

# A Software Architecture for HPC Grid Applications

Steven Newhouse, Anthony Mayer and John Darlington

Department of Computing,  
Imperial College of Science, Technology and Medicine,  
180 Queen's Gate,  
London,  
SW7 2AZ,  
UK  
s.jn5,aem3,jd@doc.ic.ac.uk  
<http://hpc.doc.ic.ac.uk/environments/>

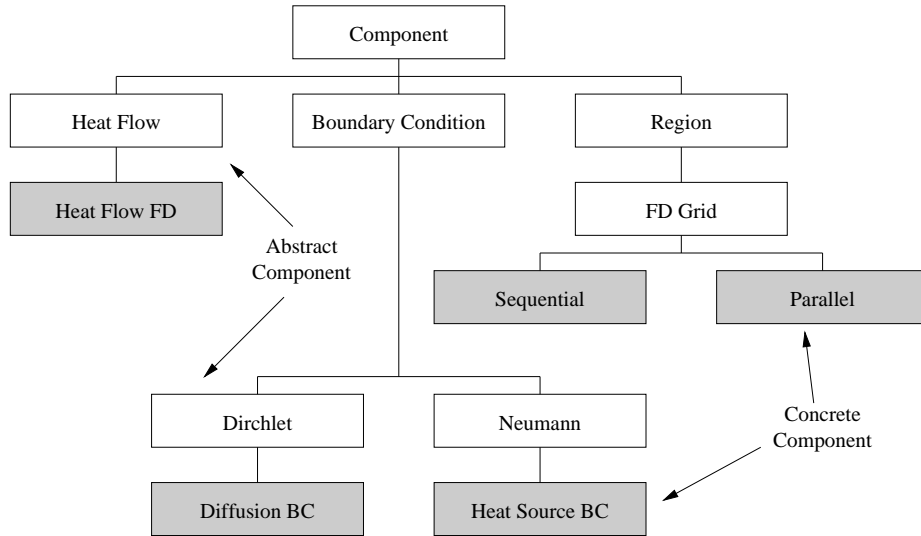
**Abstract.** We introduce a component software architecture designed for demanding grid computing environments that allows the optimal performance of the assembled component applications to be achieved. Performance over the assembled component application is maintained through inter-component static and dynamic optimisation techniques. Having defined an application through both its component task and data flow graphs we are able to use the associated performance models to support application level scheduling. By building grid aware applications through reusable interchangeable software components with integrated performance models we enable the automatic and optimal partitioning of an application across distributed computational resources.

## 1 Introduction

The emergence of local, national and international high-bandwidth networking allows physically distributed computing hardware resources to be linked enabling the development of 'computational grids'. These emerging grids present new challenges in automatic scheduling, application partitioning and resource management to deliver an effective computing environment. Effectively exploiting these dynamic and heterogeneous networking and storage resources requires automatic data partitioning to enable automatic run-time scheduling. Such scheduling only becomes feasible once performance information is integrated into the application.

We introduce a scientific component framework for HPC applications which provides relevant abstractions to the end-user, scientist and numerical programmer. Application performance is maintained through static and dynamic component optimisations which allow component implementations to be matched to the execution architecture. By adopting object and component based programming techniques we can deliver the functionality of skeletons through a conventional abstraction mechanism.

We present a simple example to illustrate these concepts and refer the reader to our complete paper (<http://www-icpc.doc.ic.ac.uk/components/papers/>).

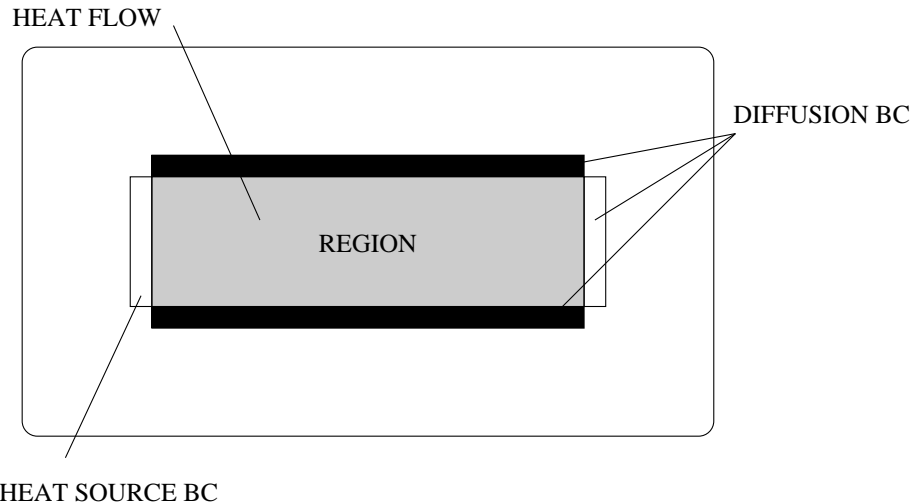


**Fig. 1.** Component Repository for a simple Finite Difference Problem showing abstract (white boxes) and concrete (grey boxes) components.

## 2 An Example: Heat flow in an Insulated Bar

We will illustrate this software architecture with a simple scientific example of heat flow in an insulated beam. A simplified component repository is illustrated in Figure 1 to demonstrate the implementation hierarchy and the use of abstract and concrete components.

1. **Problem Definition.** The problem is defined within a graphical PSE linked to a component repository (Figure 2). A component will have context dependent representations. It may be a code segment in an execution context, have a visual representation when being visually composed to build an application or have a three dimensional representation within the PSE.  
In the example, the Region component is selected from the repository, placed within the graphical PSE and manipulated to define the physical domain of the bar. Properties such as the boundary and initial conditions are attached to the edges and domain to characterise the behaviour of the Region.
2. **The Component Network.** The problem defined within the graphical PSE can also be expressed as a component network (Figure 3). This allows further non-graphical components to be added into the component assembly to complete the application which will be used to solve the problem. The parameters within the components can be adjusted to further define the physical problem.
3. **Valid Implementation Options.** The component network which has been defined by the user is validated for its correctness. In the heat flow example



**Fig. 2.** A graphical Problem Solving Environment being used to physically define the analysis by manipulating graphical components.

a boundary condition must be applied to each edge of the physical Region and one or more properties may be applied to the whole Region to give it some physical characteristics (eg. heat flow, elastic material, etc). These conditions are essential pre-requisites for any valid component network and can be incorporated at a low-level into the Region component.

The validated component network is compared to the available components in the repository. The static optimisation process described earlier is used to examine all of the feasible implementation options. In this simple example a choice has to be made between solving the problem using a sequential or parallel approach.

4. **Scheduling.** The valid implementation options are passed to the scheduler to match the application to a computational resource. This process uses the performance models derived from the assembled application and the performance of the computational resources to determine, within the constraints specified by both the user and resource provider, the resources needed to execute the application. In this example, the application's overall performance model, the target architectures and the problem size will show when it is appropriate to move from a serial to a parallel implementation. Further examination of the parallel performance model will yield the ideal domain decomposition for this problem size.

The effectiveness of cross platform scheduling can also be assessed as the component assembly will provide a profile of the computational requirements over time. If a matrix is being generated and then solved, it may be quicker to generate the matrix on a scalar parallel machine and transfer the data to a vector machine for the solution phase rather than execute both tasks

on the vector machine. The data flow and execution tasks graphs within the application, developed during the component composition process allow computing and networking performance models to be used to assess these alternative implementation options.

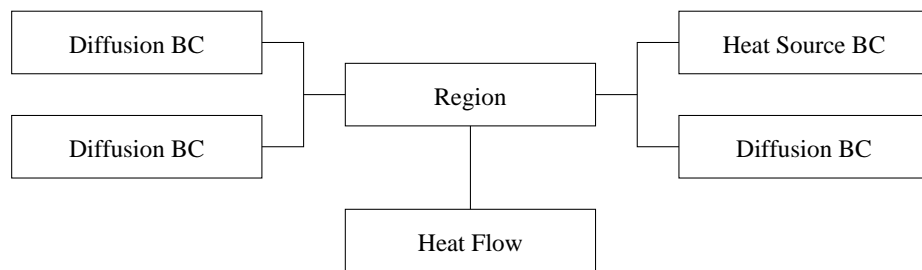
5. **Execution.** The component assembly is finally passed to the resources for execution. The execution and data flow graphs can still be exploited to examine the benefits of migrating the application to faster resources if they become available or the benefits of adding additional resources to the current calculation.

### 3 Conclusions

The proposed architecture, which we are developing and implementing across several applied scientific domains, is an extension of current or established research in compositional programming techniques. The skeletons work has already demonstrated the validity of using performance models to guide implementation and data layout decisions. We are able to exploit standard component models, such as Java Beans and CORBA, to contain the skeleton code fragments.

The repository will contain a variety of software components presenting usable abstractions to the end-user, the scientist and the numerical programmer. By defining standard interfaces, software will become easier to reuse between applications simplifying the development of large multi-disciplinary projects. By breaking the link between the problem definition and execution we are able to find the optimal implementation on the currently available computational resources.

Having built an application from components with integral performance models we are able to make sophisticated scheduling decisions and match an application to the appropriate computational resources. This will become essential for the emerging heterogeneous computational grids that will dominate high performance computing in the future.



**Fig. 3.** The Component Network extracted from the graphical Problem Solving Environment.