PLATON — A PROBLEM SOLVING ENVIRONMENT FOR COMPUTATIONAL STEERING OF EVOLUTIONARY OPTIMISATION ON THE GRID

Markus Krosche, Rainer Niekamp and Hermann G. Matthies

Institute of Scientific Computing
Department of Computer Science
Technical University Braunschweig, 38092 Braunschweig, Germany
e-mail: wire@tu-bs.de, web page: http://www.tu-bs.de/institute/WiR

Key words: PLATON, Grid Computing, Numerical Optimisation, PSE, Computational Steering

Abstract. PLATON is a Problem Solving Environment (PSE) for distributed numerical optimisation that allows the construction and steering of large-scale optimisation processes on a computational Grid. It is also an environment for coupling optimisation and simulation software. Evolutionary Optimisation is a good example of a problem which can efficiently designed to run on a Grid environment. After a short introduction to PLATON, we focus on the generic interface description used to describe the software components and couple them together, the tuple-space concept of distributed computing and the functionality of our object oriented generic communication template library, which uses standard low-level communication tools like PVM and MPI, but hides the complexity of their actual use.
1 INTRODUCTION

PLATON is a Problem Solving Environment (PSE) for distributed numerical optimisation that allows the construction and computational steering of large-scale optimisation processes on a computational Grid. Here, a PSE is a complete, integrated computer environment for composing, and running applications in a specific area [6]. It hides implementation and system details to enable both application developers and users (scientists) to focus on their specific problems. Examples of other PSE’s are PETSc [2] and SCIRun [9]. In distributed computer environments there are many more problems than in normal stand-alone systems. Some of these problems are allocation of available resources, starting jobs on remote machines and handling security issues like authentication and authorisation of users and the delegation of rights. A computational Grid provides such an infrastructure, the so-called Grid middleware, where all these problems are addressed [5].

In our software prototype of the Grid enabled PLATON we use the Grid middleware for allocating resources in a secure way. The software system PLATON has a long tradition at the Technical University of Braunschweig. It is based on the concepts of its predecessor MEPO (Multipurpose Environment for Parallel Optimisation) and earlier version of PLATON itself [7].

PLATON has an abstract view of a function depending on design parameters, where values of the design parameters have to be found in order to minimise the function. Each function value is the result of a numerical simulation (e.g. a FEM-program) for given design parameters, and we call this abstract entity the “objective function” component.

In the same way the optimiser is seen as an abstract entity (the “optimiser” component) that changes the design parameters in order to achieve the minimum. The PLATON system provides the infrastructure for a seamless coupling of these components in a distributed environment.

The abstract optimiser is a software component which is connected with a concrete optimisation routine. During the runtime of the optimisation process the user can interactively change numerical parameters of a running optimiser and even change to a different optimisation routine for improving the result, thus having the opportunity to “steer” the process.

After interfacing the simulation the scientist can choose an appropriate optimiser from a collection of available optimisation routines or add his own favourite one. The system comes with an evolutionary strategy (ES) [1] and a genetic algorithm (GA) [12], the Nelder-Mead-Simplex method (Amoeba-Algorithm) [10], and a gradient based method. We are working on integrating more optimisers.

The system provides a scalable approach for parallel evaluation of populations on a computational Grid. The distributed PSE build on the general purpose concept of the tuple space for asynchronous object communication.

As an example the material selection of a simple mechanical system has been chosen to
show the efficiency of the approach. Here PLATON couples a complex FEM simulation
code Parafep [11] as the objective function with the provides optimisers.

2 REQUIREMENTS AND ARCHITECTURE

PLATON is designed as a distributed component system which acts on a computational
Grid. Each component provides a well defined interface specifying its functionality. In our
system a component specifies a service which can be used by other software components.
In this way the resulting software system PLATON can be seen as a configuration of
such individual reusable components. We want to use a remote method invocation (RMI)
mechanism to hide details of the distributed system.

One feature of the system and the underlying software-design is to provide interfaces for
users to easily interface both third party optimisation codes and numerical simulations
without worrying much about the complexity of the underlying distributed software-
system. At this point the use of Grid services is necessary to find and allocate remote
resources and to start remote processes on them.

In our system the objective function is a software which is coupled with PLATON.
The process of interfacing the simulation can be seen as the definition of the objective
function. The next step for the engineer is to specify the parameters to optimise. Here
PLATON defines a parameter concept. \( P = D \cup F \) is the set of parameters, composed
of the “design” parameters \( D \) which can be varied, and those which stay fixed in the
computation - the set \( F \), where \( D \cap F = \emptyset \). The objective function is a map from \( P \) to
the Reals \( \mathbb{R} \), \( f: P \rightarrow \mathbb{R} \).

The return value of the objective function is the objective value, and if \( f \) is differen-
tiable, the “objective function” component may also compute the gradient.

3 DESIGN AND REALISATION

PLATON builds up on a succession of predecessors, such as EVOBOX, EPOS, MEPO,
and early versions of PLATON itself [7]. The abstract view of components has evolved
in this history through the requirement of keeping optimisation, simulation codes, book-
keeping and distribution mechanisms apart. Figure 1 shows the UML interface or class
diagram of the system which will be described in more detail in the following sections.

4 COMPONENTS OF THE SYSTEM

The core system consist of a Grid tuple space component \( \text{GTupleSpace} \) for handling
asynchronous object communication, a resource allocation component \( \text{ResourceAllocation} \)
to hide details of the distributed system, an abstract optimiser component \( \text{Optimiser} \) as a
interface for optimisation routines, the objective function component \( \text{ObjectiveFunction} \)
which is the abstract view of the objective function, a simulation task component as an
interface for simulation codes \( \text{SimulationTask} \) and the system interface component
\( \text{SuperVision} \) for supervising the system.
PLATON is designed as an open system so that additional or new components can be integrated into the system, e.g., a visualisation tool, a database system etc. The following sections give a detailed view on each component.

4.1 The Object Values Component

ObjectValues is the central class of the core PLATON system. Every other component inherits from ObjectValues. The concept of implementing a steerable/browsable object is based on a name-space concept for the attributes of an object. In our system all components which inherit the ObjectValues interface can be accessed and browsed in the same way.

4.2 The Resource Allocation Component

The resource allocation component provides methods to make the use of resources transparent. This component has the responsibility to return the location (host and directory) of a remote service component and the way to start the remote service.

The Globus-Toolkit [5] is the industry standard Grid-middleware which provides tools to build and handle a computational Grid. PLATON use services of the Globus toolkit for resource-management and security issues like authentication and authority. But PLATON can also work without Globus. The ResourceManager component encapsulates the resource handling.
4.3 The Optimizer Component

The Optimiser component realises a so called abstract optimiser. In our system the
optimiser is seen as an abstract entity which varies design-parameter. The Optimiser
component provides access to all attributes of a special optimiser in a generic way by
ObjectValues.

In our system the optimiser component is associated with a “remote-pointer” or a
“stub” of the ObjectiveFunctions Components. So we had to define a function which fits
the requirement of the third party optimiser (the function pointer template). Inside this
function we can call any method of our ObjectiveFunction component. In this way we
have full control over the iteration loop of the third party optimiser.

4.4 The Objective Function Component

Each Optimiser needs a reference to one ObjectiveFunction to evaluate design pa-
rameters. The objective function component is responsible for starting SimulationTasks
to simulate design parameter.

4.5 The Simulation Task Component

A SimulationTask couples third party simulation codes. It fires an event when the
calculation of the objective value and, if possible and required, of the gradient is ready.
In this way it is also no problem to integrate a second simulation which is generated by
an automatic differentiation tool (AD-Tool) like ADIFOR [3], providing derivates (e.g.
the gradient) of the simulation.

In practice most simulation codes are written in FORTRAN or C and we provide a
so called SimpleSimulationInterface which allows a traditional procedural interface for
integrating simulation software.

The interface defines some “setter” and “getter” functions, which can be used to access
the ObjectValues components of a running SimulationTask.

4.6 The System Interface

The system interface component provides access to the system, e.g. a graphical user
interface (GUI) may use this interface component to provide access to the system in a
user friendly way [7].

5 COMMUNICATION MIDDLEWARE AND PARALLELISM

The abstract view has also been extended to the communication mechanism. While
earlier versions of the software uses PVM directly, the new version uses an abstract re-
move message invocation (RMI) mechanism for communication between the system com-
ponents, which itself may make use of PVM of MPI (or any other communication channel
for that matter).
5.1 The Communication Template Library

We have designed a Communication Template Library (CTL) as communication middleware. Similarly to CORBA [8] and Java-RMI the CTL provides a remote object communication mechanism design for high performance computing (HPC). But in contrast to CORBA there is no need to learn a new language to describe interfaces between system components (CORBA-IDL). In the CTL interfaces are described in C++ itself. The CTL is a software layer encapsulating different communication protocols. Currently MPI, PVM and naked TCP-IP are integrated into the CTL, but others like Globus-Nexus can be integrated. PLATON uses the CTL as the underlying communication middleware (Figure 2).

![Diagram](image)

Figure 2: The CTL as a communication middleware on top of low level communication protocols like MPI and as a service platform for applications

5.2 Parallelism in the System

The behaviour of the running PLATON system can be very complex. The simulations can themselves be parallel. This can be integrated into PLATON as a objective-function which can be used for evaluating populations in several instances (SimulationTasks), so that we have a new and higher level of parallelism on top of the simulation.

5.3 Scalable Parallel Approach — The Grid Tuple Space

Introduced for example in the Linda-System [4], a tuple space is an associative container for objects, which provides a simple interface for access using some basic methods like take, write, read etc.

In the system the optimiser writes populations to the tuple space. On the other side objective-function components take tuples from space for evaluation. After evaluation the result is written back to the space. There is no additional program dealing with load-balancing or check-pointing.
6 EXAMPLE

As “proof of concept” application, an example of the optimal material selection in a simple mechanical system is provided. This shows how complex FE-simulation codes can be integrated into an optimisation loop without much overhead, and — more importantly — without extensive changes to the simulation code.

6.1 The Mechanical Model

We choose a simple three dimensional load-bending beam as shown in Figure 3. The beam is modelled with an inner core and surrounding exterior part of different material. At the beginning of the optimisation process we choose homogeneous materials for both parts of the beam. The optimiser can vary the two material parameters “Young’s-modulus” and “Poisson’s ratio” of both parts of the beam. We start with homogeneous material parameters, with a very stiff material. We want to minimise the stress at a particular point in the exterior part on the clamped surface. We use Parafep [11] as the FEM simulation software.

6.2 Results

The result of the optimisation process is that the stress is minimal when the material of the beam is heterogeneous, the beam consist of a stiff core which carries the load and a softer exterior part which spread the load. We can see the same in biology, e.g. a branch of a tree.

7 CONCLUSION

The introduction of the ObjectValues component as a attribute container as a base class for all components of the system allows the introduction of steerable objects at runtime. The use of Grid technology enables the transparent and secure use of distributed resources in a standard way without worrying about the details of the remote machines (e.g. access policy like ssh, rsh, etc.).

We show that PLATON couples the abstract optimiser with a complex objective function (the FEM tool Parafep) through well defined interfaces. This makes it easy to interface PLATON with third party codes.
REFERENCES


