

Agent Support for a Grid-based High Energy Physics Application^{*}

Aman Sahani, Ian Mathieson and Lin Padgham

Intelligent Software Agents,
School of Computer Science and Information Technology,
RMIT University,
GPO Box 2476V, Melbourne, VIC 3001
{asahani, idm, linpa}@cs.rmit.edu.au

Abstract. This paper presents an agent system ASGARD-0, that provides monitoring for the success or failure of Grid jobs in a High Energy Physics application. This application area is one where use of the Grid is extremely well motivated as processes are both data and computationally intensive. Currently however there is no mechanism for automated monitoring of jobs and physicists must manually check to see whether the job has completed and whether it has done so in a successful manner. ASGARD-0 provides some initial services in this area and is also a proof of concept for a much more ambitious agent support system.

Keywords: Intelligent Agents, Intelligent Interfaces, Grid Support Services, Systems for Real Life Applications.

1 Introduction

This paper describes work done by the RMIT Intelligent Agents group in collaboration with the High Energy Physics (HEP) group at The University of Melbourne. The HEP group is keen to exploit the advantages offered by grid computing, but has been limited by the uninformative, immature nature of the underlying grid implementation: the current grid middleware provides no scheduling system or even an effective tool to split jobs across multiple machines. Compounding the problem is the lack of job progress monitoring provided.

This paper and the project in general looks at ways in which agents can assist with these problems, as well as assisting in other areas such as data discovery. Ultimately we aim to provide a fully-fledged system to allow an opaque interface to the grid that will allow physicists to perform experiments across various grid nodes. We refer to this future system as ASGARD (Agent Support for Grid Application Research and Development).

^{*} Supported by VPAC Expertise grant EPPNRM121.2004, ARC Linkage grant LP0347025 in collaboration with the Australian Bureau of Meteorology and Agent Oriented Software P/L, ARC Discovery grant DP0346691 in collaboration with the RMIT Spatial Information Architecture Laboratory, and assisted by Tom Gamble.

The initial implementation, ASGARD-0, focuses upon providing the HEP community with feedback regarding the status of a job that has been or is currently running on a grid node, as currently they have no way of knowing whether their job has been successfully run or has failed.

2 The application

One of the main challenges for High Energy Physics is to answer longstanding questions about fundamental particles and the forces acting between them. In particular the goal is to explain why some particles are much heavier than others, and why particles have mass at all.

The answer could reside in an all-pervading presence called the Higgs field, but at the moment there is no evidence of its existence. The University of Melbourne HEP group is participating in the BELLE experiment at the Japanese KEK-B asymmetric electron-positron collider, which generates huge amounts of B-meson decay data in search of evidence for the Higgs field. It is a highly collaborative project where multiple, geographically dispersed groups are using parts, or skims, of this dataset simultaneously.

There are two types of experiments conducted by the HEP group: simulations and analysis. A *simulation*, or Monte Carlo simulation, is essentially a data generation step. It is used to test and calibrate analysis code before running it on the actual BELLE data. An *analysis* is an experiment in which physicists look for particles of interest in a simulated or real dataset. Usually they are interested in a particular decay chain or particle and will perform data cuts or queries on the data to select the events of interest. The subsets of the full data set, containing only events exhibiting the decay chain being studied, are called *skims*.

Through the construction of increasingly sensitive and precise detectors, physicists have overcome to a large extent, the problem of generating useful data. Unfortunately this has led to an unresolved issue of how to extract meaningful information out of the resulting petabyte data collections. Filtering and querying of enormous magnitude is performed upon these massive datasets, leading to a bottleneck in terms of computational time and space.

Compounding the issue is that of the highly dispersed nature of the dataset. There are many organisations around the world involved in this project, each generating their own simulation data and skims from events collected by the BELLE detector. It is essential that the various organisations are able to share this data effectively.

The conventional single processor computational paradigm has proved inadequate in terms of both computational power and resource management/storage. Cluster-based computing, where a number of computers are devoted to the analysis takes advantage of the fact that it is possible to split these files into smaller sub files and perform independent analyses in parallel.

However, the data intensive nature of the HEP experiments, combined with the widely distributed scientific community, make availability of a grid resource highly desirable.

2.1 Problems

Grids clearly offer massive benefits to large projects such as the BELLE experiment. They allow data and resource sharing on an unprecedented level, leading to important collaborative work. However grids are also characterised by a number of problems:

- *Heterogeneity*: The programs and environment at each of the grid nodes are likely to be different.
- *Network Unreliability*: The network connection(s) between a local node and a remote host may go down during the execution of a job.
- *Network Cost*: It is costly to transmit large amounts of data.
- *Security*: Extra security and authentication procedures are required in order to guarantee computational integrity and authorise resource access.
- *Dispersed administration*: While having no central administration and monitoring facility is essential in some regards, it leads to the nodes on the grid being unreliable. Users submitting jobs to grid nodes may not know if that node is functioning correctly or at all.

All these can lead to a high failure rate of jobs that have been submitted to a grid. Our challenge is to reduce this failure rate by introducing an agent system capable of pre-empting failure through intelligent scheduling and submission, and intelligent recovery and resubmission following the failure of a node to successfully complete the job. Our initial implementation demonstrates an agent tool capable of detecting failure and reporting this in a meaningful way to the user.

2.2 Globus

The interest and need in grid computing has led to attempts to develop middleware capable of supporting grid applications. Globus [4] has become the most accepted (and indeed the default) standard for providing these services, although there are others (such as Legion [6]). The European Data Grid (EDG) has been developed as an extension of Globus, and is itself being extended to the “Large Hadron Collider” Grid (LCG) for a new set of experiments, known as ATLAS. The HEP group at the University of Melbourne has a test grid using the Globus Toolkit.

Globus defines an open source toolkit of low-level services for security, communication, resource location and allocation, process management and data access.

There are two major issues or problems that grid middleware, including Globus, have yet to address:

- *Lack of monitoring*: Globus allows an end user to check the status of a submitted job. Unfortunately this capability proved relatively primitive. There is no differentiation between failed and successfully completed jobs.
- *Lack of scheduling*: When a job is submitted to a Globus grid, the machine name of the remote host must be specified. An explicit mechanism for submitting to hosts is far from the goal of a system that has abstracted the underlying grid from the user.

These two problems do not mean that the Globus Toolkit is not useful, but emphasise the fact that the Toolkit is only a building block upon which developers can construct useful applications.

ASGARD has been built to use the Globus toolkit, but it should be readily adaptable to EDG and LCG, when physicists inevitably migrate, due to their similarities.

3 Agents for the Grid

Agents have been increasingly used in complex and dynamic applications [5]. Their proactive autonomous nature, combined with the ability to react to changing situations, makes them very suitable for grid environments, which are likely to be highly dynamic.

BDI (Belief Desire Intention) agent systems [1, 9] are particularly suitable due to their fault tolerant behaviour, and their commitment to continue to pursue a goal. These systems typically provide a number of plans with which an agent can attempt to achieve a given goal. This enables the agent to choose dynamically the most suitable plan for the given situation. If the plan unexpectedly fails, an alternative plan can be chosen in an attempt to achieve the goal. They can also monitor the environment and adapt the choice of plans accordingly. To encode this behaviour in a traditional system would be difficult at best, and probably result in a complex, brittle application, whereas it is innate in BDI style agent systems.

We have chosen to use JACK Intelligent AgentsTM [2] as our BDI platform, as it is a well developed and robust system which is easily integrated into larger applications and provides extensive functionality.

A number of research groups are also working on agent support for the Grid. ARMS is a scheduling system for grid computing that uses the A4 design methodology and the PACE toolkit for internal resource scheduling. The A4 methodology describes each agent as a representative of a local grid resource.

AgentScape [7] provides middleware support (AOS) for developing agent applications via a virtual machine distributed across a WAN with heterogeneous hosts.

Both AgentScape and ARMS rely on the Grid having these systems installed on all Grid nodes. Our approach on the other hand does not rely on installing our agent system on all nodes. Rather the agent system operates at a local node, while communicating with the Grid infrastructure. This can be seen as a disadvantage and also an advantage. On the one hand a system such as AgentScape,

with an AOS at every node has the potential to provide a great deal of information about the state of the grid, the network, and all the nodes in the grid. However installing virtual machines across every node in the Grid is unlikely to be achievable in the near future as nodes are generally under varied institutional control. In this immature grid environment we believe our ASGAR architecture is more practical.

4 Overview of ASGAR Architecture and Design

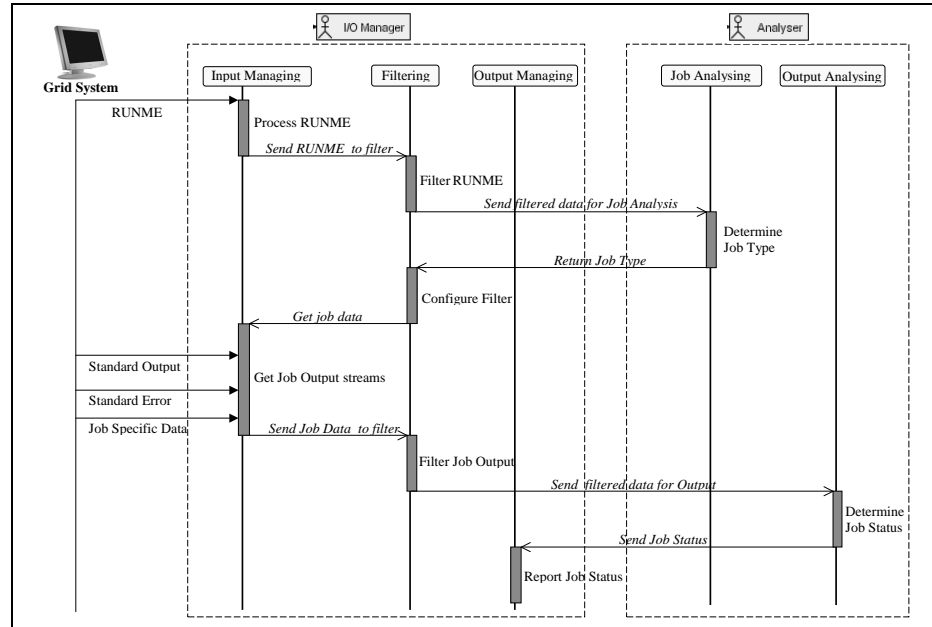


Fig. 1. Message flow within and between the ASGAR-0 agents and their capabilities.

ASGAR-0 has been designed using the Prometheus agent system design methodology [8] and the Prometheus Design Tool (PDT), then implemented using the JACK Development Environment (JDE) for JACK Intelligent Agents™ [2] a multi-agent development environment based on the BDI model.

ASGAR delivers intelligent support from *outside* the Grid. Thus it will eventually be able to monitor the commandline and act on the user's behalf.

Normally, after submitting a job to the BelleTestBed, the user would receive a *jobID* and manually check the BelleTestBed until the job appears *DONE*, then use the *jobID* to obtain the outputs from the experiment and visually inspect for signs of failure. Given that jobs can take hours or days to complete, this is

an onerous task and is not efficient. ASGARD-0 abstracts this task of failure detection and diagnosis away from the user and returns the status of the job, along with the location of the data (if any).

In our early experiments, we have concentrated on reasoning about specific problems which arise when using the BelleTestBed, rather than more general problems when connecting via Globus. Initial development was done using sample result files from previous executions on the BelleTestBed.

The system comprises two BDI agents. The *I/O Manager* agent is responsible for facilitating input and output exchanges with the environment and for filtering the data sent to the *Analyser*, which is responsible for reasoning about the job: before, during and after its execution (see Figure 1).

The *I/O Manager agent* acts to decouple the Analyser from the outside world, by controlling the manner in which inputs are obtained and forwarding only relevant material, as well as delivering any responses to the user. The filtering can be actively configured by the Analyser in response to the job description as well as during execution. The output can be reported in various forms including console messages, text reports, XML reports etc.

The *I/O Analyser agent* is responsible for determining the job type as well as reasoning about the job in order to determine the job status. The two distinct analysis phases, or aspects of the agent, are split into separate capabilities: Job Analysing and Output Analysing, also illustrated in Figure 1.

The *Job Analysing* capability receives the filtered job description data which helps the agent reason and determine the job type. Once the job type is determined a message is sent to the I/O manager agent in order to configure the filter.

The *Output Analysing* capability is the most important module of the system as its task is to determining the job status. It is discussed in more detail in the next section.

5 Output Analysis

This capability is the heart of the system, and requires the collection of input data, detection of potential errors and reporting of the job status upon successful termination of input. These tasks are delegated to the following sub-capabilities: Input Scanning, Error Handling, and Termination Handling, as shown in Figure 2.

The *Input Scanning* capability receives as input filtered job data from the standard output, standard input and other job specific files (if any) and stores this information about what it has seen as a set of beliefs, implemented as the JACK *beliefset* `JobOutputData` shown in Specific messages received may generate diagnosis goals which require further evaluation.

If potential error messages are encountered then a diagnosis goal causes plans to be chosen to try and ascertain the cause of the error These plans are part of the *Error Handling* capability to reason about the error and generate appropriate status messages. Similarly, if job termination messages are encountered then a

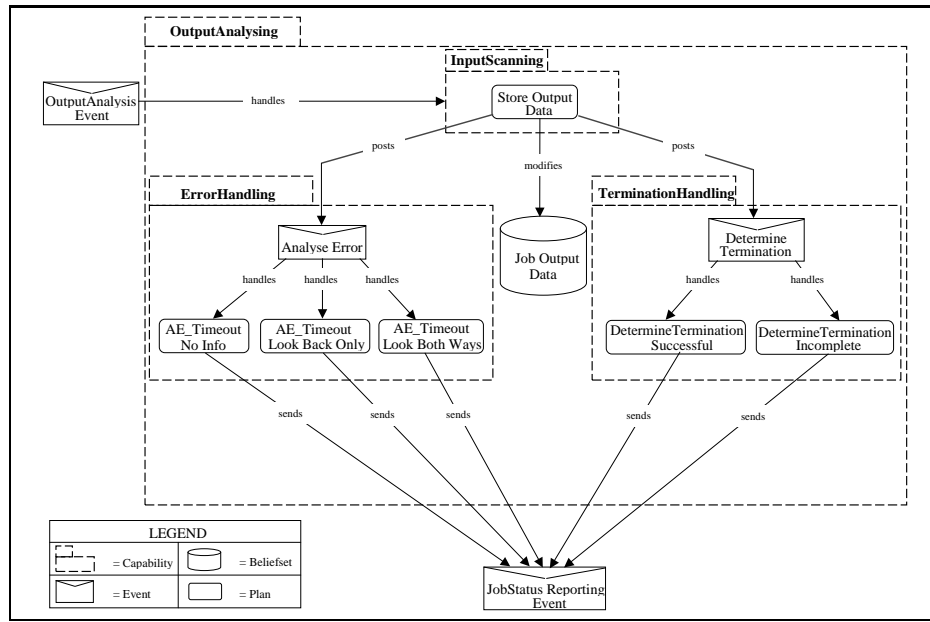


Fig. 2. Analyser capabilities, beliefsets and plans. Note that all plans have *read* access (not shown) to the beliefset, but only the *StoreOutputData* plan *modifies* the beliefset (as shown).

goal is generated to determine whether or not the job has terminated successfully. The relevant plans then reason about the successful/unsuccessful termination of the job and report the status back to the I/O Manager agent. These plans are part of the *Termination Handling* capability.

The purpose of the *Error Handling* capability is to identify and analyse errors as and when they are encountered and to report the reasons responsible for them being generated. The goal which triggers this capability is *AnalyseError* (see Figure 2). Depending on the content of the message associated with a particular instance of this goal, different plans are chosen to reason about possible causes, or to monitor for future messages which may provide information about the cause.

An example of a particular error is the “Timeout” error. In this case, the content of the message associated with the *AnalyseError* goal starts with the string "FPDAGRID: ERROR - Timeout". There are three plans which attempt to find a cause for this error:

- AE_TimeoutLookBothWays (AE stands for Analyse Error),
- AE_TimeoutLookBackOnly, and
- AE_TimeoutNoInfo

These plans are all applicable for the situation where the error encountered is a timeout error. They are tried in the order shown above, where the

AE_TimeoutNoInfo plan is simply a default plan which provides (to the I/O Manager) the information that a timeout error has occurred but the cause has not been determined while each of the first two plans look for certain messages in the beliefset that could have led to the timeout error being generated.

The first plan, AE_TimeoutLookBothWays, handles the most common situation. It does the following steps:

- wait until “event_server” error message is encountered.
- Access belief set data to look for additional information on reasons for the error.
- Report Error with details.

Once the “event_server” error message has been encountered, the plan can be certain that a “Timeout” error has occurred and can then, based on the information obtained, look in its beliefset for further information as to the cause. Items such as file availability messages and any corresponding errors that may be generated, provide further information about the sequence of events that have contributed to the “Timeout” error being generated.

TIMEOUT error messages are generated by low level library routines at the grid level and they indicate the possibility of an error. The “event_server” messages are generated from an application interfacing with the grid and they provide details about the error. However there is a possibility that an “event_server” error message may not be received, even though a TIMEOUT error has occurred. In this case, if EOF is encountered (in any of the input sources) prior to an “event_server” message being received, the first plan fails, and the agent will try the AE_TimeoutLookBackOnly plan in order to achieve the goal of analysing the “Timeout” error. This plan tries to look through the beliefset to find any evidence confirming the generation of a “Timeout” error, and associated information regarding causes. This is quite similar to the first plan but initially lacks conclusive evidence to help guide the further search for reasons. Once the evidence is gathered this plan then also tries to find further details of the error.

This plan tries to look for items such as server error responses in the beliefset. If such a message has occurred more than once then the plan succeeds at confirming the timeout error. The error detail (erroneous filename) is retrieved by looking up the Grid URL message and is reported back to the user.

In the unlikely event of the AE_TimeoutLookBackOnly plan failing as well as the AE_TimeoutLookBothWays plan, the AE_TimeoutNoInfo plan is executed. It simply reports that a “Timeout” error was observed but that no further information has been obtained. It reports only the detection of the error rather than diagnosing and specifying the details about it.

As new ideas on what to look for to try and diagnose each kind of error message are provided, these can readily be mapped to self contained plans which capture the analysis process.

Job Termination is handled by the *Termination Handling* capability. The goal is to determine the successful completion of the job upon job termination. This goal is represented by the *DetermineTermination* event (see Figure 2).

This capability has two alternative plans to achieve this goal. DetermineTermination_Successful and DetermineTermination_Incomplete in order of priority. These plans are considered if the following context conditions are *both* true:

- The agent has seen an EOF message from stdout (Job Status Message = EOF and Message Source = stdout).
- The agent has seen an EOF message from stderr (Job Status Message = EOF and Message Source = stderr).

If in addition the agent has seen the job status messages "FPDAGRID: User Cleanup" and "Processing End..... Removing IPC..... done" then the DetermineTermination_Successful plan is chosen as appropriate and a successful execution message is reported back to the user via the I/O Manager agent.

If the job status messages "FPDAGRID: User Cleanup" and "Processing End..... Removing IPC..... done" have not been seen by the agent (i.e. it is not the case that the beliefset contains these messages), then the DetermineTermination_Incomplete plan is appropriate and it simply reports an abnormal termination and execution incomplete status.

6 Conclusions and Future Work

ASGARD-0 is an initial but useful first step in providing intelligent agent assistance for grid computing, in the context of the HEP application. Being able to detect failures and reason about them (if not handle them) is useful, but our longer term aims are to provide more comprehensive support. Importantly we have demonstrated that it is possible to provide useful agent support services for the grid, without having the agents embedded tightly into the internal infrastructure of every grid node. In future work we hope to align with and make use of both our own and others' work within the world wide web research community.

The semantic grid is an extension of the semantic web, where the grid (or web) is defined as a service-oriented architecture in which services are provided to and from entities using an advertised contract system. These services are "advertised" through the use of standard Service Description Languages (SDLs). The Open grid Service Infrastructure (OGSI [10]) defines these WSDL specifications for use consistent with grid-based architectures. The ability to recognise and provide information about computation services on the grid is one aspect of the future development of ASGARD.

HEP analyses are typically very long processes, dealing with massive amounts of data. It is however possible that someone has already performed the analysis or simulation. If matching data exists somewhere on the grid, then it may make sense to locate and use this data, rather than reproduce it. An ASGARD agent could locate this data, by viewing the existing data as a service offered by grid nodes and avoid duplication.

If the analysis itself is viewed as a service, then it may be possible to further augment the BELLE process using ASGARD through semantic optimisation. According to the physicists, "80% of computation is duplicated" with the early

stages of many experiments being identical. However there is no way of getting access to other people's intermediate data. The cost of storage is the primary reason for this – it is completely infeasible to store even the output of a large number of experiments, much less the intermediate data. ASGARD could potentially recognise that two (or more) analyses were aligned in at least part of their process (more likely the earlier parts of the analysis) and run them as one single analysis, diverging them at the appropriate point. In the discourse of service composition: computation could be seen as the service, something supported by OGSi. Having each particular experiment advertise the type of analysis currently being undertaken, it may be possible to merge scheduled experiments.

The method we have used for providing intelligent agent services to grid applications indicates that this is an effective and relatively straightforward approach, as there is no necessity to obtain agreement from all nodes before the approach can be trialled. Even if it is desired at a later stage to integrate the services more fully within the grid infrastructure, the approach of having loosely attached agents in order to trial and refine such services is very promising.

References

1. Bratman, M. E.: *Intentions, Plans, and Practical Reason*, Harvard University Press, Cambridge MA, USA.
2. Busetta, P., Rönquist, R., Hodgson, A., Lucas, A.: *Jack Intelligent Agents - Components for Intelligent Agents in Java*, Technical Report 1, Agent Oriented Software Pty. Ltd, Melbourne, Australia. See web site at <http://www.agent-software.com>.
3. Cao, J., Jarvis, S. A., Saini, S., Kerbyson, D. J., Nudd, G. R.: 'ARMS: An agent-based resource management system for grid computing' *Scientific Programming* **10**. (2002) 135–148 (Special Issue on Grid Computing)
4. Foster, I., Kesselman, C.: The Globus Project: A Status Report *in* 'Proceedings of the Seventh Heterogeneous Computing Workshop' IEEE Computer Society. (1998) 4–19 See web site at: <http://www.globus.org/>.
5. Jennings, N., Wooldridge, M.: Applications of Intelligent Agents *in* Jennings and Wooldridge 'Agent Technology: Foundations, Applications, and Markets', Springer. (1998) 3–28
6. Natrajan, A., Humphrey, M., Grimshaw, A. S.: Capacity and Capability Computing in Legion *in* 'Proceedings of International Conference on Computational Science (ICCS)', Lecture Notes in Computer Science **2073**, Springer Verlag. (2001) 273
7. Overeinder, B. J., Posthumus, E., Brazier, F. M. T.: Integrating Peer-to-Peer Networking and Computing in the AgentScape Framework *in* 'Proceedings of the 2nd IEEE International Conference on Peer-to-Peer Computing', IEEE Computer Society. (2002) 96–103 See web site at <http://www.iids.org/research/aos/>.
8. Padgham, L., Winikoff, M.: *Developing Intelligent Agent Systems: a practical guide*, John Wiley and Sons, England. (2004)
9. Rao, A. S., Georgeff, M. P.: An Abstract Architecture for Rational Agents *in* C. Rich, W. Swartout, and B. Nebel, eds, 'Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning', Morgan Kaufmann. (1992) 439–449
10. Tuecke, S., Foster I., Frey J., Graham S., Kesselman C., Maquire T., Sandholm T., Snelling D., Vanderbilt P.: *Open Grid Services Infrastructure (OGSI)*, Version 1.0, Global Grid Forum. See web site at <http://www.ggf.org/>.