transmission of a single bit is equivalent to the execution of 1000 instructions with regards to power consumption on a typical node [20]. As the power level of the device decreases the utility of performing collective actions will decrease whereas the utility of performing an operation locally will increase. Different perceptrons are required for generating the requisite utility values of collective and local actions. When the device’s residual power is high the agent will communicate more often. When the power is low the agent will still sometimes communicate but not as much. The relationship between individual and collective action is not binary; it depends both on variable context values and the agent’s mental state. In this case the power of the device has an effect on the nature and degree of the commitments adopted.

11 It should be noted that term self-interest is not synonymous with selfishness.
The behaviour of agents with similar goals is not the same as agents with common goals. For example consider two agents called Alice and Bob who both have a goal of getting to a particular football match. Alice and Bob have similar but not common goals. Alice’s goal is for Alice to get to the football match, Bob’s goal is for Bob to get to the football match. For Alice and Bob to have a common goal Alice would have to have a goal for Alice to get to the football match and Bob would also have to have a goal for Alice to get to the football match. In such a circumstance Alice and Bob would collaborate to ensure that Alice gets to the match. Sometimes developers will want agents to continue to collaborate even when resources are scarce. As noted earlier, in AFME this type of functionality is modeled through the use of fixed utility values that ensure that it is in an agent’s interest to maintain common goals. If common goals are dropped agents will begin to compete.

To illustrate this point we shall consider the case when there are a number of agents operating on a node in a wireless sensor network. The residual power of the device is dropping. If all of the agents stay on the device there will not be enough resources to support them. In this scenario the agents must adopt a common goal regarding migration. The agents must negotiate in order to make a decision as to which agents should stay and which should leave as several agents want to stay on the device. It is essential that consensus is reached prior to the critical point beyond which the chances of a common goal being adopted are considerably reduced. The critical point is the time at which there is not enough power to transfer all of the agents. Beyond this point all of the agents will want to migrate because it becomes about survival.

There are a number of scenarios that can occur. If all of the agents are altruistic, they will have difficulty deciding which agents to save and which to leave behind. They identify this problem quite rapidly and begin a random select process. This is the virtual equivalent to the drawing of straws. The ultimate test of altruism is then for the agent that draws the short straw. It must be remembered that not all agents are altruistic. Malevolent agents will attempt to terminate other agents in order to guarantee their survival. Another scenario that can occur is that all agents are selfish. In this case all of the agents might attempt to migrate at the same time. This results in some, if not all, of them perishing. If there is a combination of altruistic and selfish agents, the selfish agents migrate and the altruistic agents perish. When programming agents the developer models high priority agents that operate in risky environments as selfish so as to increase their chances of survival\(^2\). It might well be the case that the agent is designed to alter its behaviour from selfishness when there is no resource shortage. That is, the agent is sensitive to context.

System designers often want agents to give up a certain amount of local autonomy to facilitate a global optimal usage of resources whereby all are better off. A balance must be struck, however, between societal authoritarianism and anarchy. In AFME the balance is in favour of local autonomy and decentralised

---

\(^2\) This will only increase their chances provided that some of the others agents are not selfish.
decision making rather than enforced societal rules. To encode normative behaviour in AFME agents must adopt social roles. That is, the agents must be aware of the rules of the society in question. The decision as to whether they adhere to the rules, however, is made by the individual agents. It depends on their utility values. It might well be the case that it is in an agent’s interest to adhere to the rules. Nonetheless, when resources are scarce society tends to break down. Agents drop their social roles in order to survive. This is also true in the real world and can clearly be observed in disaster situations. If there is not enough resources for all, people tend to protect their own interests.

The problem of getting the balance right between local autonomy and societal authority is somewhat similar to what Bellman referred to as ‘the macroscopic principle of uncertainty’ in control theory [21]. In a similar vein to the microscopic principle of uncertainty in quantum mechanics, the macroscopic principle of uncertainty in control theory concerns getting the balance right between using a small amount of data and making an early choice, or taking a large amount of time to receive a significant amount of data before making a decision. This is a non-trivial problem. It led Bellman to conclude that “There is no way to control a large system perfectly”. Either we accept errors due to the lack of information or we allow to system to carry on regardless as we are collecting data. There is an inherent cost in controlling a system. At one extreme anarchy prevails, there is no control, and at the other extreme the benefits of decentralised decision making are lost due to too much societal control. Various types of societies may be engineered in AFME; it depends on the requirements of the problem the developer is trying to solve. In AFME the societal structure is capable of dynamically altering itself at runtime so as to adapt to context and to handle emergent system requirements.

On working on approximations connected with neutron transport theory, as part of the Atomic Bomb project in Los Alamos, Bellman identified the need for powerful numeric techniques. This led him to work on the principles of optimality and invariance and ultimately to the theory of dynamic programming [15]. Dynamic programming forms the basis of the extended rational decision making capabilities introduced to AFME (see section 6.2). We would hasten to add that technology itself is not a good or a bad thing; it’s what it is used for. Some might argue that the development of atomic weapons could never be justified but what if an asteroid were coming? It is ironic that what once was considered to be the greatest threat to human civilisation might some day be what saves it. Our job as scientists is to find out what can be done. It’s a question for society as to what should be done.

As in probabilistic decision theory models, such as those based on the Savage axioms [22], dynamic programming provides an efficient solution. The decision making capabilities of AFME differ from classical probabilistic decision theory. A number of flaws in Savage’s arguments are identified in [23]. Beliefs cannot be represented by additive probability distributions [24].

\[13\] Future work will investigate the use of enforcement.

\[14\] It is likely that a controlled nuclear reaction would be used rather than a bomb.
8 Conclusion

This paper detailed three new features of AFME that were introduced to enable resources to be managed more effectively. The ability to manage resources in an intelligent and prudent manner is a key requirement in the deployment of intelligent agents on computationally constrained ubiquitous devices. The extended rational decision making capabilities enable agents to reason about the relationship between their abstract computational limitations and their commitments. Belief labeling improves the efficiency of the reasoning algorithm by ensuring that duplicated belief sequences are only evaluated once. It decreases development time by reducing the redundancy and improving the maintainability of the software.

The new threading model enables the response times of agents to be specified accurately. Agent response times are altered to be prime numbers to prevent harmonic synchronisation and to ensure that agents are out of tune with respect to each other. Agents with the same response times are phase shifted to reduce computational overlap.

AFME addresses the issue of attempting to get the balance right between local autonomy and societal authority. Agents alter their behaviour according to context and emergent system requirements. It is not claimed that this is a perfect solution to the problem. A perfect solution is impossible due to the macroscopic principle of uncertainty. Nonetheless, it is an adaptive solution.

9 Acknowledgements

We gratefully acknowledge the comments of the anonymous ESAW reviewers, which have significantly improved the quality of this paper. Gregory O’Hare and Michael O’Grady gratefully acknowledge the support of the Science Foundation Ireland under Grant No. 03/IN.3/1361.

References


