An Evolution of the WebCom Metacomputer [*]

John P. Morrison, David A. Power and James J. Kennedy
Centre for Unified Computing,
Dept. Computer Science,
National University Ireland,
Cork,
Ireland.

e-mail {j.morrison,d.power,j.kennedy}@cs.ucc.ie

Abstract

Functional enhancements to the WebCom metacomputer are described which give rise to dynamic reconfigurability and extendability of the computer platform. Component modules and interactions are described, with particular attention to the communications module that enables dynamic reconfigurability. The machines of the metacomputer can be configured to act in client/server or peer to peer mode on a number of interconnection topologies such as NOWs, Clusters or Grids. This paper addresses the dynamically extendable machine structure of WebCom facilitated by this new communications structure.

Keywords: Java, Condensed Graphs, Parallel Computing, Distributed Computing, Grid Computing, Metacomputing

Mathematics Subject Classification: 68R01

---

1 Introduction

WebCom [15, 18, 19] was initially created to investigate the execution of programs expressed as Condensed Graphs [14, 16] in the distributed environments of the World Wide Web and the Internet. Projects in this area include Cilk-NOW[13, 23], TreadMarks[6] in which the processing power of locally grouped workstations is exploited and also projects like Charlotte[2, 11], Bayanihan[7, 12, 25], Javelin[4] which utilize the processing power of the the Internet.

These Metacomputing systems typically operate by distributing work to volunteer machines. This is achieved by utilizing Java applets, with different applet types being used for different tasks. These applets may then communicate by using Remote Method Invocation (RMI)[10] and Object Request Brokers (ORB's)[26].

With these systems, a distributed application is typically constructed by using object libraries or API's provided and places the onus on the programmer to explicitly parallelise the problem. In addition to explicitly determining the parallelism of the problem and synchronizing the interaction of its individual components, the programmer also has to provide the required level of fault tolerance, load balancing and scheduling algorithms. This, in conjunction with the use of ORBs and RMI greatly exacerbates the complexity of these systems.

WebCom separates the application from the platform. It provides an implementation architecture and development tools so solutions can be developed independently of the physical constraints of the underlying hardware: the same Condensed Graphs programs run without change on a range of implementation platforms from silicon based Field Programmable Gate Arrays[22] to the WebCom metacomputer. Fault tolerance, load balancing and the exploitation of available parallelism are all handled implicitly by WebCom without explicit programmer intervention. In addition, development tools are available and provide mechanisms for creating applications expressed as Condensed Graphs. When expressing a problem as a Condensed Graph, nodes indicate its various subtasks and edges determine the way in

which these subtasks are sequenced for execution. By altering the connection topology of the graph various evaluation orders can be specified. Sequencing constraints can be specified statically by a programmer but they can also be altered dynamically using feedback from the underlying execution environment. The latter alternative leads to a tuning of the parallelism of a given problem to the parallelism of the implementation platform.

In Fig. 1, two equivalent Condensed Graphs implementations are depicted which calculate powers of two. In Fig. 1(a), the sequencing of tasks is determined by data dependencies where as in Fig. 1(b) a lazy, demand-driven sequencing of tasks results from the altered connection topology.
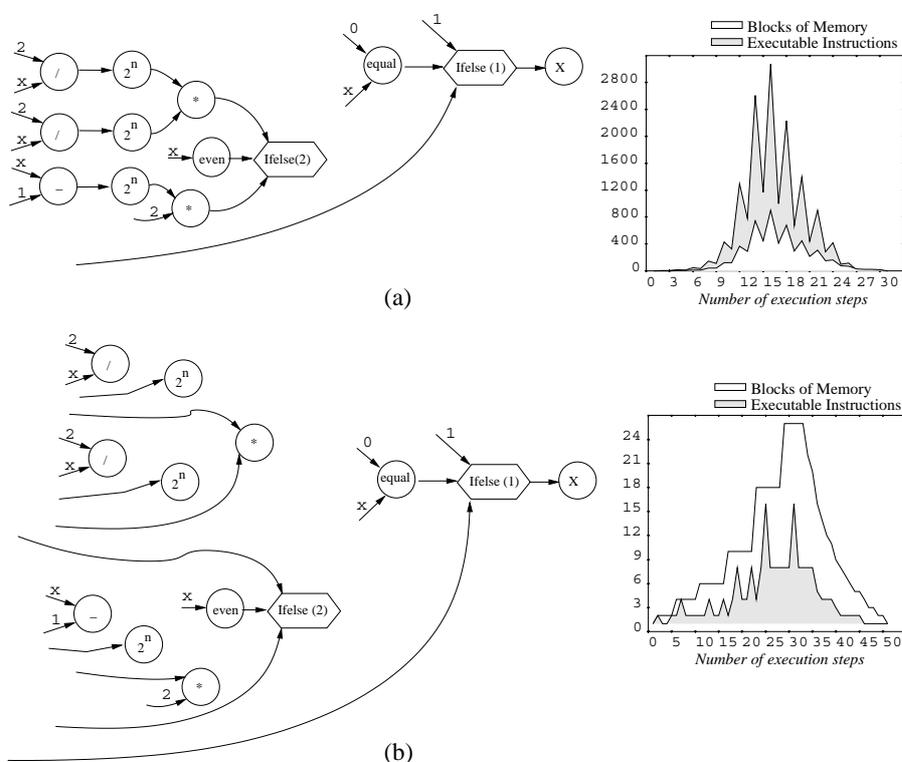

Figure 1: *An eager and lazy version of the Condensed Graph to calculate 2^n. The associated execution profiles illustrate the amount of parallelism exposed per unit time in the calculation of 2^{10}. Note that the eagerness of the graph (a) extends to calculating two to the power of negative indices. Hence speculative parallelism is exposed until such time as the correct result is calculated. In contrast the lazy version (b) proceeds tentatively towards a solution exposing only parallelism when needed.*

In common with other metacomputing systems, WebCom uses a server/client (Fig. 2)

model for task distribution. However, WebCom clients are uniquely comprised of both volunteers and conscripts. Volunteers donate compute cycles by instantiating a web based connection to a WebCom server. A client abstract machine (AM), constrained to run in the browsers sandbox, is automatically downloaded. This AM executes tasks on behalf of the server and returns results over dedicated sockets. Volunteers created in this manner can act only as clients. In contrast, volunteers that have the WebCom platform pre-installed can act as clients but may also be dynamically promoted to function as WebCom servers. This promotion occurs when the task passed to the volunteer can be partitioned for subsequent distributed execution (i.e. when the task represents a Condensed Graph). The return of a result may trigger the demotion of a server, thus causing it to act once more as a simple client.
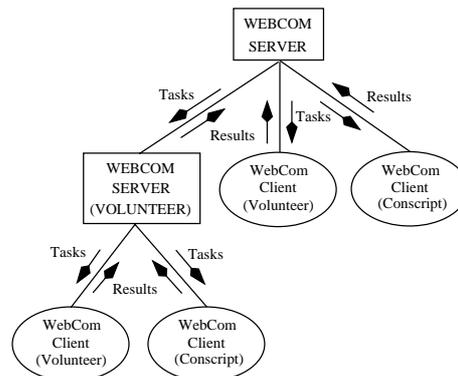

Figure 2: *WebCom Client/Server connectivity. Each Client can either volunteer cycles to a WebCom server, or be conscripted as a WebCom client through Cyclone.*

Intranet clients may be conscripted to WebCom by the Cyclone[20] server. Cyclone directs machines under its control to install the WebCom platform thereby causing them to act as promotable clients attached to a pre-defined parent WebCom server.

Although WebCom was created as a harness for executing Condensed Graphs, the compute engine was constructed as an interchangeable component. To maximise the applicability of WebCom to different application areas, it was decided to extend this approach to all of WebCom's functional components.

2 Extending WebCom

To allow for better tuning to specific application areas, WebCom was reconstructed to make every component dynamically interchangeable. Certain components maybe necessary while others are highly specialised in particular application environments. A skeletal installation is composed of a Backplane module, a Communications Manager Module(CMM) and a number of module stubs as illustrated in Fig. 3.
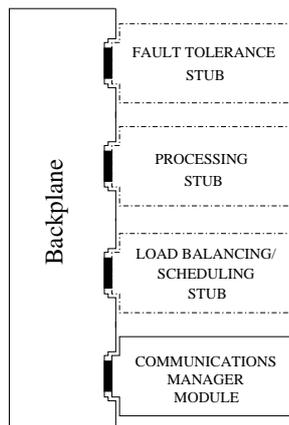

Figure 3: *A minimal WebCom installation consists of a backplane module a communications manager module and a number of module stubs for the compute engine, fault tolerance and load balancer.*

Initialisation is carried out by the backplane. This first attaches the installed communications manager module, followed by each of the module stubs for the compute engine, fault tolerance and load balancer. A stub exhibits the minimal functionality needed to facilitate the flow of tasks through the system. It is used only in the absence of a more functionally complete module implementation and can be readily exchanged for such an implementation when it becomes available. Each module may consist of a number of variants. These are maintained in a module cache as illustrated in Fig. 4. This cache is referenced whenever one variant is swapped for another. For example, there may be a number of load balancing modules, each invoking a specific algorithm. The execution environment determines which module variants to employ in each part of the execution based on the information propagated with the task.
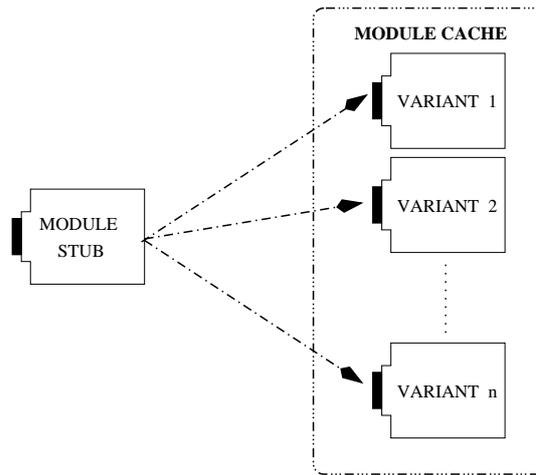
Figure 4: *Module variants propagated to a WebCom installation are maintained in a Module Cache.*

An initial, static, client configuration is performed when WebCom clients are initialised. This configuration may be dynamically altered as the computation proceeds – yielding optimal execution environments for each task. Task execution environments specify which module variants to be used and where they can be located on the network. Such information is compiled by the sender – possibly using its knowledge of the network and of the task to be performed or from information derived directly from the program code. If a client's task execution environment is to be changed, the required modules are plugged in from its module cache. The cost of switching between module variants is low. Once the module has been instantiated, this cost reduces to determining which module should be invoked. If a module variant is not resident in the cache, the initial cost of invoking it is larger: a request for the module must be issued or a search for the module must be initiated; the module must be physically transmitted to the requester, installed and instantiated. Some of the latency involved in this fetching phase may be masked by task pipelining. The mechanism to facilitate this module migration is described later. Task execution on the client will pause until the correct configuration is obtained. In the meantime, the server will monitor the tasks expected execution time. If this exceeds a predetermined value, the server will reschedule the task to an alternative client under the assumption that the initial allocation

had failed. it will recalculate the estimated completion time of the task and will monitor the new execution as before. In this scenario, it is possible that the initial task execution will terminate normally either before or after the rescheduled task. The first result to be returned will be used in subsequent computations. The appearance of many results from the same, multiply rescheduled task, would indicate that the dynamic properties of the network are at variance with our estimated time to completion algorithm. In that case the estimated time to completion could be lengthened for a specific period of time. The goal in altering this estimate is to maintain a quality of service while keeping the number of reallocated tasks to a minimum.

In the original implementation WebCom was required to execute at least one instruction from each task passed to it for execution. This characteristic meant that WebCom was unsuitable for passing tasks unchanged between servers. Consequently, servers could not act as a proxies. In practice certain machines such as those behind firewalls or those restricted to particular subdomains could not participate in a computation outside that firewall or subdomain.

By simply allowing a WebCom server to redirect tasks, gateway and firewall machines can effectively act as proxy servers; exposing the services of its hidden clients to the wider WebCom computation. In this role, the proxy server appears to act as a super client but this status does not prevent it from also being promoted to a fully fledged WebCom server.

2.1 Module Interactions

The pluggable nature of each WebCom module provides great flexibility in the development and testing of new components. The backplane co-ordinates the activities of the other modules via a well defined interface. In general, a processing module will execute tasks locally and will identify tasks for distribution to client machines. Tasks uncovered by the processing module are passed through the backplane to the load balancing/scheduling module. This then consults with the fault tolerance and communications manager modules to decide on

the optimum location for executing that task. When the decision is made, the fault tolerance module is notified and its administration is updated accordingly. If a task is to be executed locally it is placed on the execution queue of the processing module. The backplane decides on the appropriate processing module to use according to the task type, which is specified as part of the execution environment, Fig. 5. For example, if the task invokes a Corba service then a processing module capable of executing that service is selected. Alternatively, If a task represents a Condensed Graph application then the Condensed Graph processing module is used, resulting in client to server promotion. Specific virtual machines(VM) may be specified for the execution of particular tasks; and clients are targeted based on their ability to host an available VM to execute those tasks. This approach facilitates task migration in a heterogeneous network.
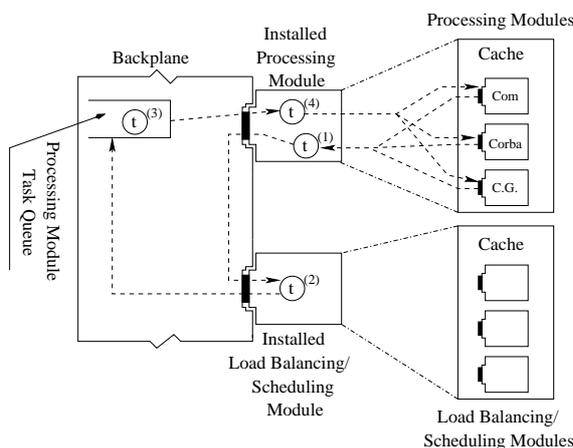

Figure 5: *Tasks uncovered by the processing module that are executed locally. (1)The installed processing Module uncovers a task t for execution. (2) The task is passed to the backplane and onto the load balancer/scheduling module. This module decides to execute the task locally. (3) The load balancer requests the backplane to queue the task for local execution. (4) The backplane determines the appropriate processing module for the tasks, installs (or switches to) that module and requests the module to execute the task.*

At preset security is restricted to the mechanisms provided by the particular virtual machine. The Condensed Graphs Virtual Machine implements the Keynote[8] standard. A cost is associated with not running the task as native processes. However, for non high-performance computations this cost is more than offset by the associated flexibility.

If a task is to be executed remotely, the appropriate execution environment is determined by the load balancing module. Again, the fault tolerance administration is updated and the task is passed through the backplane to the communications manager for subsequent dispatch, Fig. 6.

The flexibility of this mechanism allows for different paths through the execution graph to be processed using different fault tolerance and load balancing strategies; reflecting either the priority, criticality and/or coupling of various pathways. The ability to express this information in the architecture means that it can be propagated to the application layer and exploited by the programmer. In the WebCom Integrated Development Environment, the programmer can explicitly bind communication protocols, processing modules, load balancing heuristics and fault tolerance strategies with each subtask.
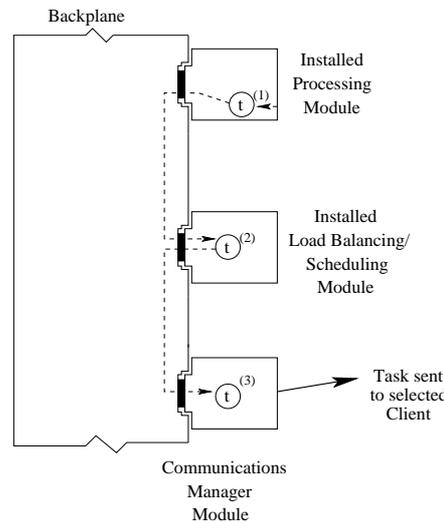

Figure 6: *Tasks uncovered by the processing module that are executed remotely. (1)The installed processing Module uncovers a task t for execution. (2) The task is passed to the backplane and onto the load balancer/scheduling module. This module decides to execute the task remotely. (3) The load balancer sends the task to the communications manager module for dispatch to the client selected to execute the task.*

For tasks which will be executed remotely, the communications manager module adds additional route information to the task and sends it on to the selected client. This route information may be subsequently modified if a fault occurs. The routing information will also be used to pass the result of the computation back to the originating server.

Results of task execution received by the communications manager module will be either incorporated into the computation or forwarded to the server that issued the task. Results incorporated into the computation expose further tasks for execution.

The load balancing/scheduling module will decide where and when tasks are executed. It will suggest one or more optimum locations for the execution of a task. The sophistication of the backplane will determine the effort expended in managing task execution. The appropriate sophistication levels will vary depending on the position of the backplane's host machine in the network topology. An extension to WebCom requires all load balancing modules to communicate directly with the backplane and relies on it to provide information on the status of the network by interacting with the Communications Manager Module. Each load balancing module may employ a number of separate strategies taking account of issues such as security restrictions, specialised resource locations and access reliability (less reliable clients may be chosen to execute speculative tasks[21], whose completion or otherwise may not affect the critical path of the computation).

To maximize its effectiveness, a load balancing module should exploit profile information[17] of attached clients. For example, in suggesting a location to execute a particular task which is an x86 binary, it would consider not only those x86 clients but also those capable of interpreting the code with sufficient speed.

In addition to these generic considerations different load balancing modules can be employed to implement specific load balancing algorithms.

Fault tolerance on 2-tier systems[11, 12] is usually a matter of fault masking in which work scheduled to a failed processor is simply rescheduled. The affect of a fault in these systems is proportional to task size. The extended version of WebCom, with its hierarchical topology requires a more sophisticated fault tolerance mechanism since the failure of a machine at some arbitrary position in the hierarchy could result in the orphaning of a subtree of processors.

Following the general WebCom philosophy, arbitrary fault tolerance and fault survival modules can be installed to complement the specific execution strategy adopted by the

processing module. In the case of a Condensed Graphs execution, locality offered by the model gives rise to a fault tolerance procedure that deals with faults in any one of three contexts as illustrated in Fig. 7.
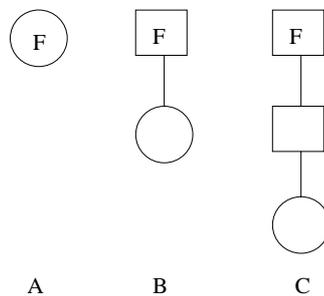

Figure 7: *Fault Contexts: Boxes represent clients acting as WebCom servers, circles represent clients executing non distributeable (atomic) tasks. F indicates the location of a fault.*

In Context A, work cannot be recovered from the failed client and fault masking techniques should be used.

In Contexts B and C, the fault tolerance procedure replaces the failed processor, with an equivalent, and reconnects clients in such a way that the network topology is maintained. Subsequently, it uses partial results stored by the client (as part of a lightweight checkpointing mechanism) to rebuild the computation to a point prior to failure. The management information regarding scheduling decisions and protocol selections are currently lost when a fault occurs. This information is rebuilt from scratch when a replacement processor is installed.

The remainder of this paper concentrates solely on the communications manager module and the high degree of control offered by this approach in conjunction with WebCom.

3   Communications Structure

To construct a fully functional metacomputer, core modules that provide fault tolerance[1, 5, 9], load balancing/scheduling [3, 27], compute engine and backplane have to be installed, Fig. 3. Initial configuration, Fig. 8, will specify where these modules are to be found: on

a local file server or WebCom host. In execution, these modules are obtained and installed locally by the communications manager module, thus dynamically bootstrapping and configuring each host.
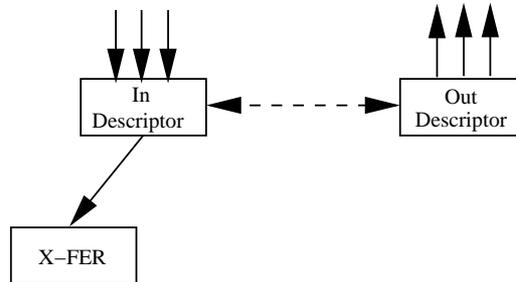

Figure 8: *Each WebCom base installation consists of two connection descriptors and a file transfer mechanism.*

The communications manager module consists of two *connection descriptors*: one maintaining an arbitrary number of incoming connections and the other maintaining an arbitrary number of outgoing connections. A connection represents a unidirectional channel between two host machines.

To allow communication of tasks and results between machines, connections occur in pairs. Where tasks and results are communicated, the number of incoming and outgoing connections on each machine will typically be the same. However, this may not be the case when a machine acts as a data source or data sink.

Associated with each connection is a *connection state*. This is consulted when managing the connections between machines. For example, it detects faults, intentional disconnections (which can be described as being either *hard* or *soft*) and the availability status of the machines. The status of a machine will be described by the *Available*, *Busy* and *Unavailable* flags. An Available flag indicates tasks can be scheduled to a machine. It will be flagged as busy if it is executing a task. It will be marked as unavailable if it is not willing to accept tasks. A machine can be both busy and available or busy and unavailable. For example, a gateway of firewall may be flagged as busy and available provided it is executing a task or at least one of its clients is busy, and it is willing to accept a task or at least one of its clients

is available.

Scheduled tasks are transferred from server to client using either a push or a pull mechanism. The push mechanism allows a server to transmit tasks to a client provided the available flag of that client is set. This mechanism assumes that sufficient buffer space exists on the client to accept incoming tasks. The pull mechanism requires a server to maintain a queue of scheduled tasks for each attached client and clients are responsible for fetching tasks from their associated queues. The pull mechanism can only be used by certain clients as a result of sandbox restrictions. For example, it is not possible to push tasks to a Java applet.

The push mechanism dynamically commits a task to a client. In general it may not be possible to undo this commitment if a better scheduling decision is subsequently determined. In contrast the scheduling queues on the server associated with the pull mechanism may be rebalanced to reflect better scheduling decisions. For example, if the rate at which the client is pulling tasks of equivalent complexity from its queue lengthens, the scheduler may infer that the client's local load is increasing, or that there is a problem with the client server communication. In either case, the server may decide to redistribute the tasks in that clients queue to ensure faster throughput.

A machine intending to gracefully depart from WebCom will perform a hard intentional disconnect. This process causes any clients of the machine to be redirected to a specified server and informs the machines server, if it exists, of its intention to leave. The appropriate connections are closed and removed from the respective descriptors. When a connection closes any pending tasks or results waiting to be transmitted over that connection will be handled either by the load balancer or fault tolerance module.

A soft intentional disconnection may be issued when a machine decides to process tasks scheduled to it offline. A soft disconnection indicates that a client is temporarily unavailable. It could, of course, be fully available from the perspective of other connections. A connection marked as a soft disconnection is not physically removed from its associated descriptors. Tasks may still be scheduled for transmission over these connections in the expectation of

them being restored at a future time. A connection that changes its disconnection mode from soft to hard will result in any pending tasks being handled by the load balancer and fault tolerance module as before. Both hard and soft intentional disconnections are issued only by outgoing connections.

The availability status of a connection is decided at the outgoing descriptor and is propagated to the machine to which it is connected. The machine is then obliged to take this information into account in communicating over its associated return connection. A client can indicate different status conditions on each connection. Therefore, a client may be free from the perspective of one machine, busy from the perspective of a second and unavailable from the perspective of a third. Indeed, since availability is determined per connection, two or more connections between two machines could all indicate a different availability status. This can be used to dynamically build priority into the communication channels.

3.1 Inbound Descriptor

An exploded view of an inbound descriptor is shown in Fig. 9. This figure shows a number of WebCom communication mechanisms. Communicating hosts can dynamically switch between these mechanisms by issuing a protocol select message. The appropriate communication mechanism is then selected on the receiving side. In this way, hosts may communicate using serialised Java objects, XML, or any other desired protocol assuming the pluggable module is available. The selected protocol will be determined by the requirements of the tasks being transmitted.

The inbound descriptor maintains a list of connections. Each connection is usually associated with an outbound connection on the same host, so that results pertaining to a received task can be returned.

All communication mechanisms implement a predefined Java interface. This allows mechanisms to be easily "plugged in" or "plugged out". A number of mechanisms may be plugged in at any time, however, only one is associated with a connection at any one time.
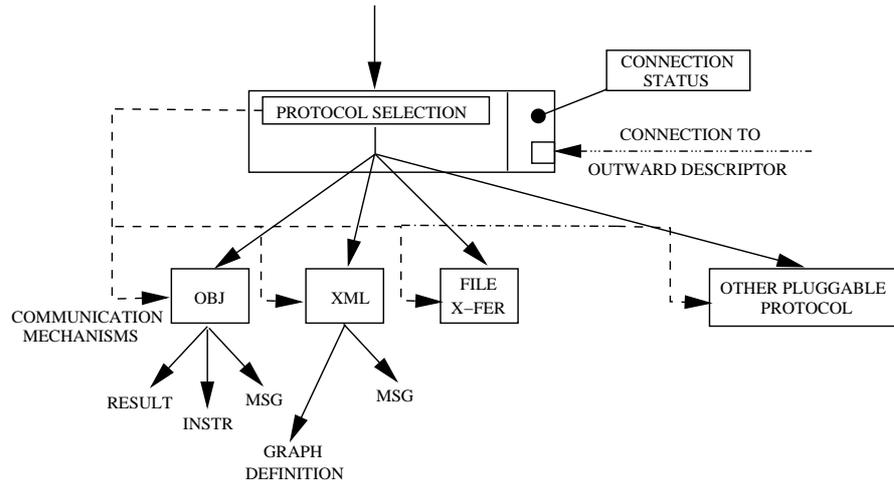
Figure 9: *An inbound descriptor and associated component communication mechanisms.*

3.2   Outbound Descriptor

An exploded view of an outbound descriptor is given in Fig. 10. Before a task is transmitted over an outbound connection a protocol select message may be transmitted indicating the communication format in which the task will be sent.
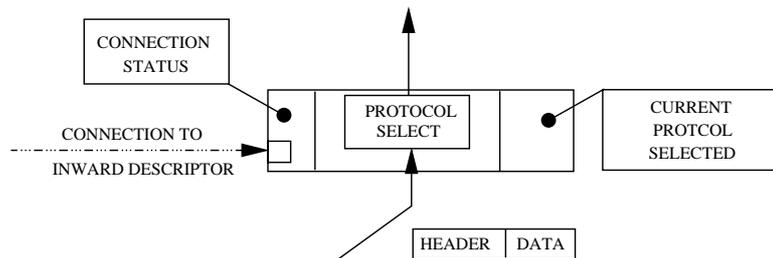

Figure 10: *An Outbound Descriptor.*

On a machine where there are pairs of connections to other clients, there is a bidirectional communication between the descriptors for each pair of connections.

This feature has a number of purposes. It can be used to return acknowledgments of successful protocol changes. If faults are detected which result in one of the channels becoming unavailable and subsequently being removed, the corresponding channel can also be removed from the other descriptor. If the availability status of one of the connections is changed, it may also be necessary to change the status of its pair connection.

4   Conclusions & Future Work

WebCom implements an extremely flexible metacomputing architecture in which various pluggable components interoperate. This is achieved by requiring each component to implement a specific predefined interface. The quality of the information delivered by each component will vary based on its level of sophistication but will not influence the subsequent interaction with other modules of the system.

This paper concentrates on the communication structure of WebCom which is used not only to distribute tasks and results but also to dynamically bootstrap and reconfigure the system during the execution of an application. Application specific information such as the criticalness of certain subtasks and the desired tightly, or loosely connected nature of other subtasks can be propagated to the architecture layer where it is enforced through the task execution environment mechanism.

Using the VM paradigm, WebCom may view a heterogeneous network as being homogeneous at a higher level of abstraction. The benefits of this approach are better use of resources and better fault survival since a greater variety of underlying hardware is exploited. However a cost lies in increased execution time. Nevertheless with further advances in VM technology and in particular in binary interpretation[24] this cost will undoubtedly diminish.

A future consideration of the WebCom project is to support general process migration. Although not trivial, this involves the wrapping and dispatching of an executing task together with its operating environment to an arbitrary destination machine. The mechanisms described here provide an important initial step in this direction.

References

[1] A. Avizienis. Fault-Tolerence: The surrival attribute of digital systems. In *IEEE, Vol 66, No. 10*, pages 1109–1125, 1978.

[2] Arash Baratloo, Mehmet Karul, Zvi Kedem, and Peter Wyckoff. Charlotte: Metacomputing on the Web. 9th International Conference on Parallel and Distributed Computing Systems, 1996.

[3] Wolfgang Becker. Dynamic Load Balancing for Parallel Database Processing. Faculty Report No. 1997/08, Institute of Parallel and Distributed High-Performance Systems(IPVR), University of Stuttgart, Germany.

[4] P. Cappello, B. O. Christiansen, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-Based Parallel Computing Using Java. In Geoffrey C. Fox and Wei Li, editors, *ACM Workshop on Java for Science and Engineering Computation*, June 1997.

[5] A. Avizienis Chen L. N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation. Proceedings of FTCS 8, 1978, pp.3-9.

[6] Cristiana Amza et al. TreadMarks: Shared Memory Computing on Networks of Workstations. IEEE Computer, Pages 18-28, Vol. 29, No. 2, February 1996.

[7] Luis F. G. Sarmenta et al. Bayanihan Computing .NET: Grid Computing with XML Web Services. Workshop on Global and Peer-to-Peer Computing at the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 02), Berlin, Germany, May 2002.

[8] M. Blaze et al. The Keynote Trust Management System, Version 2. Internet Request for Comments 2704. September 1999.

[9] J. L. Gaudiot and C.S. Raghavendra. Fault-Tolerence and Data-Flow Systems. In *5th international conference on Distributed Computing Systems*, Denver, Colorado, May 13-17 1985.

[10] Sun Microsystems Incorporated. Remote Method Invocation. http://java.sun.com/products/jdk/rmi/.

[11] Mehmet Karul. Metacomputing and Resource Allocation on the World Wide Web. Phd thesis, New York University, May 1998.

[12] Satoshi Hirano Luis F. G. Sarmenta and Stephen A. Ward. Towards Bayanihan: Building an Extensible Framework for Volunteer Computing Using Java. ACM 1998 Workshop on Java for High-Performance Network Computing, Palo Alto, California, Feb. 28 - Mar. 1, 1998.

[13] M. O. Rabin M. A. Bender. Online Scheduling of Parallel Programs on Heterogeneous Systems with Applications to Cilk. Theory of Computing Systems Special Issue on SPAA '00, 35: 289-304, 2002.

[14] John P. Morrison. *Condensed Graphs: Unifying Availability-Driven, Coercion-Driven and Control-Driven Computing.* PhD thesis, Eindhoven, 1996.

[15] John P. Morrison, James J. Kennedy, and David A. Power. Extending Webcom: A Proposed Framework for Web Based Distributed Compuiting. MSA2000.

[16] John P. Morrison, David A. Power, and James J. Kennedy. A Condensed Graphs Engine to Drive Metacomputing. Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA '99), Las Vagas, Nevada, June 28 - July1, 1999.

[17] John P. Morrison, David A. Power, and James J. Kennedy. Load balancing and fault tolerance in a condensed graphs based metacomputer. The Journal of Internet Technologies. Accepted for Publication 2002.

[18] John P. Morrison, David A. Power, and James J. Kennedy. WebCom: A Volunteer based meta computer. The Journal of Supercomputing, 18, 47-61, 2001.

[19] John P. Morrison, David A. Power, and James J. Kennedy. WebCom: A Web Based Distributed Computation Platform. Proceedings of Distributed computing on the Web, Rostock, Germany, June 21 - 23, 1999.

[20] John P. Morrison, Keith Power, and Neil Cafferkey. Cyclone: Cycle Stealing System. Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA 2000), Las Vagas, Nevada, June 26 - 29, 2000.

[21] John P. Morrison and Martin Rem. Speculative Computing in the Condensed Graphs Machine. proceedings of IWPC'99: University of Aizu, Japan, 21-24 Sept 1999.

[22] J.P. Morrison, Padraig J. O'Dowd, and Philip D. Healy. Searching RC5 Keyspaces with Distributed Reconfigurable Hardware. To appear, ERSA 2003, Las Vegas, June 23-26, 2003.

[23] Philip A. Lisiecki Robert D. Blumofe. Adaptive and Reliable Parallel Computing on Networks of Workstations. *Proceedings of the USENIX 1997 Annual Technical Symposium*, January 1997.

[24] Alan Robinson. Why Dynamic Translation? http://www.transitive.com/pressroom.htm#techpapers.

[25] Luis F. G. Sarmenta. Bayanihan: Web-Based Volunteer Computing Using Java. 2nd International Conference on World-Wide Computing and its Applications (WWCA'98), Tsukuba, Japan, March 3-4, 1998.

[26] The OMG Group. Common Object Request Broker Architecture, July 1995. http://www.omg.org.

[27] Jerrell Watts and Stephen Taylor. A Practical Approach to Dynamic Load Balancing. Scalable Concurrent Programming Laboratory, Syracuse University.