A Reconfigurable Component-based Problem Solving Environment

K.A. Hawick†, H.A. James† 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—
Problem solving environments are an attractive approach to the integration of calculation and management tools for various scientific and engineering applications. These applications often require high performance computing components in order to be computationally feasible. It is therefore a challenge to construct integration technology, suitable for problem solving environments, that allows both flexibility as well as the embedding of parallel and high performance computing systems. Our DISCWorld system is designed to meet these needs and provides a Java-based middleware to integrate component applications across wide-area networks. Key features of our design are the abilities to: access remotely stored data; compose complex processing requests either graphically or through a scripting language; execute components on heterogeneous and remote platforms; reconfigure task subgraphs to run across multiple servers. Operators in task graphs can be slow (but portable) "pure Java" implementations or wrappers to fast (platform specific) supercomputer implementations.

Keywords: problem solving environments; DISCWorld; high performance computing; grid computing; distributed data; Java; technology integration.

I. Introduction

Problem Solving Environments (PSEs) provide a way of integrating complex computing technologies so that users need not be computer specialists. Collaborative PSEs (CPSEs) extend this goal so that groups of users can work together across distributed computing systems without needing to be aware of the system details. Users can focus on the application domain expertise needed to solve their problem while the distributed computing "glue" or middleware that makes the system work can remain transparent to them. This goal of transparency is however rather difficult to achieve and this is particularly so when a degree of change in the computer system configuration is required for problem domains. This change or "reconfiguration" in the system is required in all but the simplest of real world applications. Application scenarios illustrating the need for reconfiguration in CPSE are described in section II.

Reconfiguration will typically involve changes to the collaborating users and program sequences being run in a CPSE. The running system must be able to handle these changes – which may be unpredictable – gracefully and should not leave broken pointers or references to objects (programs, data or users) that have changed. Our solution to the reconfiguration problem in CPSE involves a software architecture and a prototype implementation for a distributed computing middleware system known as DISCWorld. We have designed two critical mechanisms in our DISCWorld system for managing remote data and services in a data flow model as well as a "futures" mechanism for managing data and services that exist only *in potentia* when the request for them is first made. We have found these mechanisms provide a model capable of robustness and stability in a changing runtime environment and also allow exploitation of parallelism in complex jobs.

A key feature of our architecture is the recognition that the system can be managed in terms of data processing operators with data items flowing between them. In particular a human user interacting with the system can be modelled as a particular sort of operator with extreme latency and reliability properties. Following the standard task graph approach of dataflow systems, we model applications in terms of Directed Acyclic Graphs (DAGs) of operators at the nodes and simple data flow along the connecting edges. We describe the essentials of this reconfigurable DAG model in section III. DISCWorld is outlined in section IV where we explain the DISCWorld Remote Access Mechanism (DRAM) and DRAM Futures (DRAMF) mechanisms and how these embody the operator/data control ideas and their expression as DAGs.

We have built a number of prototype CPSEs using our

technology, mostly for decision support applications involving access to distributed archives of geospatial data such as aerial photography, satellite images and mapping data. However we have focussed mainly on developing the server-side technology to support CPSEs, since we believe that determining the fundamental architectural principles and interface specifications will be critical to genuinely wide-area, open, and scalable CPSEs. An objective of our work is to thoroughly explore the technical issues necessary to propose draft standards for interfacing between distributed components of CPSEs.

The idea behind users collaborating through shared access to programs, data and each other has been prevalent for some years [3]. Users on a networked computer system have found ways to interact using *ad hoc* mixes of computer technologies. These include email; shared networked file systems; remote logins to other users' computers and more recently the use of desktop video conferencing facilities. Difficulties have arisen in the growing complexity of operations and data that users wish to share. Particular obstacles have been the plethora of different data formats even for simple data exchange; the lack of portability of programs across different computer platforms and the lack of interoperability between programs. Research into CPSEs and their supporting technology will hopefully open up many new and significantly more complex interactions between users across disciplines.

II. CPSE Application Scenarios

In this section we describe some of the application areas we have considered and how they benefit from our approach to Collaborative Problem Solving Environments (CPSE).

CPSEs are a much sought after solution to the needs of human decision makers to share data, resources and ideas. Such decision support systems are very important for a wide variety of organisations, in the commercial, government and defense sectors. A classic example is a defense command and control system that will involve collaboration between many decision makers in different geographical locations, and data inputs from many sources, including real-time data from the field of operations as well as data from a number of distributed data archives. In the government sector, emergency services command centres have a similar functionality and require similar collaborative decision support systems. Other examples include areas such as environmental analysis and modeling and land use planning, which may require data and specialised analysis from many different government departments.

We have developed some prototype CPSEs for both defense command and control [6, 8] and emergency services response [7, 18] systems. These are technology demonstrators that illustrate how powerful Java-based object-oriented CPSEs can be developed using the DISCWorld grid computing architecture. Our applications have focussed on the use of geospatial (i.e. spatially referenced) data such as maps, satellite images, aerial photography, elevation, vegetation cover, etc. A typical decision support application may require large amounts of data from a number of distributed geospatial data archives. We have worked with the Australian Defence Science and Technology Organisation (DSTO) in building systems for geospatial imagery analysis and exploitation [6,8]. A fundamental aspect of defense command and control systems is the requirement for geospatial data such as maps and images from earth observation systems. This requires a collaborative environment linking image data archives, image processing applications and compute servers, specialised image analysts, and the decision makers who use the image data. In many situations the image analyst or the decision maker may want on-demand access to an image for a specific region, and may also request that some specialised image processing, such as feature detection or map overlay, be done on the image. We have developed a prototype software environment that supports on-line distributed data access and processing across a federated collection of geospatial image archives and image processing services.

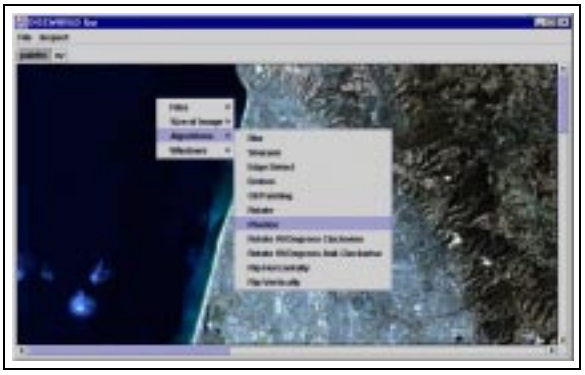

Fig. 1. This figure shows a prototype image analysis and processing tool that enables users to edit and annotate a geospatial image in a collaborative setting. All users see the results of the changes, and they can add their own annotations or comments to the collaborative effort. The image, along with the annotation transcript, can be saved and replayed for later viewing by a decision-maker.

Figure 1 shows a simple demonstrator system we have built to allow catalog access, process control and shared access to annotations and other collaboratory information. The architecture underpinning this demonstration is described in sections III (basic reconfigurable dataflow model) and IV (software implementation). We focus on the problems of manipulating shared imagery in CPSEs for the purposes of illustration in this paper, but the data and operators shared in our architecture are not limited to this area.

Another demonstrator we have developed is a prototype system for geospatial data collection, visualisation and processing to support emergency services response [7,18]. The example application we have targeted is response to a wildfire emergency. This requires coor-

dination of a large team of people including firefighters, police and paramedics at the scene and an emergency services response unit at a central command centre.

Our prototype system is based on a Web-based Java applet interface connected to a server that handles input from multiple data sources and manages state information so that all users can see the same information on screen via the applet interface (this is similar to the approach of the Habanero [24] and TANGO [25] systems for Web-based collaboration). For example, airborne imaging systems can transfer information on the extent of the fire front; police and firefighters in the field (who may be working on a laptop with a modem connection via a mobile phone) can add information on current positions of fire trucks and roadblocks; decision makers in the command center can provide recommendations on where to attack the fire or set up evacuation points. Our system allows users to overlay many types of geospatial information including roads, terrain elevation, vegetation types, satellite images and aerial photography, positions of emergency services personnel and equipment, and the extent of the fire front. It also allows users to access data processing services, including the ability to run fire simulation models to estimate the spread of the fire, least path algorithms to suggest optimal routes for emergency services vehicles, and algorithms that use roadmap and population density information to suggest evacuation strategies.

We are presently considering other application areas such as those in collaborative science and engineering where we can provide a collection of component operators and can experiment with how collaborative users can best deploy them. Typically users wish to steer large and complex simulations or calculations; visualise complex datasets; or carry our parameter searches, coordinating the results of many runs or jobs. A CPSE which allows both graphical and scripting control is a good solution to this problem.

Figure 2 shows a fundamental idea behind our approach. Collaborating users can combine reconfigurable component operators into processing tasks, sharing data, operator codes and results with one another. Users can compose component operators graphically, but the system uses a scripting language representation of the graphical composition so that it may manipulate, decompose and schedule parts of the computation.

A key issue which has emerged from all these application areas is the need to handle complexity and change in a running system. We believe a number of proprietary systems have provided superficial solutions to the graphical CPSE problem but none have based their system on a well-considered architectural model capable of supporting reconfiguration.

III. Reconfigurable DAGs

We model all user processing requests in our system with Directed Acyclic Graphs (DAGs). A DAG is a

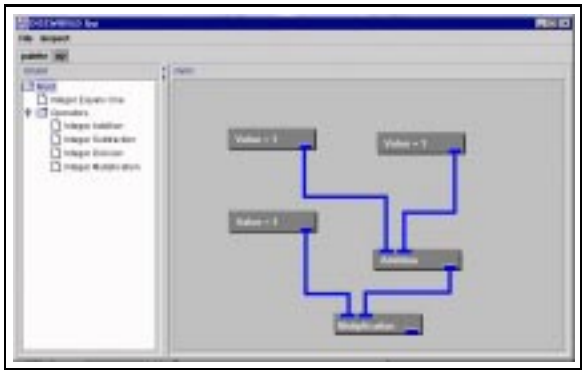

Fig. 2. Simple graphical operator composition. Available operators are listed in the palette on the left-hand side of the display. Users select the appropriate operators by dragging them onto the canas at the right-hand side of the screen. When operators are released onto the canvas they are drawn in boxes with smaller rectangles representing the inputs they require and all the outputs they produce. Operators are joined by the user; the joins represent data flow between operators. When executed operators may be placed onto different machines in the distributed system.

graph consisting of nodes (operators) and vertices (data passed between operators). It is directed, meaning that data flows in one direction only. Furthermore, the graph is acyclic, prohibiting the presence of cycles within the structure. This allows us to treat the data items flowing between operators simply; we do not have to use any advanced dataflow mechanisms such as token colouring to distinguish between data originating from different iterations through the same graph [31]. The Directed Acyclic Graph (DAG) model we employ for expressing combinations of program components is a simple but powerful model. It is used in a number of other systems such as Java's Advanced Imaging (JAI) [32] toolkit and also some standards interfaces for distributed imagery access and exploitation [35,36]. It allows us a well-defined mechanism for configuring and reconfiguring operators at runtime. This is particularly useful when an operator may be an interactive program component – representing the "human in the loop" in a CPSE.

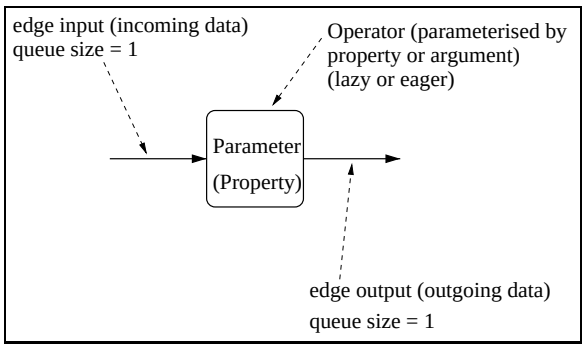

Fig. 3. Parameterised operator showing data flow along edges for input or output.

Figure 3 shows a parameterised operator component as employed in our model architecture. Operators have none

or more inputs and one or more outputs. Parameters can be raw data, in which case they have no "producing operator" or derived, in which case they do. An input can be modelled as either an edge or as a parameter. For the purposes of illustration we use both to explain our system. The embedding context for operators can supply parameters and in a sense these are the same as special parameter-type input edges. This is shown again in figure 4 a) and b) where an equivalent DAG can be set up with two special input edges and two parameters or with four input edges and no parameters. This point is important for considering humans interacting with components. Humans will typically interact with a GUI component and will typically input parameters by typing or clicking a mouse or through some other input device. Our point here is that while the front end environment will have to provide the necessary support infrastructure for these input parameters, we can still model the dataflow process as though there were no parameters and just simple data flow edges.

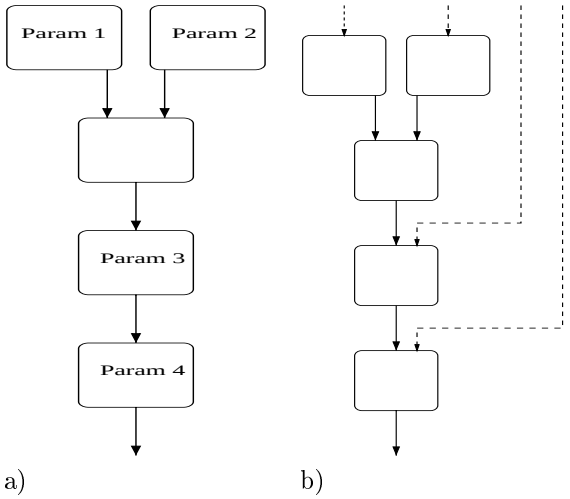

Fig. 4. a) Parameterised DAG using single edge type and b) Equivalent Unparameterised DAG using differently typed edges.

Operators could be relatively simple program components such as: read an image from store; display an image; crop an image producing a new one; superpose two images producing a third; split an image into high and low pixel intensity values producing two new images. Alternatively operators could be entire complex programs involving a large set of inputs and/ or outputs. We describe how we implement operators either as reusable (Java) components or as (Java) wrappers to complete programs in section IV.

There are a number of component primitives that can be used to model the various activities in the CPSE. We can implement base component classes for the five principle of these: source; sink; filter; fork/split; and merge/join. In a simple system where we employ only one base level data type such as an all encompassing base image class, it is very simple to construct DAGs from operators based on these five primitives. Any output can feed onto any output since the data type is the same. It is surprising how many simple but useful systems can be built on this assumption. Allowing different data types makes the model considerably richer, but it becomes more difficult to establish rules for interconnecting operators to ensure type safety.

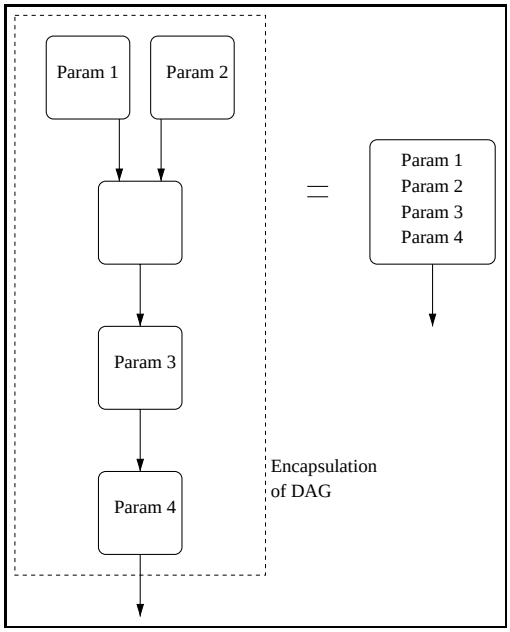

Fig. 5. Rubber Banding to encapsulate a DAG.

Systems like AVS [2] provide a graphical solution to the type connectivity problem. The front end GUI colour codes the different data types present in a particular graph and enforces connections only between matching colours. This solves the problem partially for relatively simple systems but not for examples where two inputs to an operator are of the same type but have different semantics. There are considerable arguments one can get into about how rigorously the type system needs to be defined in such cases. For our experimental system we do not claim to have solved this problem. This underpins one of the key issues for wide-area CPSEs however. It is necessary to decide how users and their systems will exchange both data and operator components that they did not write. There is a clear need for some interface standards for such components if CPSE users are to have the freedom to access and exchange libraries of component services. This issue is discussed further in section VI.

Interactive users of a CPSE will construct complex operations or queries to their system consisting of a choice of a palette of operators drawn from libraries belonging to themselves or others. Interactions and changes in the system environment due to users' own inputs or those of others in a CPSE will require the system to be dynamically reconfigured. This reconfiguration is needed to support the need to change and allow late change of the DAGs which express the computations [20]. Figure 5 illustrates one of the important concepts in our system

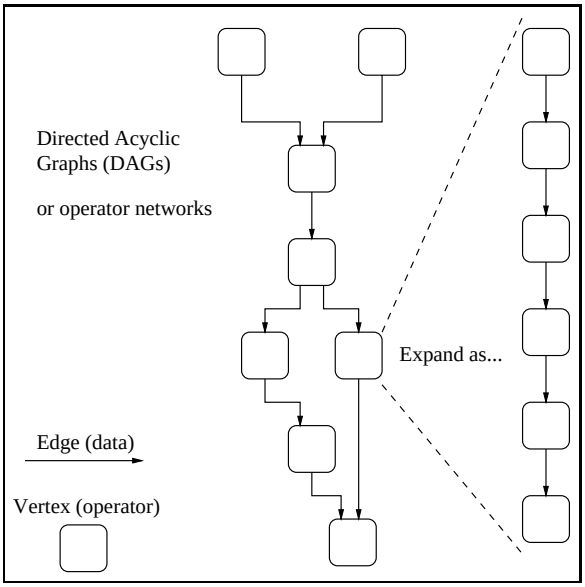

Fig. 6. DAGs can contain compound services, which are expanded at run-time. This is possible due to the lazy evaluation of services in the DISCWorld environment. Compound services are represented by their own self-contained sub-graph, which can in turn contain other compound services.

whereby an entire DAG can be collapsed down to a single sub-graph or indeed down to a single operator, in this case a four-input or four parameter operator.

The collapse of DAGs in this way supports the idea of combining useful operators or DAGs into reusable components. An underlying issue is how operators and DAGs are named. Naming and binding to data and operators is a difficult problem. We have made some progress using a simple name space where every data item and component is assumed to have a unique name. Implementation of this assumption is generating interesting research as there are various possible algorithms and models to enable this [10].

Collapsing the sub-graphs into operators is important to allow users to work on large scale and interesting problems and to reuse the components previously constructed by them or their collaborators or available in libraries or databases. We have experimented with the concept of a "CodeServer" [23] which can be remotely queried using metadata and which will deliver a suitable operator packaged as executable Java bytecode. This allows programs as well as data to be easily migrated around a wide-area network. This can be very useful for sharing modules in collaborative work, and also for PSEs that involve processing of large distributed data sets, where it is more efficient to move the program to the data rather than the data to the program. The main issue underpinning the success of such a system is how to treat and standardise the metadata describing coded operators.

Figure 6 shows the converse idea of expanding an operator or part of a graph into a sub-graph in its own right. These collapsing and expanding ideas are vital for producing a powerful reconfigurable system. Expanding operators is useful to decompose complex operators into parts that can be easily farmed out to achieve parallel execution for performance reasons, or indeed to allow decisions to be made by different computers or users in the collaboration network as to how a job should actually be scheduled. This latter point is vital for both scalability and reconfiguration reasons in a distributed system.

One reason for using CPSEs is to be able to exploit and share remote resources that may be owned by collaborators. These are often specialist or high-performance resources and access to such scarce resources is often a major driving force behind collaboration. It is therefore evident that efficiency and scalability of the underlying distributed computing software infrastructure are also important issues for a CPSE.

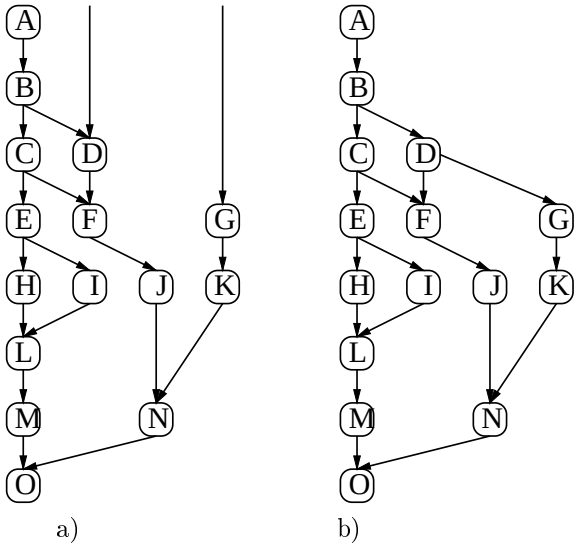

Fig. 7. Two different orderings of execution of the operators in a DAG

Figure 7 shows how a DAG consisting of many operators can be executed in different orders – lazily or eagerly – resulting in different execution profiles. The flexibility to allow this sort of reconfiguration is particularly important in a system involving human interacting components where it may be important that other components execute as independently as possible of a slow and unpredictable human. The flexible execution profile allows system control components to make sensible scheduling decisions [21] about when and where to execute operators.

Scheduling is important for accessing data resources as well as computational ones. An application area we have studied at length involves access to satellite imagery and bulk data sets that may be too expensive or simply too large to store in their entirety at more than one or two locations. These data sets might be accessible only via tape robots or in some cases through human operators who are involved in some way in loading and unloading tape sets. In a sense the tape or resource operator is a human operator in the CPSE. We are investigating how

storage services can be managed separately in a transparent fashion from other computational aspects of CPSEs. Our approach is to use file system software technology to make remote data and hierarchical storage systems such as cached tapes and tape robots appear as remote file systems or databases [28].

In this section we have discussed the model on which our architectural approach is based. We now discuss how we have implemented this model in a prototype of our DISCWorld system.

IV. DISCWorld - An Outline

Our Distributed Information Systems Control World (DISCWorld) project was started in 1996 with the long term goals of researching the underpinning issues for a metacomputing system. Recently the popular term "Computational Grid" [12] has superceded that of metacomputer but the ideas remain the same – providing a high-level middleware system that enables transparent access to high-end compute servers and data storage distributed over a wide-area network. We aim to further explore the algorithmic and systems issues involved in building, running and maintaining such systems, which form the basis of PSEs. CPSEs extend the goal to cope with more interactive multi-user systems, but as explained above we believe that interacting users can be modelled as operator components in our system.

The key to a working system is the software design of the DISCWorld servers, the protocols they use to communicate and the interoperability mechanisms with legacy applications and data systems. We have approached the problem by placing a DISCWorld daemon on every server that participates in a DISCWorld community. The software for the daemon is intended to be a minimal framework for loading up customised modules for specialist operations. These may be Java [33] byte code or wrappers to specialist native platform codes. The design is motivated by a need for scalability to a large number of cooperating nodes, with as little central decision making as possible. The local administrators or owners of a resource participating in a DISCWorld can decide what services they will offer by setting various policy and environment information in its local database.

DISCWorld acts essentially as a software glue, providing interoperability between new and existing services. Figure 8 illustrates the key ideas behind the DISCWorld daemon "DWd". A dual network is present between all servers in principle. A control network is used to communicate small lightweight messages that specify how servers are interacting and cooperating, and a bulk data transfer mechanism is used to transfer data products and partial products between nodes. In practice these mechanisms may be implemented on the same hardware network, or on separate networks when available. The control information is lighter but needs a reliable network, whereas the bulk data needs high bandwidth but can always be retransmitted if a network is blocked. Daemons are used to provide all the necessary control information to wrap up legacy codes and systems as well as being a framework for interfacing to new codes. We have also built a file-system based mechanism to provide the support services necessary to support legacy applications for which we do not have source code access [27]. Software adapters are used to bridge firewalls as well as interface to other metacomputing software systems.

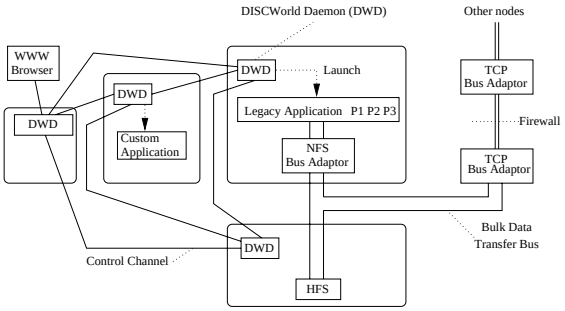

Fig. 8. DISCWorld Software Architecture Overview

The daemon's operations are best explained by a discussion of the sequence of events that take place when a client contacts it. A user query is posed to the daemon (through a CPSE front end), in the form of a well defined request for some computation to be performed on some data (using the DAG/operator model). The request must be mapped onto the available resources (operators mapped to particular compute systems) that the daemon knows about and even though the daemons are peer-based, the daemon in question acts as a manager for the purpose of brokering this request. Management of the request consists of monitoring and supervising the ongoing request once it has been initiated. The daemon must also provide the user with an ability to revoke the request should it be necessary. The daemon is also responsible for brokering the safe delivery of the results to the user, or temporary storage of the results and notification of the user. It can also organise the collection and storage of performance data for the purposes of predicting future performance and costing.

The daemon is initiated as a program by the system administrator or as a system daemon and, once initiated, it reads some local state information from a persistent store (e.g. database) to configure itself and subsequently listens to a particular port for user requests. Each daemon has a "gossip" mechanism to allow it to determine what services and hosts are available to it. Each daemon stores its own internal database of servers offering component services as well as their associated characteristics, and maintains a priorities list according to some locally defined policy. The daemon is highly configurable, and can be customised to reflect the local server/services it provides to other DISCWorld daemons. It also has a configurable policy for where to seek assistance from its peers in satisfying incoming user queries.

DISCWorld was primarily motivated to deal with relatively large computations that need to be carried out

on large datasets, and to allow use of distributed high-end computing, communications and storage resources to execute these problems in a timely fashion. This is particularly useful for decision support systems and CPSEs, which generally require timely access and processing of multiple distributed data sources.

The major services/servers that each DISCWorld daemon knows about were originally substantive programs running on high-performance platforms. Nevertheless, the control information and "glueware" needed to build such a system are complex but not so demanding of computational performance. We have based the development of our CPSEs on the DISCWorld environment, which has been written in Java. Whereas up until now we have mainly implemented operators as Java wrappers around high-performance legacy code, the rapid improvement in Java Virtual Machine performance has prompted us to build more component services using pure Java. A Java implementation allows for a number of capabilities that can be readily implemented using the Remote Method Invocation (RMI) Java package.

The multi-threaded mechanisms in Java allow the daemon to be implemented with a prioritised set of threads to handle the user and server-server interactions. The use of Java threadgroups coupled with the Java security manager mechanism enables threads to be placed into groups with controlled access between the groups. This enables the daemon control threads and groups of user threads to be kept apart and scheduled separately within the daemon. The daemons communicate via typed messages. These are received at a central point within each daemon, allowing messages to be routed to the appropriate module. Some of these modules within our current daemon prototype, include a server/services database manager, a system administration console manager, a computation organiser and a local execution manager. These only accept control level messages from the central point, although the DISCWorld model allows for a high bandwidth direct communication channel between modules. Control messages contain instructions for a single module, and if modules within the same daemon wish to communicate they must sent messages via the central point.

The use of the Java object serialisation and dynamic loading and binding allows arbitrary data structures to be passed between both servers and clients in the DISCWorld. The code for these data structures may be loaded dynamically. This includes operations on the data structure which means that the same data structure can have different method implementations depending on the data contained within – thus allowing polymorphic operations. At present, serialisation is limited by the need for both communicating parties to use Java, which requires Java wrappers around non-Java code. Wrappers can be done using the Java Native Interface (JNI) API. We believe the additional effort of implementing Java wrappers is outweighed by the ease with which objects with state can be stored and communicated.

Using Java coupled with native methods to embed other programs does not mean that the system itself is no longer portable. The glue that controls the system remains portable although the native method services will not be portable. This is in keeping with our intention that a particular daemon should be configurable by the local administrator to reflect the locally available server/services. In fact, a limited portability can be achieved by careful loading of the native library for some applications. The library that can be loaded can be dependent of the architecture and operating system version, which information can be made available to the daemon as part of its configuration. This is not completely dependable, however, and it is an attractive possibility to find a mechanism whereby a growing collection of application components can be maintained as portable modules.

The Java Bean mechanism is a useful way to approach this. Beans are essentially codes conforming to a strict application programming interface (API) through which the system can manipulate them. Native methods as well as Java code and also data can be encapsulated as Java Beans. We use the Java Bean mechanism to implement our portable code modules which are saved on a code-server [23] and are transmitted as high level instructions to servers at the request of clients. Use of well-defined APIs for components means that they may have a basic portable Java implementation, and an optimised version using native code, or even parallel code for a high-performance parallel computer, all of which can be invoked by the CPSE using the same interface.

The DISCWorld model for distributed, high-performance computing is readily implemented in Java. Performance was originally achieved primarily from embedding native application components in a Java software glue rather than from code running in Java itself. However, Java Virtual Machine (JVM) implementations have improved sufficiently so that high performance Java application components themselves can be utilised in a distributed environment. We have investigated some critical benchmark operations such as numerical analysis and image processing operations in pure Java and as native methods on various supercomputer platforms which can act as performance accelerators or specialist compute servers in a DISCWorld environment [22].

DISCWorld therefore provides us with a base platform architecture upon which CPSEs can be constructed. Two architectural aspects of DISCWorld are worth explaining in more detail and which underpin the DAG model described in section III. The DISCWorld Remote Access Mechanism (DRAM) provides a rich pointer to allow references between operators and data in the running system. These are used to represent the edges in the DAG and need to be carefully controlled if the system is to be reconfigured. This is particularly important for long running computations or in the case of partial failure of a CPSE due to network vagaries or indeed some users going

offline deliberately, such as reaching the end of their work shift. The DRAM facilities and its associated mechanism for data futures (DRAMFs) [19] provide a general architectural model for specifying how the system behaves under reconfiguration.

A. Remote Access Glue

The DRAM is a *rich pointer* because besides 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 to move the service to the data. DRAMs are unlike other, more traditional approaches to object references; they do not refer to any memory locations, therefore they are not fragile in the presence of remote server restarts [17]. This is particularly useful in a collaborative environment as there are no guarantees that the machine from which the data is being sourced will stay up for the entire collaborative session. In addition multiple references (DRAMs) can be created to the same DISCWorld object, thus all the members of a collaborative team can receive updates to the object when changes are made.

DRAMs can be used to point to data and services. Part of the metadata contained within a DRAM describes its bulk characteristics, such as the size (in bytes) of the data or program byte-code. In addition DRAMs for services (DRAMSs) contain rudimentary information on the runtime characteristics of the services to which they point. These characteristics can be used to create references to data which has not yet been created. We name these *DRAM Futures (DRAMFs)* [19].

A DRAM Future contains an estimate of the time that will take to produce the data to which it refers. This estimate contains tolerances for any other data that may need to be produced from previous DRAMFs. A DRAM Future represents a guarantee by the server that creates it, that the server *will* be able to generate the data in the estimated time frame.

We believe the combination of DRAMs, pointing to data, services and not-yet-created futures, together with an appropriate scripting language [16] provide the necessary framework to enable the efficient construction and utilisation of PSEs and CPSEs in a wide-area distributed system.

B. Builder Environment

We have largely focussed on developing the server-side software infrastructure of the DISCWorld metacomputing environment. However we have experimented with some simple client front ends that can act as DISCWorld nodes in their own right. It is a challenging problem to design an architecture for the visual environment that allows users to build collaborative applications.

Figures 1 and 2 show examples of our builder interface experiments. Figure 1 is a simplified example showing how a user might combine component operators together to produce a complete sequence. The example is restricted to very simple integer calculations to illustrate the builder idea only. Figure 2 shows how a more complex operator environment might be parameterised in some way to allow users to interact with particular components.

V. Interfacing to Other CPSEs

Technology and tools for PSEs in specific application areas have been around for some years now. Tools for problem solving in computational algebra and numerical simulations such as Matlab [34], Mathematica [39], Maple [38] and similar systems are quite mature. More recently efforts have led to PSEs which enable resource sharing such as NetSolve [4] and Ninf [30]. These were predated by a little cited research system developed by Mike Rezny which used the `mex` scripting capability of Matlab to interact with powerful back end systems to support operations like matrix solving [29].

Other systems such as Habanero [24] and TANGO [25] have exploited the powerful visual and graphical capabilities of Java to produce collaborative systems using the now well known metaphors of shared whiteboards, shared text processing and video conferencing. Such systems are able to tap into the now widespread desktop video camera and microphone technologies and provide an integrated approach to sharing access to running programs. Fox *et al.* have produced a demonstrator system WebWindows [13] showing how many of our everyday applications could be run at server side rather than on desktop client systems.

Application Service Providers (ASPs) are starting to become more prevalent on the Web and are providing access to specialist programs or services through the now well established Web client/server technologies. These include the Common Gateway Interface (CGI) approach; the servlet and smart applet approach; and web browser plug ins. These are all appropriate for the architectural model of single client and single server at any one transaction. These technologies are deficient however for a robust and multi-user CPSE environment.

Metacomputing or grid computing systems offer an ideal platform for supporting collaborative PSEs. The DISCWorld model shares some similarities with other grid computing systems currently in development. All of these systems currently use their own APIs, which makes interoperability a major issue. This is currently being addressed by the Grid Forum [14], who are working on standard interface specifications.

Globus/Nexus [11] is a system under active research and has a powerful ability to manage interprocess communication and servers by moving communications endpoints around. We are currently evaluating this feature of Nexus and trying to determine how Java networking mechanisms can be used to provide a similar capability. Unlike the DISCWorld model, Globus/Nexus addresses

fine-grained tasks. The external interface made use of an entity called the I-POP which performed the tasks of user authentication and security assurance. The I-POP has some degree of commonality with a DISCWorld daemon which must also handle these operations.

DISCWorld is designed to help schedule and run a range of distributed computations. This is an objective which it shares with the Nimrod [1] system. Nimrod allows a user to create an interface for managing parameterised simulations on a series of machines using user-supplied code. DISCWorld is aimed at the type of user who will be less likely to want to add their own code directly, but rather will want to utilise existing services, albeit in user specified combinations.

Legion [15] also provides an environment for integrating applications across wide-area networks. We believe Legion tackles the problem at a substantially finer grain than DISCWorld, allowing the user to link communications mechanisms to their own programs. The SNIPE [9] system also appears to operate at a finer grain than DISCWorld.

CORBA [26] is an alternative technology to Java in many of the technical aspects required for a CPSE. We believe current CORBA implementations are more appropriate for "local area DISCWorlds", where the resources involved are all owned by one organisation and accordingly administered. We are currently investigating how some additions to a CORBA based system might allow ORBs to find each other and exchange server/services information over a wider area and cross boundaries of ownership and administration, as we use in DISCWorld.

We are aware of other efforts to use Java for wide-area distributed computing, such as InfoSpheres [5]. We believe this too differs from DISCWorld in the granularity of application modules provided for.

The developers of Java and Jini have developed a number of new technologies that provide many of the low level mechanisms and infrastructure that make construction of a DISCWorld significantly easier. They do not however solve any of the higher level problems we outline.

We are working on a series of software protocol adapters and brokers that may provide a level of interoperability between DISCWorld servers and some of the systems mentioned above. Our own efforts with DISCWorld and other research systems such as the Visual Component Composition Environment (VCCE) of Walker, Rana and coworkers [37], and other Computational Grid technologies such as Globus and Legion are gradually exploring the fundamental issues for such a robust CPSE.

We believe these issues can be divided into two important categories. Firstly a group of low level issues that mostly involve standardisation and software engineering and for which possible solutions exist.

- technologies for integration;
- security and authentication;
- networking and ensuring connectivity between users;
- compression and transport of data to make best use of networks;
- remote execution of code on shared resources;
- migration of code and data for performance optimisation;
- exchanging data and format translations;
- exchanging libraries of components or services.

We believe these are potentially solvable issues with present and emerging technologies and with a degree of standardisation for exchange of component programs and data.

Secondly there are a number of more fundamental problem issues for which it is not obvious (at least to us) how to formulate a workable solution. These include:

- naming and metadata to describe the components, resources and data;
- resource discovery;
- reliability and robustness against failure;
- achieving full transparent reconfigurability in a system.

We anticipate some years of interesting research and debate in these latter issues.

VI. Conclusions and Ongoing Strategy

Metacomputing and grid computing systems provide high-level middleware to support efficient and transparent access to distributed data and compute resources, and therefore provide an ideal platform for developing collaborative problem solving environments. We have introduced our DISCWorld metacomputing environment and shown how this Java-based, object-oriented system has been used to develop prototype decision support systems and CPSEs that can be used over a wide-area network.

Reconfiguration is an important aspect of Problem Solving Environments in general and for Collaborative PSEs in particular. Our data flow model is a preliminary step to understanding the technical issues required to build working systems that truly span administrative boundaries and allow collaboration across wide areas.

Our key observation is that users can be treated in much the same way as programs – they are all "operators in control network" albeit with different latency and reliability properties. Collaborative systems involving interacting users rather than a single user making demands of the system present their own special problems. Reconfiguration lies at the heart of these.

We believe that using modern GUI technology such as Java it is possible to construct sophisticated and useful CPSE front ends that can exploit our architecture and middleware components. Our prototype has proved an effective way for image analysts and decision makers to collaborate across departments in defense and government organisations.

Outstanding issues remain those of constructing a metadata or name space and accompanying standards to allow user communities to exchange data and programs more easily. We believe that a decentralised approach to this problem is the only feasible one, but that some loose standards need to be first established in much the same

way that an HTML specification was required to stimulate open development of information to exchange on the WWW.

We believe the area of CPSEs is at a critical point socially and that the technology is almost mature enough to allow rapid growth. It will require more successful end-user application examples to be constructed to unearth the next level of technical issues for interdisciplinary collaborative systems. We believe the critical issues for future progress in CPSEs are those of component interoperability and interface standardisation.

References

[1] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A Tool for Performing Parameterised Simulations using Distributed Workstations. In *Proc. 4th IEEE Symp. High Performance Distributed Computing*, Virginia, August 1995.

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

[3] Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. Harper Collins Publishers, Inc., 1999. ISBN 0-06-251586-1.

[4] Henri Casanova and Jack Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *Int. J. Supercomputer Applications and High Performance Computing*, 11(3):212–223, Fall 1997.

[5] K. Mani Chandy, Adam Rifkin, Paolo A.G. Sivilotti, Jacob Mandelson, Matthew Richardson, Wesley Tanaka, and Luke Weisman. A World-Wide Distributed System Using Java and the Internet. In *High Performance Distributed Computing (HPDC-5) 1996*. Caltech, March 1996.

[6] 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, May 2000.

[7] P. D. Coddington and K. A. Hawick. Emerging Distributed Computing Tools and Technologies for Coordination of Emergency Services. Technical Report DHPC-080, Distributed and High Performance Computing Group, Department of Computer Science, University of Adelaide, June 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] Graham E Fagg, Keith Moore, Jack J. Dongarra, and Al Geist. Scalable Networked Information Processing Environment (SNIPE). In *Proc. SuperComputing 97*, 1997.

[10] Katrina E. Falkner. *The Provision of Relocation Transparency through a Formalised Naming System in a Distributed Mobile Object System*. PhD thesis, Department of Computer Science, The University of Adelaide, May 2000.

[11] Ian Foster and Carl Kesselman. Globus: A Meta-computing Infra-structure Toolkit. *Int. J. Supercomputer Applications*, 1996.

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

[13] Geoffrey C. Fox and Wojtek Furmanski. WebWindows Motivation and Application to Distributed Metacomputing. CRPC Presentation, Annual Review at Houston, Texas, March 1995.

[14] The Grid Forum. Grid Forum Home Page. http://www.gridforum.org/.

[15] 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.

[16] 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.

[17] 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*, 1999.

[18] P. W. Eklund J. E. Mann, S. D. Kirkby. A JAVA Based Fire Simulation Model for the WWW. In *Proc. 3rd Australian World Wide Web Conference*, pages 296–298, Brisbane, 1997.

[19] 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.

[20] H. A. James, K. A. Hawick, and P. D. Coddington. An Environment for Workflow Applications on Wide-Area Distributed Systems. Technical Report DHPC-091, Distributed and High Performance Computing Group, Department of Computer Science, The University of Adelaide, May 2000. Submitted to HICSS'34.

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

[22] J. A. Mathew, P. D. Coddington, and K. A. Hawick. Analysis and Development of Java Grande Benchmarks. In *Proc. of the ACM 1999 Java Grande Conference*, April 1999.

[23] J. A. Mathew, A. J. Silis, and K. A. Hawick. Inter Server Transport of Java Byte Code in a Metacomputing Environment. In *Proc. TOOLS Pacific (Tools 28) - Technology of Object-Oriented Languages and Systems*, 1998.

[24] National Center for Supercomputing Applications. Hanbanero. Available at http://www.ncsa.uiuc.edu/SDG/-Software/Habanero/, last visited May 2000.

[25] Northeast Parallal Architectures Center at Syracuse University. The Tango Project. Available at http://www.npac.syr.-edu/projects/tango/, last visited May 2000.

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

[27] C. J. Patten, F. A. Vaughan, K. A. Hawick, and A. L. Brown. DWorFS: File System Support for Legacy Applications in DISCWorld. In *Proc. Fifth IDEA Workshop*, February 1998.

[28] Craig J. Patten and K. A. Hawick. Flexible High-Performance Access to Distributed Storage Resources. In *Proc. Ninth IEEE Symp. on High Performance Distributed Computing*, August 2000.

[29] Mike Rezny. *Aspects of High Performance Computing*. PhD thesis, Department of Mathematics, University of Queensland, November 1995.

[30] Satoshi Sekiguchi, Mitsuhisa Sato, Hidemoto Nakada, and Umpei Nagashima. – Ninf – : Network base information library for globally high performance computing. In *Parallel Object-Oriented Methods and Applications (POOMA)*, February 1996.

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

[32] Sun Microsystems. Java Advanced Imaging API. Available from http://java.sun.com/products/java-media/jai/, November 1998.

[33] Sun Microsystems. Java Products Homepage. Available from http://www.javasoft.com/products/, last visited July 1999.

[34] The Mathworks, Inc. MATLAB. Available at http://www.-mathworks.com, last visited May 2000.

[35] U.S. National Imagery and Mapping Association. USIGS Geospatial and Imagery Access Services (GIAS) Specification, version 3.1, N0101-B. Available from http://www.nima.mil/-aig/products/uip/gias/, February 1998.

[36] U.S. National Imagery and Mapping Association. Geospatial and Imagery Exploitation Service (GIXS) Specification. version 2.0, June 1999.

[37] D. W. Walker, M. Li, O. F. Rana, M. S. Shields, and Y. Huang. The Software Architecture of a Distributed Problem-Solving Environment. Technical Report ORNL/TM-1999/321, Oak Ridge National Laboratory, February 2000.

[38] Waterloo Maple, Inc. Maple. Available at http://www.-maplesoft.com, last visited May 2000.

[39] Wolfram Research, Inc. Mathematica. Available at http://-www.wolfram.com/products/mathematica, last visited May 2000.