Using groups to support the interconnection of parallel applications

Pedro D. Medeiros, José C. Cunha
Departamento de Informática – Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Portugal

Abstract

Among the efforts of building heterogeneous applications through the interconnection of independently developed components, we have proposed a group-oriented approach called PHIS.

In this paper we identify several types of interaction between application components and show how a group-oriented approach eases the modelling of these interaction patterns. A brief description of PHIS primitives is made and through the use of three examples we show how the PHIS system supports the group-oriented approach.

A comparison is made between several related systems and PHIS and its distinctive characteristics are highlighted. Our experience in the building of some medium sized application like a parallel genetic algorithm execution environment supports the claim of the power and flexibility of PHIS as an interconnection model.

1 Introduction

In many areas of Science and Engineering, the simulation of physical processes has so big demands in terms of processing power and data storage that parallel processing is the only viable alternative. The development of parallel applications presents many difficulties comprising the decomposition of the problem in multiple concurrent activities, the specification of their interactions and the mappings to the parallel architectures. These difficulties impose long development periods for getting moderate gains in performance.

One approach to overcome these difficulties is to build the application through the interconnection of multiple and possibly heterogeneous components. There are three main reasons to this:

- **functional decomposition** to capture the functional decomposition of the application;
- **divide to conquer** it’s easier to reason in terms of the composition of the behaviour of simpler components
- **software reuse** allowing the reuse of already developed software

1.1 Motivating examples

In the following we describe two examples of the building of an application by the interconnection of two or more components:

The first one is computational steering
• One or more computationally intensive components that simulate some physical process where some parameters of the simulation can be modified;

• A visualization component supports on-line data visualization.

• An interactive control component allows modification and inspection of the parameters of the simulation(s).

Reference [16] describes an application where two components are used. The reason for such a decomposition is that the physical processes of the two simulators suggest a different programming model for each component. This poses the need for a suitable interconnection model.

1.2 A communication infrastructure for component data exchange

The ability of reusing pre-existing components and tools in the development of parallel applications demands a communication infrastructure in order to transfer data between the components of the application.

The main requirements of this interconnection service are:

1. To support component interconnection without requiring a major rewriting of each component.

2. To be compatible with the distinct computation models that may appear in each component.

3. To provide an adequate degree of independence regarding the hardware and runtime environment.

If all the components use the same programming model (e.g., a message-passing system like PVM or MPI), this system can be used to interconnect the components. This has the advantage of not introducing a new programming model but has some limitations which are explained in section 5.

If all the components do not use the same programming model, the building of such an infrastructure requires one of the following solutions:

Use of an already existent message passing systems One of already deployed interprocess communication mechanism (like TCP/IP sockets, PVM or MPI) could be used to insert code in the components every time they must exchange data.

Common runtime system There are some efforts to build a low-level runtime system like Converse [17] or Nexus [14] which can be used to implement different interprocess communication mechanisms. One of these runtime systems could be used to implement all the communication mechanisms. This would allow the interoperability of all the components.

A separated system dedicated to component interconnection This alternative has the advantage of allowing the system to be designed specifically for the role of component interconnection. In section 5 several of these systems are mentioned.

This last option is a more general solution and is the more adequate to the above requirements of an interconnection system. We have followed this approach in the development of the PHIS system that we discuss in this paper.
1.3 PHIS: a group-oriented interconnection system

The PHIS system provides a communication and synchronization service between the components of an application. PHIS only supplies communication and synchronization primitives and does not support other type of services like the spawning of the components.

PHIS defines a simple interconnection model (see figure 1) based on the following entities:

**Component** which corresponds to a program which executes in a virtual machine

**Interconnection messages** which support the data exchange between components and are distinct from the messages exchanged within each component.

**Process groups** which are used for collective message passing and synchronization support.

This group-oriented approach, as we explain in the following, allows great flexibility in the component interconnection and the easy support of the most common component interaction patterns.

1.4 Overview of the article

The paper is organized as follows. In section 2 an identification of common interconnection patterns is made. An overview of the PHIS system is made in section 3. In section 4 we show how PHIS can be used to support the types of interconnection described in section 2. A comparison between PHIS and other related systems is made in section 5. In section 6 we discuss the relation of PHIS with the standard approaches to component interconnection e.g. as the ones based on CORBA.

2 Types of component interaction

The first part of our work on the development of the PHIS model has been centered on the identification of the required functionalities for the interconnection of heterogeneous application components and tools. That is why PHIS design has been largely driven by our experimentation
with applications requiring distinct forms of parallel processing, data and performance visualization, and user interaction and dynamic program control. This work allowed us to identify the following common component interaction patterns:

**Shared structures:** In this pattern components interact through a logically shared structure like a database or a blackboard. It can be associated with a form of implicit invocation of remote operations by the detection of changes in the shared structure.

**Client/server:** This pattern identifies an asymmetric interaction between components: one that offers a service and waits for an invocation from the client through a message.

**Pipe/filter:** This pattern supports data streams between components called filters. These filters perform transformations on input data producing the modified data on its output.

**Symmetric:** This pattern captures the interaction between components that occurs in the simulation of two different activities by two different components, where data must be transferred in the boundary. It can be considered a special case of the pipe/filter interaction pattern where corresponding processes in the two components establish a bidirectional stream to transfer boundary data.

**Publish/subscribe or event-based:** In this pattern components declare a set of procedures and a set of events. Components generate events that trigger the execution of the procedures in the components. This pattern is sometimes associated to a “message bus” where messages of specific types are published by components and are delivered to other components that have subscribed to them.

This also agrees with a classification presented in [2]. For illustration purposes, we selected the last three types of interaction. The pipe/filter is included in the symmetric pattern and the shared structure pattern can be easily simulated through the availability of a publish/subscribe mechanism.

### 2.1 Client-server interaction

An example of this kind of interaction exists when a parallel component that simulates some physical process is connected to a component that supports the visualization of data produced by the simulation component. The relationship between the two components is asymmetric and corresponds to a client—server model (see figure 2).

### 2.2 Symmetric interaction

This corresponds to a situation where we simulate a physical process that occurs in a system composed by two or more parts, and in each part the simulation is based on a different mathematical model. The simulation includes the interaction in the boundaries. One example of this situation is in the area of weather forecast [12] where we have a simulation of ocean and of the atmosphere that exchange values in the boundary (sea level). The information exchange in the boundary demands that each boundary process of one simulation can address the process that is responsible by the same area in the other component. This example also suggests the need of synchronization operations that involve processes from different components (see figure 3).

### 2.3 Publish/subscribe interaction

As an example of this kind of interaction we show a situation where different kinds of visualizers are dinamically connected/disconnected to a computation-intensive application (see figure 4).
3 Overview of PHIS

The main concept in PHIS is the component which corresponds to a sequential or parallel program that runs over a virtual machine. A PHIS application program is built by interconnecting multiple components.

We assume that the virtual machine associated with each component can be supported on top of a heterogeneous network of workstations, a distributed-memory multiprocessor or a shared-memory multiprocessor. Regarding the programming model, we currently restrict our system to components that rely on a message-passing library.

The application builder must have access to the source code of the applications and must be able to introduce some modifications. These modifications are small and confined to well defined parts of the code.
3.1 PHIS process groups

The main concept for the integration of distinct components in the PHIS interconnection system is a specialized form of process group. Groups are open [19] allowing external processes to address them as a single entity, and correspond to a collective address. Multiple processes from distinct components may explicitly join a group. This provides a very general form of component interaction where the programmer must identify, in each component, the processes that will interact with the other components. Groups are dynamic and by default each application component defines a group which includes all its processes. This addresses the very common situation where a component behaves like a server, e.g. for data visualization, where its internal composition is hidden from the clients. This is summarized in the figure 5.

PHIS communication and synchronization is based on process groups and this approach is central to the PHIS approach as it captures

Application structuring: Each component corresponds to a group and all its internal processes the processes can be addressed as a unique entity;

Communication: Point to point and collective communication allows the support of data stream and client/server patterns;

Asynchronous events Synchronous and implicit reception of messages supports the publish/subscribe pattern;

Synchronization Support of component synchronization through barriers and primitives associated to the group membership;

Figure 4: A publish/subscribe interaction
Reconfiguration Groups are dynamic, allowing the modification of the interconnection structure during the execution.

3.2 Modifications made to the source code

The modifications made to the source code of each component should concentrate on the following parts:

- On the application initialization phase, the integration of each process into the relevant groups is required.
- Where appropriate, the sending and handling of interconnection messages.
3.3 PHIS primitives

A simplified description of the PHIS primitives follows:

Component and group related:

<table>
<thead>
<tr>
<th>Initialization</th>
<th>PHIS_Init( +ComponentName )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Termination</td>
<td>PHIS_Finalize()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Joining a group</th>
<th>PHIS_Join( +GN, +HandlingRoutine )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaving a Group</td>
<td>PHIS_Leave( +GN )</td>
</tr>
<tr>
<td>Freezing a group</td>
<td>PHIS_FreezeGroup( +GN, +NrOfElements )</td>
</tr>
<tr>
<td>Barrier</td>
<td>PHIS_BARRIER( +GN, +NrOfElements )</td>
</tr>
<tr>
<td>Waiting for a group</td>
<td>PHIS_WaitForGroup( +GN, +NrOfElements )</td>
</tr>
<tr>
<td>Group existence</td>
<td>PHIS_GroupAvailable( +GN, +NrOfElements )</td>
</tr>
</tbody>
</table>

Sending and receiving messages:

<table>
<thead>
<tr>
<th>Buffer initialization</th>
<th>PHIS_InitPackBuffer( -bufId )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer filling</td>
<td>PHIS_PackXXX( +bufId, + data )</td>
</tr>
<tr>
<td>Message sending</td>
<td>PHIS_Send( +GN, +Member, +bufId )</td>
</tr>
<tr>
<td>Message availability</td>
<td>PHIS_MessagesAvailable( +GN, -res )</td>
</tr>
<tr>
<td>Message reception</td>
<td>PHIS_Recv( -bufId )</td>
</tr>
<tr>
<td>Getting info</td>
<td>PHIS_Info( +bufId, -GN, -Member )</td>
</tr>
<tr>
<td>Extracting data</td>
<td>PHIS_UnpackXXX( +bufId, - data )</td>
</tr>
<tr>
<td>Freeing the buffer</td>
<td>PHIS_FreeBuffer( +bufId )</td>
</tr>
</tbody>
</table>

Messages are delivered to all processes in the destination group. A default handler in each process is responsible for enqueuing the incoming messages, for later processing, when the PHIS_Recv() is invoked. Alternatively, an user-defined message handler can be specified.

3.4 Status of PHIS implementation

PHIS has been implemented in several types of parallel architectures namely networks of workstations connected by Ethernet or by a FDDI ring. Implementations also exist for Transputer-based multicomputers and for heterogeneous configurations of these multicomputers and networks of workstations. Details about the different implementations can be found in [20] and [21].

As shown in [21] PHIS performance is comparable to MPI and PVM in the same hardware and compares favourably with some similar systems like PLUS[5].

4 Examples of PHIS use

We show how PHIS supports each of the types of interconnection described in section 2.

4.1 Client-server interaction

As an example of the client-server interaction, we present a case where a parallel PVM performs several simulation steps and interacts with a parallel visualization component using PHIS primitives (see figure 6).
4.2 Symmetric interaction

Figure 7 illustrates a symmetric interaction between two parallel simulators of the physical process of heat diffusion through a wire. Each process calculates a set of points and uses a one-dimensional finite-difference method. PHIS is used to exchange values in the boundary. The “rightmost process” of the PVM simulator and the “leftmost process” of the MPI simulator join a group named ’boundary’.

4.3 Publish/subscribe interaction

The example shown in figure 8 is the observation of a simulator component by a performance visualisation component and an application-specific visualiser. The publish/subscribe interaction is achieved through the membership of the group “visual”.

Figure 6: PHIS in a client-server interaction
5 Related work

We concentrate in components that are built using message-passing libraries and we divide our analysis in two parts:

- the interconnection of parallel applications written using the same programming model (for example PVM or MPI);
- the interconnection of parallel applications written using different programming models

5.1 Interaction of homogeneous components

Current message-passing systems like PVM and MPI provide limited facilities for the interconnection of independently-started components.

For example, in MPI, the MPI intercommunicator mechanism [11] allows the linking of processes in two different groups that may communicate using send and receive calls. This is supported by having all the processes in both groups invoking a collective call that establishes
the interconnection: each group must specify its own local communicator, a local leader process, an agreed remote leader process, and an agreed context for leader-to-leader communication. Additionally, a parent communicator must be specified containing all the processes in both groups (by default this includes all processes in the MPI configuration).

Regarding MPI we must mention the PVMP1 [10] project where PVM process groups are used to interconnect independently-started MPI components and the part on interoperability of the MPI-2 standard [22]. In this case a facility similar to the connection-oriented sockets was
integrated in the MPI world.

Concerning PVM, until version 3.3 its use as an interconnection model would require the rewriting of major parts of the application code when message tag conflicts could arise among separately developed components. This has changed in PVM3.4 [15] where the concept of communication context has been introduced; this concept allows separately developed applications to define distinct contexts where tags can be used without conflicts.

5.2 Interaction of heterogeneous components

The support of the interoperability of different parallel codes has been the goal of several projects like I-Way [13] CAVEComm [9], CUMULVS [18] and WAMM [4]. A pioneering effort has been the Schooner system [6] which proposes a RPC-based model.

Some of the above systems are too connected to a particular message-passing system (LAM with MPI, CUMULVS and WAMM with PVM), or don’t have a high level interconnection model (for example I-Way suggests the use of TCP/IP sockets for the interconnection of components).

An effort closely related to PHIS is the PLUS library [5]. This library is used to interconnect parallel programs that were written using different message-passing systems. As in PHIS, some minor rewriting of each component is necessary. The main difference between PLUS and PHIS is that in PLUS the devices that are used to communicate with external processes to each component are mapped into devices of the underlying message-passing system (e.g., if the component uses PVM, external processes are identified by PVM task identifiers). This has the drawback of making the porting of PLUS to some environments non-trivial (for example, it is not clear how PLUS is adapted to the static model of processes of MPI).

6 Comparison with CORBA-based approaches

At first sight it would appear that the use of standard specification for component interaction such as CORBA would be the best solution.

However the efficiency of CORBA precludes its use to support the transfer of high volumes of data between components. This issue is illustrated by several works, namely the TENT component integration framework [1] [3], where a special solution for data exchange shortcuts the CORBA-based interactions (which are only for control).

7 Conclusions and further work

In this paper we showed the relevance of PHIS-like systems in the context of the development of parallel applications.

Comparing PHIS with related approaches we can conclude that PHIS has very acceptable performance and that is process group orientation has proved to be a powerful interconnection programming model. This claim is based on its use in medium sized applications like a parallel genetic algorithm execution environment which is described in [21], [8] [7].

Acknowledgements

This work was partly supported by the CIENCIA and PRAXIS XXI (project PROLOPPE) Portuguese Research programmes, the EEC Copernicus and TEMPUS programmes and DEC EERP PADIPRO project.
References


