dClips: A DISTRIBUTED CLIPS IMPLEMENTATION

Y. Philip Li
li@aero.org

Aerospace Corporation

June 30, 1993

Abstract

A distributed version of the Clips language, dClips, was implemented on top of two existing generic distributed messaging systems to show that: (1) it is easy to create a coarse-grained parallel programming environment out of an existing language if a high level messaging system is used, (2) the computing model of a parallel programming environment can be changed easily if we change the underlying messaging system. dClips processes were first connected with a simple master-slave model. A client-server model with intercommunicating agents was later implemented. The concept of service broker is being investigated.

Introduction

In the process of exploring the opportunities of utilizing multiple workstations on a network as a single parallel computing environment, we have built a simple distributed Clips environment, named dClips, running with multiple parallel Clips processes on a Sun network. Clips, C language integrated production system [Clips91], is a forward-chaining rule-based language with object definition capability. Clips was developed by NASA Johnson Space Center.

dClips, was implemented on top of two existing generic distributed messaging systems to show that: (1) it is easy to create a coarse-grained parallel programming environment out of an existing language if a high level messaging system is used as the underlying layer, (2) the computing model of a parallel programming environment can be changed easily if we change the underlying messaging system. In this paper, we describe two versions of dClips implementation on top of two different messaging systems. One messaging system supports only the master-slave model while the other supports a much more flexible communication scheme.

A Master-Slave Model for Task Assignment

dClips was first implemented on top of AERO [Sullivan89], the Asynchronously Executed Remote Operations from UC Berkeley. AERO allows parallel programming in a master-slave mode on a UNIX network. Communication is only allowed between the master and slaves, but not in between slaves. The master process can assign tasks asynchronously, but it has to block and wait for the result to come back.

As depicted in Figure 1, a single dClips master process controls multiple dClips slave

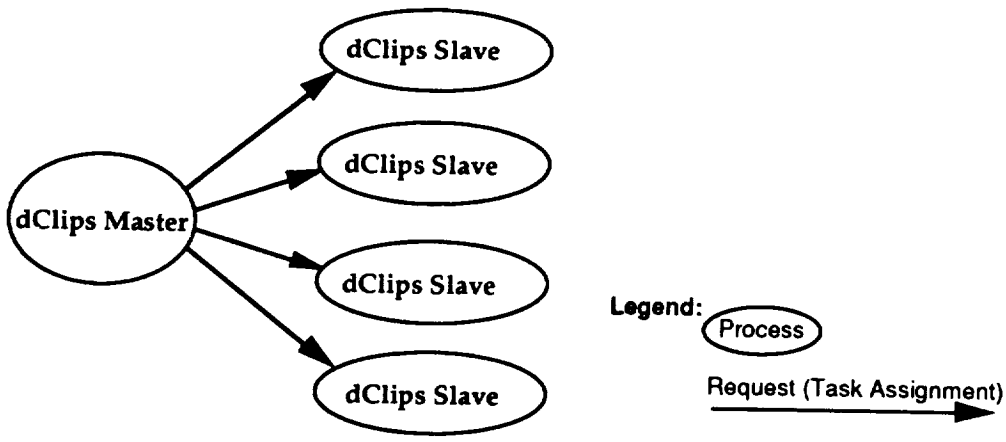

Figure 1. Master-Slave Model for dClips

processes. The master process first asks all the slave processes to load the necessary Clips constructs (i.e., rules, objects, and functions) from the file system into their runtime environments. The master then assigns tasks to slave dClips processes by one of the following three methods:

i). Assert a fact into Clips knowledge base — This request from the master process is executed on all the slave processes simultaneously. If a slave process is busy, it first finishes its current task then asserts the fact. By asserting facts into the working memory of slave processes, the master process could change the inferencing process in the slaves.

ii). Call a Clips function — Any built-in Clips function and user-defined functions in a slave process can be called from the dClips master process. This is a form of remote procedure call in the context of Clips language. Also, the state of the working memory of a dClips slave process can be examined by the master by issuing Clips function call. This allows the master to decide if further task assignment is necessary.

iii). Send a message to a Clips object — A Clips object message can be sent from the dClips master process to dClips slave processes. An active object instance within the slave process can receive messages and process the messages based on the behaviors defined in a message handler.

This version of the dClips implementation is done in C using Clips 5.1. Four function calls are available between the dClips master and slave processes: loadClipsConstruct, assertClipsFact, callClipsFunction, and sendClipsMessage. The *loadClipsConstruct* primitive can take a list of construct-files (i.e., a file with Clips rules, objects, and functions) and process them based on the sequence of the list elements. The sequence in the list is the sequence of execution in the loading process. For example, the list (function.clp object.clp rule.clp) will cause a slave process to load function.clp first, object.clp next, and rule.clp last.

The *assertClipsFact* primitive takes a string with a single Clips fact and requests every slave to assert it into the knowledge base. The *callClipsFunction* primitive takes a string with a single function name and the function parameters, and sends it to all the slave processes.

The *sendClipsMessage* primitive is also capable of passing a list of messages from master to slave. Each message is itself a list in the form: (class-name instance-name method-name method args). The slave that receives a sendClipsMessage request processes the messages based on the sequence of the list elements. This allows multiple class methods to be defined and executed in sequence as a single work assignment.

An Application

A Clips-based image data access application, DataHub [Handley92], has been ported to the dClips/Aero environment. The master process issues concurrent image data access/conversion requests for different dataset types. Each dataset type has different data format and data semantics. The knowledge about the image datasets is stored in Clips constructs, and loaded by the slave processes at startup time. One dataset type is handled by one slave process. There is no interaction needed among slave processes. Data conversion tasks are both CPU intensive due to format changes (e.g., byte swap, data decompression) and I/O extensive due to massive read and write of files.

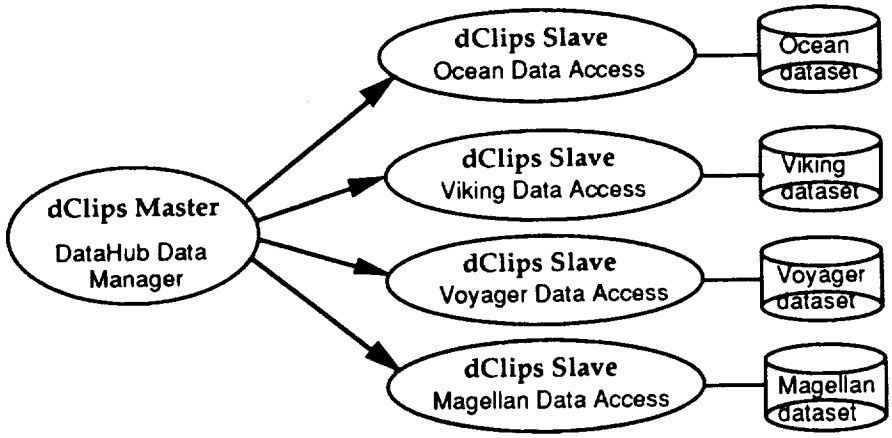

Figure 2. DataHub Data Manager on top of dClips

The DataHub data manager with the same master-slave task assignment scheme has also been ported to the dClips/ISIS environment (see next section for details). The master slave model stays with the ISIS implementation because of the nature of the application, rather than the limitation of the ISIS computing model.

A Client-Server Model with Service Brokers

The master-slave process model imposed by AERO is not desirable if we want to build systems with communicating intelligent agents. After evaluating Sun's ToolTalk™ [ToolTalk91] and Cornell's ISIS [ISIS90] for an alternative distributed computing model, we decided to build dClips on top of ISIS. ToolTalk was not chosen because: 1) the message arrival sequence from multiple senders in a network environment is not guaranteed, 2) only 1 handler is allowed for a request message (others are observers). Message arrival sequence is important because the arrival sequence of messages for asserting a fact or for updating object instances in the dClips environment is critical to the local Clips inferencing process. Different message arrival patterns could result in different inference outcomes. Furthermore, the constraint of having a single handler for a request message makes it unnatural for task distribution/assignment in a parallel programming environment.

On the other hand, ISIS, developed at Cornell University, provides a set of tools built around virtually synchronous process groups and reliable group multicast [Birman91]. A virtually synchronous distributed system has the following characteristics: (1) all processes observe events in the same order (global order and causality), (2) an event notification is delivered to all or none of the audience (atomicity). A virtually synchronous system looks synchronous to every process in the system, but executes asynchronously. For the dClips implementation, the virtually synchronous broadcast (cbcast, for causal broadcast), which guarantees the causality and atomicity, was the main reason for using ISIS as the underlying distributed computing model.

Figure 3 shows the architecture of ISIS-based dClips, where a set of dClips Server processes team up with a dClips Administrator process to form a process group. This process group provides the cooperative problem solving capability to the outside world. The dClips Administrator plays the role of a *service broker*, providing a consistent interface to the outside clients, while the details of the server processes are transparent to the clients. The interface between a service broker and its clients has yet to be defined. At this point, the CORBA (Common Object Request Broker Architecture) IDL (interface definition language) type interface is being considered [OMG91]. The interface between the dClips Administrator and the dClips Servers is a shared knowledge base with a set of common access methods.

The dClips Servers form a conceptual hierarchy, which is known to the dClips world, but is not visible to the ISIS environment. In other words, this Server hierarchy is not a hierarchy of ISIS process groups. In the ISIS environment, all the servers are equal members of a single process group. Broadcasts to the group will reach every server process in the same order. Each server is an autonomous problem solving agent with its own knowledge base and its own task. The Server hierarchy defined within dClips environment helps a server to find another potential problem solver if a problem cannot be solved locally.

A shared knowledge base is available to dClips Servers for knowledge exchange and interaction, which is designed to facilitate the cooperative problem solving process conducted by multiple dClips Servers. At the same time, each dClips server can have its own individual non-shared knowledge base. The Server hierarchy is defined as a Clips Class Hierarchy within the shared knowledge base, which is known to every server. The message communication between servers can be: (1) a broadcast to the whole process group, or (2) a message to a designated server.

The shared knowledge base is realized by having a set of Clips constructs replicated in each server. Each server loads in this shared knowledge base at initialization time. Any update to any of the objects in this shared knowledge base in any dClips Server will trigger a broadcast of the update to other members in the process group. A server applies the updates sent in from other servers one by one as if they are local updates to the knowledge base. Since the shared knowledge is designed to keep only the critical knowledge that needs to be shared among servers, the size of this shared knowledge base should be small. The effort to keep it consistent across multiple servers, i.e., sending and receiving update messages and applying updates triggered by remote update messages, should be minimal.

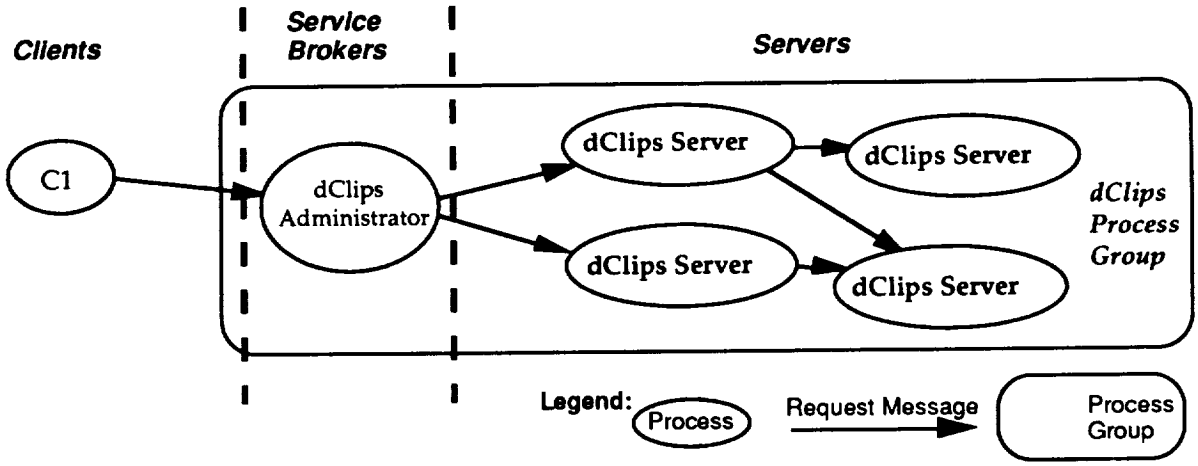

Figure 3. Architecture of ISIS-based dClips

Future Opportunities

Based on the client-server model of dClips, we would like to pursue the following extensions:

i). dClips with Database Access Capability — This involves a dClips database gateway, which runs as a database client to some database server. An intelligent agent can not be intelligent without necessary knowledge about the real world. Accessing existing databases is one way of acquiring data/knowledge from the outside world. As shown in Figure 4, a database pass-through process can serve as the gateway to the database server. The function of this gateway can be as simple as passing a SQL statement to a relational database system and receiving the results back in a buffer. Or it can provide more sophisticated functions such as allowing joins of tables across multiple database systems.

ii). A distributed blackboard system on top of dClips — The ISIS-based dClips implementation can easily evolve into a distributed blackboard system. This can be done by making dClips server processes run as *knowledge sources* in a blackboard system and by

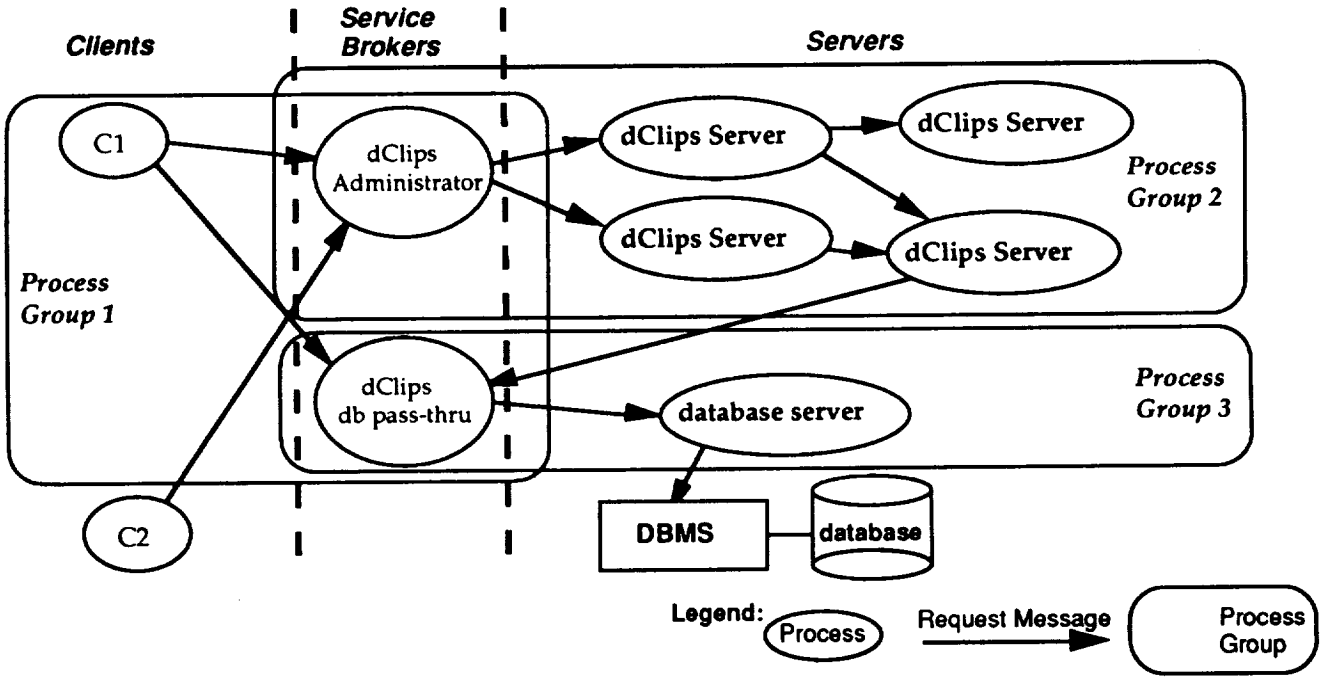

Figure 4. dClips with Database Access Capability

using the shared knowledge base among dClips servers as the *blackboard* [Nii86]. A blackboard system like this is a realization of the original blackboard metaphor because there is no centralized control mechanism involved in the blackboard reasoning process. Each knowledge source reacts only to the change on the blackboard. A domain problem can be solved cooperatively this way by multiple knowledge sources.

References

[Birman91] Birman, K., Schiper, A., Stephenson, P., Lightweight Causal and Atomic Group Multicast, ACM Transactions on Computer Systems, Vol. 9, #3, August 1991.

[Clips91] Clips, Version 5.1, Reference Manual, Software Technology Branch, Johnson Space Center, Sep. 1991.

[Handley92] Handley, T., Li, Y. P., DataHub: Knowledge-based Data Manage-ment for Data Discovery, ISY Conference on Earth and Space Science Information Systems, Feb. 10-13, 1992.

[ISIS90] ISIS, Version 2.1, User's Guide and Reference Manual, Cornell University, Sep. 1990.

[Nii86] Nii, H. P., Blackboard System: The Blackboard Model of Problem Solving and the Evoluatioin of Blackboard Architectures, THE AI Magazine, Summer, 1986, p. 38-53.

[OMG91] Object Management Group and X/Open, Common Object Request Broker: Architecture and Specification, December 1991.

[Sullivan89] Sullivan, M., Anderson, D., Marionette (Also Known As Aero): a System for Parallel Distributed Programming using a Master Slave Model, IEEE 9th International Conference on Distributed Computing Systems, 1989.

[ToolTalk91] TookTalk, Version 1,0, Programmer's Guide, SunSoft, Dec. 1991.

DATA VISUALIZATION AND SENSOR FUSION

Dr. W. Vance McCollough
Hughes Aircraft Company

August 4, 1993

DATA VISUALIZATION AND SENSOR FUSION

Introduction and Outline

HUGHES
UNIVERSITY OF CHICAGO
FRANKLIN MEDICAL INSTITUTE

- **Objective**
  - Application of Medical Imaging and Visualization Techniques to Remote Sensing
- **Investigators**

University of Chicago	Hughes
Dr. Chin-Tu Chen	Dr. W. V. McCollough
Dr. X. Penn	Dr. R. C. Savage
Dr. M. Warnick	

- **Progress**
- **Discussion**
- **Planned Work 1993- 94**

DATA VISUALIZATION AND SENSOR FUSION

Progress To Date- Summary

HUGHES
UNIVERSITY OF CHICAGO
FRANKLIN MEDICAL INSTITUTE

- **DMSP Data Set Selected for Study**
- **Image Processing and Visualization Algorithms Developed and Tested on CAT and PET Medical Data**
  - Image Matching Using Landmarks
  - Contouring and Thresholding
  - Image Linking Using Above Techniques
  - 3-D Image Processing/ Visualization and Superposition of 2-D Images
  - Image- Indexed Color Table Generation

DATA VISUALIZATION AND SENSOR FUSION

Characteristics of Remote Sensing Data- DMSP

HUGHES
UNIVERSITY OF CHICAGO
FRANKLIN MEDICAL INSTITUTE

- **Concurrent Data Availability- OLS (Vis & IR); SSM/I- Microwave**
- **Spectral Bands- 7 Microwave, 1 Visible, 1 IR**
- **Resolution- Varies from 50 km to 0.5 km**
- **Several Scan Geometries (OLS- linear; SSM/I- Conical)**