An Environment for Workflow Applications on Wide-Area Distributed Systems

H.A. James†, K.A. Hawick† and P.D. Coddington‡
†Distributed and High Performance Computing Research Group
School of Informatics, University of Wales Bangor
Dean Street, Bangor, Gwynedd LL57 1UT, Wales, UK
Email: {heath,khawick}@informatics.bangor.ac.uk
Tel: +44 1248 382717 Fax: +44 1248 361429

‡Distributed and High Performance Computing Research Group
Department of Computer Science, University of Adelaide
Adelaide, SA 5005, Australia
Email: paulc@cs.adelaide.edu.au
Tel: +61 8 8303 4949 Fax: +61 8 8303 4366

Abstract—

Workflow techniques are emerging as an important approach for the specification and management of complex processing tasks. This approach is especially powerful for utilising distributed data and processing resources in widely-distributed heterogeneous systems. We describe our DISCWorld distributed workflow environment for composing complex processing chains, which are specified as a directed acyclic graph of operators. Users of our system can formulate processing chains using either graphical or scripting tools. We have deployed our system for image processing applications and decision support systems. We describe the technologies we have developed to enable the execution of these processing chains across wide-area computing systems. In particular we present our Distributed Job Placement Language (based on XML) and various Java interface approaches we have developed for implementing the workflow metaphor. We outline a number of key issues for implementing a high-performance, reliable, distributed workflow management system.

Keywords: Workflow; DISCWorld; distributed computing; Java; XML.

I. Introduction

Workflow is a technique by which a complex process is expressed as an interconnected series of smaller, less complicated, tasks. The concept of workflow [11] has successfully been used in many areas of human endeavour, including industrial and administrative process management, and also major design tasks. Workflow concepts are fundamental to the theory of computation on many levels. The most fine-grained level is that of machine-level instructions, where commands are directly executed by the computer's central processing unit, usually causing modification to the data stored in memory. This style of workflow is too fine-grained for most practical purposes. At a higher level workflow is used by both main styles of computer programming: imperative and declarative. In *imperative* programming the user specifies a list of commands to be executed on data and the control of execution lies with the commands [3]. In contrast, the declarative paradigm of computing known as *functional*, or *dataflow* computing usually involves the specification of a program as a series of higher-level instructions that execute as soon as the data they need to work becomes available [1, 25]. These two approaches differ in the emphasis they place on data: the first views data as passive, being passed between instructions, while the latter views data as the enabling mechanism, without which the instructions could not fire.

The field of distributed computing is considerably

aided by the ability to express complex tasks in terms of groups of component tasks. The current pervasiveness of the Internet has led to a rapid increase in the number and complexity of distributed computing applications. When computations are to be distributed across physically separate processing nodes, care must be taken to ensure that the amount of computation performed by each task is enough to ameliorate the cost of inter-task and inter-node communications. One of the ways in which workflow can aid the field of distributed computing is by allowing the abstraction away from individual instructions to that of *high-level services*. This provides a suitable granularity of tasks (in which the ratio of computation to communication is high) that can be efficiently manipulated in the distributed system.

Workflow ideas are particularly useful for designing a software architecture for handling service-based distributed computing. We have developed a distributed computing environment, known as DISCWorld, which is based on workflow principles. DISCWorld acts as distributed computing middleware, providing transparent access to remote data storage and compute servers, and handling issues such as scheduling and placement of data access and processing services. We have deployed this environment in the development of decision support applications involving processing and analysis of geospatial imagery such as satellite data [5–8, 32].

In this paper we describe our DISCWorld environment and how it can be used to construct workflow applications that may be executed across multiple computers distributed over a wide-area network. DISCWorld allows complex processing chains of services to be specified, without the need to know where each service is to be executed. We also describe the technologies we have developed to facilitate the workflow abstraction, and discuss issues arising from implementing a high-performance, reliable, distributed workflow management system. DISCWorld, the distributed computing environment that we have designed and implemented, is described in section II. The use of workflow metaphors, and the technologies we have developed are described in section III. Section IV describes an example application we have developed using workflow and DISCWorld, and section V discusses some issues for implementing a high-performance, reliable, distributed workflow management system. The paper is summarised and future work is discussed in section VI.

II. DISCWorld Distributed Computing Middleware

Our approach to the problem of workflow management in distributed computational environments is called Distributed Information Systems Control World, or DISCWorld [17]. This section provides a general description of DISCWorld and describes the way in which it helps with workflow problem management.

DISCWorld is a service-based metacomputing [27] (or grid computing [10]) environment, providing high-level middleware to enables transparent access to data and compute resources distributed over a wide-area network. The fundamental ideas behind the DISCWorld environment are very simple: it essentially consists a collection of federated servers with hosting a DISCWorld daemon which acts as a broker to applications, or *services*, and data.

Services can produce multiple outputs and can use multiple inputs. Multiple services can use the same output of a previous service. This has the logical effect of copying the shared data. Data produced by a service is immutable – once a data item is created it cannot be modified. The effect of a service on data is the creation of new, or *derived*, data. This immutability is achieved by canonically *naming* all data in DISCWorld; references to data are via their name, not by any location-dependent memory pointer [16].

Within the DISCWorld environment the use of canonical names for objects (data and services) is crucial. We make the assumption that if the same service is executed with the same input data on multiple occasions, the output data is named the same every time. Furthermore, we make the assumption that if two separate DISCWorld objects have the same name, they are defined to be two instances of *the same object*. This assumption becomes vitally important when we consider the topics of remote data references.

Clients compose *processing requests*, which may be very simple or quite complex, comprising of a number of cooperating services. The processing request is arranged as a directed acyclic graph (DAG). DAGs are a standard mechanism for representing dataflow and workflow processes. Connections (graph edges) between services (graph nodes) represent data shared between the services. The graph is directed since data flows in a single direction only. We do not allow cycles in the graph, which would represent recursion. Users have two ways of composing processing requests – through

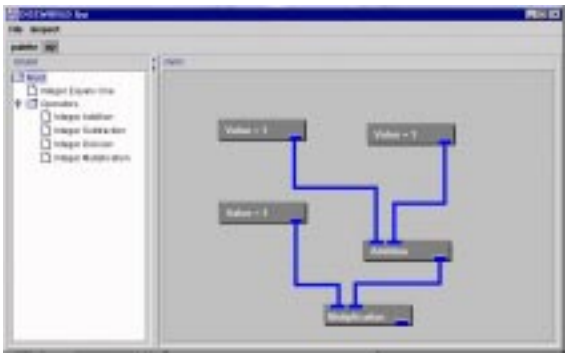

Fig. 1. Distributed workflow calculator example. Requests are formed as directed acyclic graphs of services. The DISCWorld environment provides an execution context for the user processing request.

a text-based scripting language (described in section III-B) and via a graphical tool. The graphical tool is shown in figure 1. We allow remote references to data and services in the DISCWorld; this is described in section III.

Once a user has specified their processing request it is sent to their local DISCWorld daemon. The daemon inspects the contents of the processing request and invokes a *placement algorithm* on the request. The placement algorithm assigns each of the services to a processing node in a way that attempts to minimise both the execution time of the whole request and the amount of data that must be transferred between nodes. Due to the large number of processors, and possible sources for data and services, the placement process is may not be optimal. A fitness heuristic is used on a "near" subset of available processing resources to produce a good (near-optimal) solution. The federation of processing resources which host the DISCWorld environment all make their own placement and scheduling decisions independent of each other. The DISCWorld environment provides the necessary framework for the seamless scheduling and execution of the component parts of the users' requests across the distributed system [21], and the necessary security and fault tolerance to ensure the request is fulfilled. The workflow metaphor is used to divide the problem into smaller, discrete, services connected by data transfer. Workflow ideas are central to the architecture of the DISCWorld distributed computing environment.

III. Workflow in DISCWorld

There are a number of applications in conventional and distributed computing that use workflow techniques, including AVS [2], Ptolemy [12] and Khorus [22]. AVS and Khorus are tools for visualising scientific data sets. Ptolemy is a tool for the specification and execution of low-level process networks in a distributed system. It is mainly used for dataflow computation.

These applications allow the user to graphically compose a graph of modules that are executed together to perform a processing request. Typically users must be able to choose the input data to their processing request via the file system (whether the files are local or remotely accessed via NFS) and each of the modules must be able to execute locally. Usually there is no facility whereby the user can specify input data or processing modules that are stored in a data archive on a remote machine. Furthermore, systems like AVS are designed for short-running visualisations with a high degree of user interaction, whereas the DISCWorld daemon is intended to be a long-running process, even if individual processing requests are relatively short. While AVS does provide rudimentary support for distributed computing, it is limited to use those machines that have been configured to run predefined processing modules; no advanced heuristics are used to decide where to place processes for efficient execution.

DISCWorld aims to provide a fault-tolerant, reliable, distributed computing solution in wide-area environments. We feel such a system must be adaptable to the following conditions:

- the services that make up the process network may be remote;
- the data to be used may be remote, or stored within a data archive;
- the data may have to be moved to the location of the service, or vice versa, in order to execute the process network;
- due to ownership or restricted usage conditions, data or services may not be allowed to move between physical machines; and,
- there may be multiple copies of the same module in the distributed system, perhaps implemented in different ways, for example high-performance parallel code which can only be run on a parallel computer, or a portable lower-performance serial version written in Java that can be easily transferred between machines.

Workflow techniques based on access to well-defined services are fundamental to the design of DISCWorld. DISCWorld's abstraction over services allows a greater flexibility than found in most other distributed systems [17]. The remainder of this section describes the workflow techniques used within DISCWorld and the abstractions we have developed

to facilitate the metaphor.

In order to achieve a solution to the above problems, there must be some way in which data, that may be on a local *or* remote processing node, can be referenced. This is one of the hardest problems in distributed or grid computing. Typical methods for referencing data objects use pointers into a remote process' memory space [4,13] or use an object broker [24]. The fundamental limitation of these approaches is that if the remote machine needs to be restarted, then the object to which the reference points may not be recreated, and if it is then it most likely will not be recreated at the same memory location. Therefore the pointer becomes useless. This is known as the "dangling pointer problem".

In section III-A we describe our approach to the problem of distributed remote object references, the DISCWorld Remote Access Mechanism (DRAM). In section III-B we describe the language used to represent complex processing requests in DISCWorld and how these object references are used in request execution.

A. DISCWorld Remote Access Mechanism

The DISCWorld Remote Access Mechanism is used to reference objects which may reside on local or remote processing nodes. The DRAM is a *rich pointer* because as well as containing an object reference, it contains additional metadata that can be used, for example, to decide whether the data can be moved between machines, or whether it is more feasible to move the data to a remote machine for processing or move the service to the data. The differences between the approached mentioned above and DRAMs are discussed in [16].

Data and services within the DISCWorld are given a canonical (global) name. This name reflects the "recipe" that can be used to reconstruct it, if necessary. DRAMs created from data or services share the canonical name. The canonical name makes no reference to either a processing resource or memory location. This enables us to avoid the "dangling pointer problem" faced by other remote-reference systems.

The basic structure of a DRAM is shown in figure 2. We have chosen to implement the DRAM in Java [28]. This decision allows us to leverage off many of Java's useful features, including byte-code portability, run-time byte-code verification, and consistent serialisation behaviour. Of particular importance to the workflow metaphor is the use of the object's size, global name and mobility in the DRAM.

```
public abstract class DRAM implements Serializable
{
  private String publicName;    // descriptive name for GUI use
  private String globalName;    // internal ID
  private Icon icon;            // associated icon e.g. thumbnail image
  private String description;   // long free textual description
  private String className;     // queryable searchable text
                                //   representation of class
  private String remoteServer;  // location of the Data to which
                                //   the DRAM points
  private int objectMobility;   // whether the object being pointed to
                                // can be transferred across
                                // the network
  private int objectSize;       // size of the object being pointed
                                // to, in bytes
}
```

Fig. 2. DRAM Java base class. The DRAM provides a high-level pointer to an object.

When data is created within the DISCWorld, or is imported into DISCWorld, the data's size is fixed. Any services that use the data as input result in new data being created, which is logically separate from the original. The `className` field in the DRAM specification allows us a rudimentary type system.

As DRAMs have the same names as the object from which they originate, they can be passed between DISCWorld servers and clients in lieu of transferring the original, bulk, data. This avoids the needless transfer of large data objects to servers that may only need to pass the object on, unused. Figure 3 shows the situation where both servers possess a remote pointer to the same data; both pointers have the same name as they refer to the same object. DRAMs are first-class objects; they can be sent between DISCWorld servers in the distributed system. Only when the original data is actually needed, is a transfer performed.

Whether the transfer is able to proceed is determined by the object's mobility. As mentioned in the previous section, some objects (data or services) may be subject to restrictions on whether they may be copied, or moved between machines. Such restrictions must be taken into consideration when making the decision of where to place services and data to ensure optimal (or at least near-optimal) performance.

When the user has created their processing request, it is sent to a DISCWorld daemon for processing. An approximation of the time the processing request will take to execute is made, based on the local DISCWorld node's current knowledge of the available source of data and service byte codes. The approximation takes into account the reported slowdown in service execution time caused by multiple services executing concurrently on a single processing resource, and the time taken to transfer data or service byte code to another machine. DRAMs are used extensively in the process of setting up a pro-

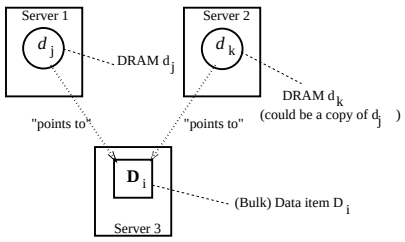

Fig. 3. Many DRAMs may point to a given DISCWorld object. Each copy of the DRAM has the same name. A DRAM may be sent between machines. The data (or service) a DRAM points to is only copied (or moved) upon explicit request.

cessing request. The time estimate is used to create a new set of DRAMs, which refer to the data that *will be* created when the processing request is executed. These are called *DRAM Futures* [20]. When a request is made to retrieve the data to which a DRAMF points, the processing request that eventuated in the DRAMF being created is actually executed. DRAMFs have allowed us to implement a form of lazy evaluation in our DISCWorld workflow environment. Only the required services are executed, thus saving on computational resources.

Optimisations are allowed on the DAG structure submitted as a processing request. The server handling the request may be aware of one or more servers that already possess the data which is to be created, or have previously supplied DRAM references to the data sought. If the data is available, and the time spent transferring it will not exceed the time spent waiting until the DRAMF's data is created and transferred, then the decision may be made to transfer the data from an alternative source. By the same reasoning, if there exists another reference pointing to a different data source, and the estimated time is lower than that returned, the alternative reference may be followed in the expectation that either the computation has already been requested by another holder of the reference, or the data should be available sooner due to the lower estimated arrival time. The method by which daemons are made aware of DRAMs on different servers is beyond the scope of this paper [26].

DRAMFs can be sent between servers, and may also be used by clients in the composition of new processing requests. What this means is that a DRAMF may be copied or moved to many other servers. Subsequently, it may be used as an input to a service by the scheduler, and only when that processing request is *executed*, will the data to which that DRAMF points be copied to a remote machine. By using the DRAM Futures mechanism, and more generally, the DRAM mechanism, unnecessary bulk data transfers are prevented.

In the DISCWorld model, the partial results of processing requests are deemed to be as significant as the final results. Therefore, in addition to sending the DRAMFs to the nodes that may use them, they are also sent to the daemon or client which submitted the processing request. In the case of the user client, DRAMFs corresponding to all the partial products are returned – if the user wishes to view the contents of the DRAMF, they can request the DRAM's contents. As a DRAMF represents the result of a remote computation that has not begun execution, when the results are requested the service is started, and the result is returned when available. When the DRAMF is inspected by the user client or any other daemon, a request is sent to the producing daemon, which initiates the computation.

Using partial or final data products of previous processing requests allows the possibility of pruning the execution schedule for the current processing request. Thus, if there is no need to re-compute a data product, then the system will avoid it if possible. Of course, the situation may arise where a user has followed a reference corresponding to a final data product, causing the computation to be started, and at the same time following a reference to a partial data product. If the reference that is an input to the service which is to produce the final data product has an estimated time greater than a cheaper source of the same output data, then the data may be retrieved from the alternate source, even though by following a partial-product reference, the user has dramatically reduced the time it would take to retrieve the data. This case is not addressed in the current implementation of the DISCWorld environment – we simply make a best-estimate of the execution times, and settle for that.

When a daemon creates DRAMFs to some future data, the service does not automatically begin executing. The agreement by a daemon to execute a service with given parameters is, however, binding. The daemon has acknowledged that if requested, it will perform the computation and return the result to any requesters. If the input data required by

a service is to be created by a previous service on perhaps a remote machine, then the DRAMFs for the current service cannot be made until the input DRAMFs are received. They are needed in order to estimate the time at which the data, to which the DRAMFs point, should become available. It is in this way that a complex processing request may be set up. This allows resource reservations to be made for daemons that will participate in the processing of a request. Thus, while a processing request may be expressed as a network of services connected by data transfer, culminating in the production of some desired data, its execution is constructed in reverse order. Unlike other models of resource reservation, multiple services can be reserved on a single DISCWorld node simultaneously.

A mechanism is needed to express the services and the data that comprise the processing request. We call our mechanism for this the Distributed Job Placement Language (DJPL). It is designed as a platform-independent specification language.

B. Distributed Job Placement Language

One of the fundamental mechanisms that separates grid computing environments from most batch queueing systems or simple client-server systems is the ability to specify a number of operations that are to be performed on some data, and have the operations individually scheduled in a set of distributed compute servers. This can be done without the intervention of the user at every step of the processing, whether that intervention is the initialisation of the next processing step, or the submission of raw or derived data to a new program. The mechanism used to express processing requests in DISCWorld is the Distributed Job Placement Language (DJPL) [14].

In a high-level distributed system, user requests are represented as process networks of metacomputing services linked together by the sharing of data. Such sharing of data may be due to the services' reliance on the same input data, or the sharing may be more explicit, as in a producer-consumer relationship. Such process networks may be specified and their services statically assigned to hosts in the distributed system. This approach can lead to optimal solutions in the case where the characteristics of the distributed system are known in advance, but especially if all characteristics are *not* known in advance, the dynamic allocation of services to hosts and the optimisation of execution schedules can produce the best near-optimal solution [21].

This section describes the DJPL, and how it is used to specify a complex chain of processing steps on behalf of the client, whether the client is a human user or another DISCWorld daemon. DJPL is implemented in XML [33]. XML provides a platform-independent, unambiguous, extensible mechanism with which structured data can be represented within a text file. Furthermore, there are a number of freely available XML parsers (e.g. IBM's XML4Java parser [18]) which will validate the input against a Document Template Definition (DTD). Relevant parts of the current DJPL DTD are shown in figure 4.

The DJPL is designed to encapsulate all the information necessary for a client's request to be executed by a daemon within the DISCWorld environment. The client can be either a human user or another DISCWorld daemon that is making an execution request. At a minimum the information required by the DISCWorld daemon in order to execute the request is: the user's identification; the location from which this request is being submitted; and a list of instructions that comprises the user's processing request.

An example of the DJPL is shown in figure 5. This script shows a processing request to invoke three services on a series of integer inputs. While the actual services do not matter, the example illustrates that the user's DAG has been flattened, and is able to be reconstructed by any program that can access the DTD and input script; we are not bound to any particular implementation language as we would be if we had simply serialised the Java DAG representation.

Close inspection of the DTD in figure 4 will reveal that the user identification section is not needed for a script to be syntactically correct. This is done in order to allow reconfiguration of processing requests at runtime, and also to allow the "factoring out" of common services into "meta-services".

The service abstraction used within DISCWorld is of a software "black box", where nothing is known about the implementation of the service except its gross characteristics, as represented in its DRAM. It does not matter to the DISCWorld environment whether the service is implemented as: a single- or multi-threaded Java application; a Java wrapper around a high-performance legacy application [32]; an interface to a cluster-based parallel solution [15]; or as a group of services cooperating to produce the advertised effect. In fact, the service named `generalAdder` in figure 5 is implemented by the script shown in figure 6. Part of our ongoing re-

search is the development of tools to allow is to splice together the individual scripts and also factor out common groups of services.

The major benefit of allowing services and meta-services to be logically equivalent is the ability to optimise request execution at runtime. As a meta-service is represented by a DRAM, which has a mean run-time, a notion of run-time variance, and an originating server, the scheduler/placement mechanism can use this information to both reduce the complexity of the scheduling process, and also to optionally choose the best implementation of the services for the user request's characteristics. For example, if the server scheduling the script in figure 5 finds a more economical implementation of the `generalAdder` service, or perhaps one which is on the same server as the data to be used, it is able to transparently substitute the new service. This is possible only if the names of the services are the same. This is illustrated in figure 7, where two meta-services share the same name, although they are implemented as different task graphs.

IV. Example Applications

An example workflow application that has been implemented within the DISCWorld framework is imagery exploitation and analysis [5]. In general terms, this problem involves the manipulation and processing of satellite imagery to identify features of interest. The results from the feature detection algorithm are overlayed on the image which has been enriched with relevant geospatial data stored in another database. The image can then be passed to a user or a client application such as an environmental monitoring application or a defence command and control system. The problem lends itself to the workflow metaphor as most of the processing decomposes quite naturally into a series of individual processing steps. The program was implemented in Java, which allowed us to use the Java Advanced Imagery (JAI) [29] application programming interface (API), which supports image processing chains described using DAGs, and therefore fits in well with the DISCWorld workflow approach.

Consider the following example. A satellite image is captured and sent to a ground-based receiving station. The image is ingested into an image archive that is registered with the DISCWorld environment [8], thus becoming available for exploitation. A processing request is submitted either by a human image analyst or by an automatic process noticing the availability of a new image.

```
<?xml encoding="US-ASCII"?>
<!ELEMENT DJPL (USER?,JOB?,ALIASES?,INPUTS,OUTPUTS,(INSTRUCTION|FOREACH)+)>
  <!-- USER, JOB, ALIAS sections are optional. INPUTS, OUTPUTS and at
       least one INSTRUCTION or FOREACH is required -->
<!ATTLIST DJPL NAME CDATA #REQUIRED>
<!ELEMENT INPUTS (#PCDATA)> <!-- Lists necessary input parameters to script -->
<!ELEMENT OUTPUTS (#PCDATA)> <!-- Lists the minimum necessary output
       parameters to this script (there may me more in actuality) -->
<!ELEMENT USER (GROUP?,PERMISSION?)>
<!ATTLIST USER ID CDATA #REQUIRED>
<!ELEMENT GROUP (#PCDATA)>
<!ELEMENT PERMISSION (#PCDATA)>
....
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT PORT (#PCDATA)>
<!ELEMENT FOREACH (VARIABLE,RANGE,BODY)>
  <!-- Prototype looping mechanism which avoids recursion in implementation -->
<!ELEMENT VARIABLE (#PCDATA)>
<!ELEMENT RANGE (#PCDATA)>
<!ELEMENT BODY (FOREACH?,INSTRUCTION+)>
<!ELEMENT INSTRUCTION (PARAMETER+,NODE?)> <!-- Each Instruction is a
       service. It has a name, input and output parameters, and
       possibly a processing node to which it has been assigned -->
<!ATTLIST INSTRUCTION SERVICE CDATA #REQUIRED>
<!ELEMENT NODE (#PCDATA)>
<!ELEMENT PARAMETER (#PCDATA)>
<!ATTLIST PARAMETER TYPE CDATA #REQUIRED>
<!ATTLIST PARAMETER NAME CDATA #REQUIRED>
```

Fig. 4. Document Template Definition (DTD) for the prototype DISCWorld Distributed Job Placement Language (DJPL). The DTD allows DJPL scripts to be syntactically verified at run-time. There are no DISCWorld-dependent features in the language's definition in order for it to be used by other systems.

```
<?xml version="1.0"?>
<!DOCTYPE DJPL SYSTEM "djpl.dtd">
<DJPL NAME="myAdder">
  <USER ID="heath"> <GROUP>dgis</GROUP>
    <PERMISSION>unrestricted</PERMISSION> </USER>
  <JOB ID="00005" PRIORITY="5">
    <COST> <SOFT>100</SOFT> <MAX>120</MAX>
      <ESTIMATE>90</ESTIMATE> </COST> <TIME>360</TIME>
    <SERVER> <NAME>geronimo.cs.adelaide.edu.au</NAME>
      <PORT>6668</PORT> </SERVER> </JOB>
  <INPUTS></INPUTS> <OUTPUTS>SUMS</OUTPUTS>
  <INSTRUCTION SERVICE="generalAdder">
    <PARAMETER TYPE="INPUT" NAME="LHS">Integer:2</PARAMETER>
    <PARAMETER TYPE="INPUT" NAME="RHS">Integer:3</PARAMETER>
    <PARAMETER TYPE="OUTPUT" NAME="SUMS">
      generalAdder:SUMS(Integer:2,Integer:3)</PARAMETER>
  </INSTRUCTION>
  <INSTRUCTION SERVICE="generalAdder">
    <PARAMETER TYPE="INPUT" NAME="LHS">Integer:4</PARAMETER>
    <PARAMETER TYPE="INPUT" NAME="RHS">Integer:5</PARAMETER>
    <PARAMETER TYPE="OUTPUT" NAME="SUMS">
      generalAdder:SUMS(Integer:4,Integer:5)</PARAMETER>
  </INSTRUCTION>
  <INSTRUCTION SERVICE="generalAdder">
    <PARAMETER TYPE="INPUT" NAME="LHS"></PARAMETER>
    <PARAMETER TYPE="INPUT" NAME="RHS"></PARAMETER>
    <PARAMETER TYPE="OUTPUT" NAME="SUMS">
      generalAdder:SUMS(generalAdder:SUMS(Integer:2,Integer:3),
                        generalAdder:SUMS(Integer:4,Integer:5))</PARAMETER>
  </INSTRUCTION>
</DJPL>
```

Fig. 5. Example DJPL code. The instruction section of the code represents the user's processing request as a DAG. Data is shared between instructions by the use of its name. Because services can have multiple outputs, each output has a different name; the data produced has the specific name as part of its "recipe".

```
<?xml version="1.0"?>
<!DOCTYPE DJPL SYSTEM "djpl.dtd">
<DJPL NAME="generalAdder">
  <INPUTS>LHS, RHS</INPUTS>
  <OUTPUTS>OUTPUT</OUTPUTS>
  <INSTRUCTION SERVICE="addSomething">
    <PARAMETER TYPE="INPUT" NAME="input_lhs">LHS</PARAMETER>
    <PARAMETER TYPE="INPUT" NAME="input_rhs">RHS</PARAMETER>
    <PARAMETER TYPE="OUTPUT" NAME="sum">addSomething:sum(LHS,RHS)</PARAMETER>
  </INSTRUCTION>
</DJPL>
```

Fig. 6. DJPL script representing a DISCWorld compound (or meta-) service. The service represented by this script can be substituted for a single service that matches the same description and has the same number of inputs/outputs.

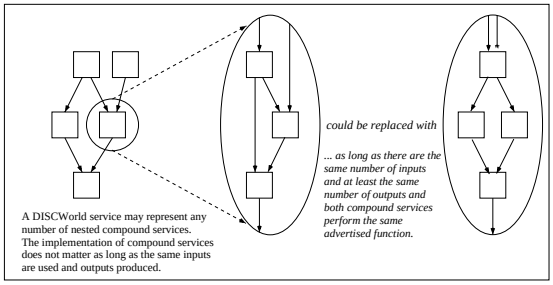

Fig. 7. Services with the same name need not necessarily be implemented in exactly the same way. The circled services are interchangeable iff they advertise the same purpose, accept the same number of inputs, and produce at least the same number of outputs as the required service.

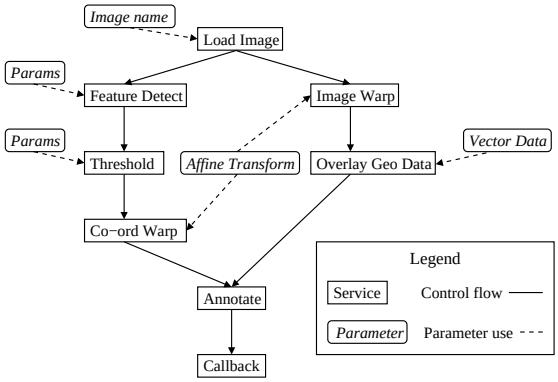

Fig. 8. A workflow imagery exploitation and analysis example which has been implemented within the DISCWorld framework. Each part of the process can be represented by a DISCWorld service; links between services represent data flow. Intermediate data products can be stored for re-use in future processing requests.

The analysis script performs a number of quality-checking routines on the data and then passes the data to two separate forks in the request tree. One fork performs a simple feature detection algorithm on the image, creating annotations that can be overlayed on the image, while the other fork transforms the image to a common spatial coordinate system so that geospatial data stored in a database can be overlayed. This may involve the translation and rotation of the image. Once the feature detection and transformation processes are complete, the annotations produced by the detection algorithm are transformed and overlayed onto the resultant image. This process is shown in figure 8, where it is clear that the two arms of the fork can be executed in parallel.

This application was based on the United States National Imagery and Mapping Agency (NIMA) Geospatial and Imagery eXpoitation Service specification (GIXS) [31]. The specification is published as CORBA [23] Interface Definition Language (IDL). In conjunction with Australia's Defence Science and Technology Organisation we have adapted this specification in order to implement a federated version using the DISCWorld workflow environment. Java's Advanced Imagery (JAI) [29] application programmer interface (API) is used to supply the imagery exploitation services. A GIAS-compliant [30] imagery archive [8] is accessed through a CORBA interface.

V. Workflow Architecture Issues

The problem of providing a high-performance, reliable, distributed workflow management system is not easy. There are a number of systems that provide partial solutions to some of these problems, but no one system presents a complete solution. DISCWorld is a research project, in which partial solutions to each of these aspects are addressed.

The problem of (high) performance is tied with efficiency of the services. For a single service this can be done by placing it on a computer that is most closely matched to the service's execution characteristics; when many services and a large amount of data must be matched to a wide array of possible computers, the problem of service placement and execution scheduling becomes non-trivial [21]. We believe the use of DRAMs and meta-services reduces the sheer complexity of the problem by introducing some levels of abstraction.

Reliability in a distributed system is an important issue even when the pool of available computational resources is under local control. As soon as the user (or the user's IT manager, for example) runs their program across a set of physically distributed computational resources, the probability of a single machine failing, due to hardware failure or network failure, becomes significant. We address this problem in three ways: by creating redundancy wherever possible in the system, for example, replicating services across different machines; by federating the control and execution mechanisms across each server in the DISCWorld; and by canonically naming derived data and DRAMs with the "recipe" needed for reconstructing them. This ensures that when a server fails (or becomes unavailable), the most no-

ticeable effect of "missing" derived data is the latency caused by reconstructing it on the fly.

DISCWorld relies on two basic tenets in order to function successfully in the distributed environment: DISCWorld servers are peer-based; and the global naming scheme allows request optimisation. Having a peer-based system allows the system to function more-or-less normally even in the presence of massive machine failures because all the control and job scheduling mechanisms for each machine are local.

A significant problem in the wide-area distributed computing (or grid computing) community is the lack of proper standards. For example, many research groups around the world are creating good, but piecemeal, solutions to the problems of describing both remote data and complex processing tasks. The lack of standards means that in many cases these efforts will be replicated, often with the net effect that the technologies will be incompatible. In addition, different research groups are approaching the problems from vastly different perspectives. For example, the approach taken by the Globus/Nexus [9] and Legion [13] projects is that of a single program which is able to use distributed resources, and the underlying software provides such services as: security, remote data access, and scheduling. This is in direct contrast to the DISCWorld which, as previously described in section II, is a service-based approach to grid computing.

VI. Summary and Future Work

Workflow applications are proving to be an extremely versatile metaphor for describing real-world processes. This metaphor allows one to define an application in terms of smaller, manageable, components that themselves may be further broken down. We find that for a certain class of applications, the workflow metaphor has advantages in the process of specification and managing complex tasks, particularly when using distributed computing.

We have introduced a distributed computing environment, DISCWorld, which allows users to compose a complex processing request consisting of a directed acyclic graph of high-level services. These services are connected by the sharing of data in a system that is able to span a high-latency computational environment. Furthermore, we have introduced the technologies we have developed to express and implement the workflow metaphor. We have developed a high-level pointer mechanism, DRAMs, which we can use to make intelligent scheduling and placement decisions on large distributed data items and services. We have also introduced a general, yet powerful, scripting language called DJPL that is used to encapsulate and describe a user's processing request for use between machines in the distributed system. Although the DRAM and DJPL mechanisms have been developed within the context of the DISCWorld environment they are general enough to be used within any distributed system that features a canonical naming strategy for objects.

We have described one of the applications we have successfully implemented using the workflow metaphor within the DISCWorld environment, and we have discussed a number of key issues for implementing a high-performance, reliable, distributed workflow management system.

Key areas we are addressing in our current work, among which are those of DISCWorld policies, user and data security mechanisms, processing request exception handling and performance monitoring tools. We wish to express policies, such as the inclusion or exclusion of a user, group of users, services, and data, on either a local or global scale. The problem is one of a suitable language for expression and a logic engine for the enforcement of such policies. In addition, careful checks must be made to ensure consistency between policies.

The current DISCWorld prototype has little in terms of user and data security. We are investigating the use of LDAP [19] in order to implement a distributed user namespace mechanism. Data security can be quite simple – one such scheme may rely on a user's membership of a hierarchically-arranged group structure, and having data (both raw and derived) belong to that group. The complication becomes evident when members from different, unrelated groups wish to share data. Furthermore, the problem of service namespace management is not trivial, especially when users are able to add their own services into the DISCWorld environment's namespace.

The current mechanism by which exceptions are reported is not very robust: the local DISCWorld daemon recognises that a service has stopped abnormally and sends a message back to the user client for rectification. Any resources which had been reserved by virtue of a DRAMF being created are not released – there is no need as DRAMFs are not actively consuming processor cycles. We wish to implement two possible improvements to this: firstly to define a hierarchy of exceptions with default rectification behaviour; and secondly to further annotate the user DJPL script with sufficient information to

express the appropriate behaviour *for that processing request.*

References

[1] Paul W. Abrahams, James H. Andrews, Jim Blandy, Robert J. Chassell, Daniel J. Edwards, Timothy P. Hart, Brian Harvey, Michael I. Levin, John McCarthy, and Jim Veitch. *Handbook of Programming Languages Vol. IV Functional and Logic Programming Languages.* Macmillan Technical Publishing, IN, USA, 1998. ISBN 1-57870-011-6.

[2] Advanced Visual Systems (AVS). *AVS Developer's Guide.* Advanced Visual Systems Inc, release 4 edition, May 1992.

[3] Walt Brainerd, Ron Cytron, Ralph E. Griswold, Glenn Grotzinger, Dennis M. Ritchie, and Steve Summit. *Handbook of Programming Languages Vol. II Imperative Programming Languages.* Macmillan Technical Publishing, IN, USA, 1998. ISBN 1-57870-009-4.

[4] K. Mani Chandy and Carl Kesselman. CC++: A Declarative Concurrent Object-Oriented Programming Notation. Technical report, Caltech, 1993. MIT Press.

[5] P. D. Coddington, G. Hamlyn, K. A. Hawick, J. F. Hercus, H. A. James, D. Uksi and D. Weber. A Software Infrastructure for Federated Geospatial Image Exploitation Services. Technical Report DHPC-092, Distributed and High Performance Computing Group, Department of Computer Science, University of Adelaide, June 2000.

[6] P. D. Coddington, K. A. Hawick, and H. A. James. Web-Based Access to Distributed, High-Performance Geographic Information Systems for Decision Support. Technical Report DHPC-037, Distributed and High Performance Computing Group, Department of Computer Science, University of Adelaide, June 1998.

[7] P. D. Coddington, K. A. Hawick and H. A. James. Web-based Access to Geospatial Image Archives. Technical Report DHPC-089, Distributed and High Performance Computing Group, Department of Computer Science, University of Adelaide, May 2000.

[8] P. D. Coddington, K. A. Hawick, K. E. Kerry, J. A. Mathew, A. J. Silis, D. L. Webb, P. J. Whitbread, C. G. Irving, M. W. Grigg, R. Jana, and K. Tang. Implementation of a Geospatial Imagery Digital Library using Java and CORBA. In *Proc. Technologies of Object-Oriented Languages and Systems Asia (TOOLS 27).* IEEE, September 1998.

[9] Ian Foster and Carl Kesselman. Globus: A metacomputing infra-structure toolkit. *Int. J. Supercomputer Applications*, 1996.

[10] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, Inc., 1999. ISBN 1-55860-475-8.

[11] D.Georgakopoulos, M.Hornick, A.Sheth. An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure. Distributed and Parallel Databases, 3:119–153, 1995.

[12] Mudit Goel. Process Networks in Ptolemy II. Master's thesis, University of California, Berkeley, December 1998.

[13] Andrew S. Grimshaw and Wm. A. Wulf and the Legion team. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1), January 1997.

[14] K. A. Hawick and H. A. James. A Distributed Job Placement Language. Technical Report DHPC-070, Department of Computer Science, The University of Adelaide, May 1999.

[15] K. A. Hawick and H. A. James. A Java-Based Parallel Programming Support Environment. In *Proc. High Performance Computing and Networking (HPCN) Europe 2000*, volume 1823 of *Lecture Notes in Computer Science*, pages 363–372. Springer-Verlag Berlin Heidelburg, May 2000.

[16] K. A. Hawick, H. A. James, and J. A. Mathew. Remote Data Access in Distributed Object-Oriented Middleware. *To appear in Parallel and Distributed Computing Practices*, 2000.

[17] K. A. Hawick, H. A. James, A. J. Silis, D. A. Grove, K. E. Kerry, J. A. Mathew, P. D. Coddington, C. J. Patten, J. F. Hercus, and F. A. Vaughan. DISCWorld: An Environment for Service-Based Metacomputing. *Future Generation Computing Systems (FGCS)*, 15:623–635, 1999.

[18] International Business Machines Corporation. XML Parser for Java v3.0.1. Available from http://www.-alphaWorks.ibm.com/, last visited May 2000.

[19] Internet Engineering Task Force (IETF). Lightweight Directory Access Protocol (LDAP) Specification version 3. Released 1997.

[20] H. A. James and K. A. Hawick. Data futures in DISCWorld. In *Proc. High Performance Computing and Networking (HPCN) Europe 2000*, volume 1823 of *Lecture Notes in Computer Science*, pages 41–50. Springer-Verlag Berlin Heidelburg, May 2000.

[21] Heath A. James. *Scheduling in Metacomputing Systems.* PhD thesis, Department of Computer Science, The University of Adelaide, July 1999.

[22] Khoral Research, Inc. Khorus. Available at http://-www.khoral.com, last visited May 2000.

[23] Object Management Group (OMG). The Common Object Request Broker: Architecture and Specification (revision 2.0). Framingham, MA, July 1995.

[24] Object Management Group (OMG). CORBA/IIOP 2.2 specification. Available from http://www.omg.org/-corba/cichpter.html, July 1998.

[25] John A. Sharp. *Data Flow Computing.* Ellis Horwood Limited, 1985. ISBN 0-85312-724-7.

[26] Andrew Silis and K. A. Hawick. The DISCWorld Peer-To-Peer Architecture. In *Proc. Fifth IDEA Workshop*, February 1998.

[27] Larry Smarr and Charles E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.

[28] Sun Microsystems. Java products homepage. Available from http://www.javasoft.com/products/.

[29] Sun Microsystems. Java products homepage. Available from http://java.sun.com/products/java-media/jai/.

[30] United States National Imagery and Mapping Agency (NIMA). USIGS Geospatial and Imagery Access Service (GIAS) Specification. Version 3.1, February 1998.

[31] United States National Imagery and Mapping Agency (NIMA). USIGS Geospatial and Imagery Exploitation Service (GIXS) Specification. Version 2.0, June 1999.

[32] Derek Weber and Heath James. A Federated GIXS Model. Technical report DHPC-088, Defence Science and Technology Organisation (DSTO) and The University of Adelaide, February 2000.

[33] World Wide Web Consortium (W3C). eXtensible Markup Language (XML). Available at http://www.-w3.org/XML/, Last visited May 2000.