MEDICOS: An MDO-Enabling DIstributed COmputing System

Shen Yi Jin[0], Karp Joo Jeong, Jae Woo Lee, Jong Hwa Kim, Yu Xuan Jin
College of Information and Communication, Konkuk Univ., Seoul, Korea

MDO를 위한 분산 컴퓨팅 시스템

김신의[0], 정갑주, 이재우, 김종화, 김우현
건국대학교 정보통신대학
skim@imc.konkuk.ac.kr jeongk@konkuk.ac.kr

Abstract

This paper presents a computing system, called MEDICOS, that enables *Multidisciplinary Design Optimization* (MDO) technology for engineering design on distributed environments. In MDO, various legacy softwares have to be integrated, so dynamic configuration and seamless coordination between these legacy softwares must be supported. MEDICOS is designed to address these issues by the Linda shared memory model-based design and the agent-based wrapper technology. A prototype system for engineering designs is developed and tested with designing a super high temperature vacuum furnace.

1. Introduction

Multidisciplinary Design Optimization (MDO) is crucial for most engineering design because it allows us to achieve a global optimal design decision among mutually-conflicting local design decisions. The challenging problem with MDO is how to coordinate or integrate various types of commercial and research prototype software systems on heterogeneous platforms to interact towards a globally optimal solution. In MDO, most of these systems are based on legacy softwares. There are a number of distributed computing systems for system coordination and integration, but they are not designed for the support of legacy software in an efficient and flexible way. MDO-Enabling DIstributed COmputing System (MEDICOS) is such a framework that addresses these issues by shared memory model-based Linda system and agent-based wrapper technology.

Linda is a memory model (called tuple space) that consists of a collection of logical tuples. The Linda model does not care how the multiple execution threads in a Linda program what they compute, it deals only with how these execution threads are created, and how they can be organized into a coherent program[1]. This feature for Linda model solved distribution, platform and language independence problem for us. Components in MDO framework are developed in various computer languages and are widely spread on the network. By taking advantage of the Linda model, existing legacy and commercial softwares can be easily integrated.

Another core technology for MEDICOS is the agent-based wrapper technology. Wrappers encapsulate various legacy or commercial softwares by a well-defined interface to the system (especially, the middleware used in MEDICOS) and invoke them on request.

As mentioned above, much effort has been made before and many well-known MDO frameworks have been developed up to now. They include iSIGHT, IMAGE, and ModelCenter. Common drawbacks with these frameworks are monolithic-based design in which distributed computing facilities and MDO supports are tightly-coupled. In the MEDICOS design, there is a well-defined interface between MDO supports and distributed computing middleware, and a commodity middleware such as Java RMI and JINI can be used in stead of the current Linda system.

2. Overview of MDO

MDO is a methodology for the design of complex engineering systems and subsystems that coherently exploits the coordination of mutually interacting phenomena[2]. In most engineering design works, optimal design decision is required among local decisions from sub-design work which are often mutually conflicting (e.g., to pursue an optimal decision for a certain local design causes performance decrease in other local designs). For example, in aircraft design, local designs including propulsion analysis, heat conduction and convection analyses wants to get to their local optimum value while a global optimum solution is required for achieving the global optimum decision for the entire design. The MDO framework supports implementation and execution of MDO applications and provide a set of support services commonly needed in applications of this type[3]. Four kernel technologies are comprehended in MDO framework. They are MDO technology, problem-specific technology, distributed computing technology and system integration technology.

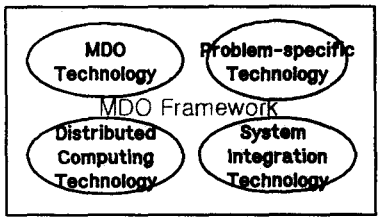

Figure 1: MDO Framework

Problem-specific technology corresponds to diverse legacy or commercial softwares for the specific engineering design. MDO technology, including object functions, optimize method and constraints, is used to achieve final optimum solutions. Distributed computing technology is a independent IT technology such as distributed programming, remote procedure invocation and parallel processing. The last technology, system integration technology is an interdisciplinary technology that makes the diverse software integration possible and data transmission controllable. The challenge we are facing is to support a framework that synthesis these four technologies into an entire one. The technical detail solutions we used will be discussed in the next section.

3. System Design

3.1 Design Approach

In our design approach, we use a fault tolerant version of the shared memory-based Linda model, called Persistent Linda (hereafter, shortly PLinda), whose implementation is based on C++[4]. The Linda model allows loosely coupled system integration which facilitates dynamic configuration. In Linda, system components can be easily integrated.

In addition, we use an agent-based wrapper technology that turns legacy software systems into integratible system components. It provides mechanisms for invocation, input preparation, output postprocessing, and data structure transformation.

3.2 Components

Major components of MEDICOS are: analysis modules, optimization modules, CAD modules, DBMS modules, visualization modules and GUI modules. Among these modules, the GUI module is a little different from the others, because for each specific product design. GUI will be newly developed while other modules are relatively fixed. This is a hint that GUI may have a certain "knowledge" of the framework and may embed some requisite functions. Wrappers shelter softwares to be seamlessly integrated and invoke each modules at any requested time. A management toolkit provides a clear communication channel, monitoring and scheduling that addresses dynamic configuration and automatic data transference.

3.3 Architecture

MEDICOS has a strict layer-based architecture. Figure 2 shows the global architecture at the view of layer:

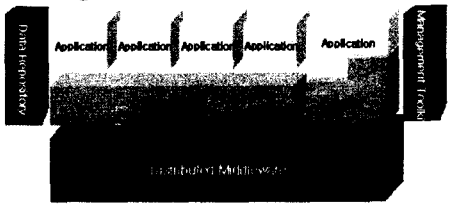

Figure 2 : Layered Architecture

MEDICOS consists of three layers: distributed middleware layer, wrapper/agent layer and application layer. The distributed middleware layer based on the PLinda supports loosely-coupled interactions among system components. Due to the Linda model, these interactions are independent of. component locations, invocation times, and underlying platforms.

The wrapper/agent layer performs the following: (1) make transformation between XML-based standard data formats in the shared memory and component-specific data in working directories, (2) prepare input files and invoke legacy software components, (3) detect the termination of the execution. Wrappers and agents for different system components communicate with the distributed middleware[5].

Figure 3 shows the runtime structure of the system.

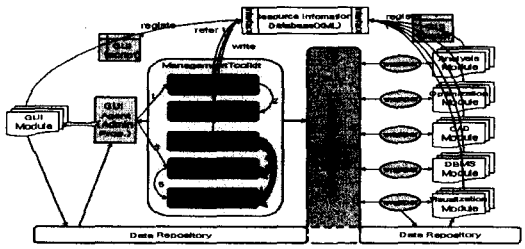

Figure 3 : Runtime Structure

In this structure, Resource Information Database manages the detailed management information of each module which is accessed via Register GUI. This information enables us to control each module[6]. In Management Toolkit, resource monitoring, automatic wrapper generation, problem formulation, scheduler and ID assignment services are provided. It also controls data flow and selects correct data from data repository by referring to the Resource Information Database.

3.4 Major System Operations

The major operations of MEDICOS is: module registration, wrapper manufacture, problem formulation and final optimization and analysis result retrieve.

registration

All the modules (resources) that are willing to be integrated must registrate to the resource information database through an registration GUI beforehand. The

registrated information is in the form of XML documents. For example, the analysis module output file XML is like below:

```
<output type = "file">
 <numOfFile = "1">
  <file name="out_1.dat" location="D:₩data₩output" size="2M" />
  <template name="out_01_tem.dat" location = "C:₩template" />
 </numOfFile>
</output>
```

wrapper manufacture

Users can monitor the resources registered on the system and chose the module that they wants. Once a module is selected, the resource monitor enable the Wrapper Manufacturer to make the wrapper for the selected module automatically by referring to the Resource Information Database.

formulation

Since modules are selected, users can formulate problems by dragging and drawing from graphic user interface. Basically, parallel processing is supported by PLinda in our system. Formulation Script enables the ID Assigner to assign process and data ID and generate the schedule by Scheduler that proceed the formulation.

data retrieve

When outputs are generated, final results can be retrieved by visualization module or GUI module. During the procedure time, all the data are safely kept in the Data Repository and controlled by the Management Toolkit that tells the wrappers and agents the correct data needed.

4. System Implementation and Experiments Data

In the implementation, we developed a prototype of a *super high temperature vacuum furnace.* In this prototype development, GUI module, analysis module, optimum module, DBMS module, CAD module and visualization module are the working entities that create real data for this system. For some reason, registration and parallel processing are not supported yet. Persistent Linda works as the distributed middleware. The modules are located independently in a distributed environment and integrated into MECODIS. The DB-Optimize working interface is as below :

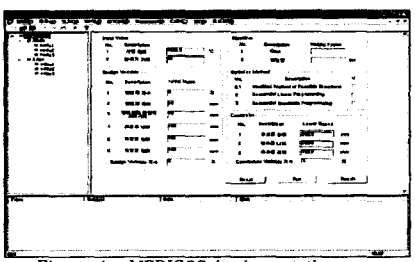

Figure 4 : MEDICOS implementation

During implementation, we found concurrency problem and solved it by the state check mechanism.

When a resource is required while it is in use, it is needed to allocate the resource for every user without conflict. The solution we used is to check the state of the module each time it is required by users. That is, a resource state check process is supplied. The state of the required module is informed to users. So they can decide to use the resource at present or not.

The experiments are made based on MEDICOS. The final results visualized by Formula One are shown in figure 5:

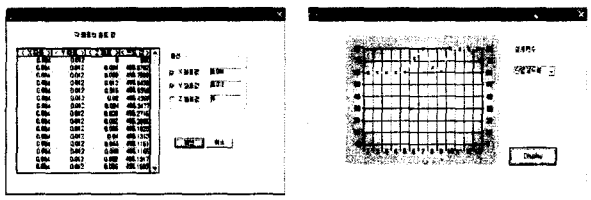

Figure 5: optimum result - by data grid and graph

5. Conclusions and Future Work

We have designed an MDO-enabling distributed computing framework and implemented a prototype for it. In our system, legacy and commercial softwares are seamless integrated and communicate unimpededly. During implementation, we found some problems either.

First of all, security problem is not concerned throughout the design and development process. Because it is difficult to judge reliability of legacy codes. So we assume that all the modules that in MEDICOS is safe and reliable.

Then, in the current system, resource registration is not implemented either. As the result, wrappers are not automatically manufactured but are coded by developer directly and enabled manually. That is, the dynamic configuration is not addressed yet.

What we are going to concentrate upon is solving these problems concerned above and make MEDICOS an essential tool for engineering design.

Reference

[1] Nicholas Carriero and David Gelernter How to write parallel programs: a first course Massachusetts Institute of Technology, second printing, 1991

[2] http://mdob.larc.nasa.gov/

[3] A. O. Sales and J. C. Townsend Framework Requirements for MDO Application Development AIAA-98-4740 1998

[4] Karpjoo Jeong, Bin Li, Denis Shsha, Peter Wyckoff PLinda/C++ and Fortran77 User's Guide p8 1996

[5] Joon Sang, Yoo Development of a Collaborative and Distributed Computing for Multidisciplinary Design Optimization p 15 Konkuk University 2003

[6] Davis S. Linthicum Next Generation Application Integration p207 Addison-Wesley