Numerical Libraries and the GrADSoft
University of Tennessee
August 2, 2001

This document presents a proposal of how the ScaLAPACK library interface and the interface of other numerical libraries can be organized into a general framework using the GrADSoft architecture. While some of the components and features discussed in this document are mentioned in the GradSoft document, other components and features discussed in this document can be interpreted as possible enhancements to GradSoft (GrADSoft – A Program-level approach to using the Grid, http://www.cs.rice.edu/~mmzn/grads/Current_Version.html). This document also tries to make less opaque the Big Opaque Box (BOB) mentioned in the GradSoft document. The document realizes the above goals by analyzing the different steps taken in the ScaLAPACK prototype demo and suggesting improvements for each of the steps.

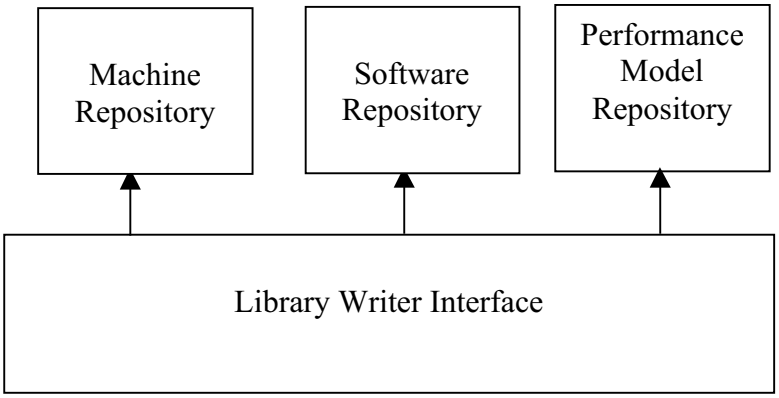
Following are the various components that will be needed for the enabling of numerical libraries over the grid system.

- *Library Writer Interface* – Used by the library writer to integrate his library into the grid. This component is invoked before the submission of the user's problem involving the library.
- *Application Manager* – The central caretaker of the user's problem, coordinating the marshalling of the user's parameters and invoking the different components of the grid system for the successful completion of the user's request. Part of the grid and initiated during the grid setup.
- *Phase 1 AART Constructor* – Constructs a preliminary Application Abstract Resource and Topology (AART) model that encapsulates simple constraints for machine capabilities to solve the problem, like the software availability, topology of the application and so on. Invoked by the Application Manager after receiving the user's problem.
- *Resource Selector 1* – Takes the Phase 1 AART and returns back the list of machines that satisfies the constraints specified by Phase 1 AART.
- *Phase 2 AART Constructor* – Constructs the Phase 2 AART that encapsulates complex set of constraints for machine capabilities to solve the problem. These constraints take into account, functions for memory and other needs for the problem specified by the library writer in the performance model. The Phase 2 AART is used for pruning down the list of machines returned by Resource Selector 1. The Application Manager invokes the Phase 2 AART Constructor after getting resources from Resource Selector 1.
- *Resource Selector 2* - Takes the Phase 2 AART and returns back the list of machines that satisfies the constraints specified by Phase 2 AART.
- *Schedule Constructor* – Constructs the final scheduler or resource selector 3 using the performance model for the problem and the minimizer or scheduling algorithm suitable for the problem. Invoked by the Application Manager after Resource Selection 2 Phase.

- *Scheduler / Resource Selector 3* – Used for returning the final list of machines. Invoked by the Application Manager after the schedule construction phase.
- *Contract Negotiator* – The central scheduling system for the grid used for accepting or rejecting the application contracts. This component is part of the grid and is initiated during the grid setup. Invoked by the Application Manager to receive an approval for the problem execution on a specific set of machines.
- *Application Launcher* – Invoked by the Application Manager to start executing the user's application on the machines returned by Resource Selector 3.
- *Contract Monitor* – Invoked by the Application Launcher along with the application to monitor the application's progress.
- *Violation Reason Finder* – Used for analyzing the performance of the user's problem or contract violations. Invoked by the Application Manager when violations are detected for the user's problem.
- *Application-Specific Migrator* – Migrates the application from the machines where it is currently executing on to a new set of machines. Invoked during problem execution by the Application Manager to reschedule the problem to new resources.
- *Machine Repository* – Contains information about the list of machines and architecture of the machines.
- *Software Repository* – Contains a list of software installed on the grid and machines containing the software.
- *Performance Model Repository* – Contains performance models for different numerical libraries.
- *Schedule Algorithm Repository* – Contains different scheduling algorithms.

The full functionalities of these components will be made clear as we go through the steps of the current ScaLAPACK-GrADS prototype.

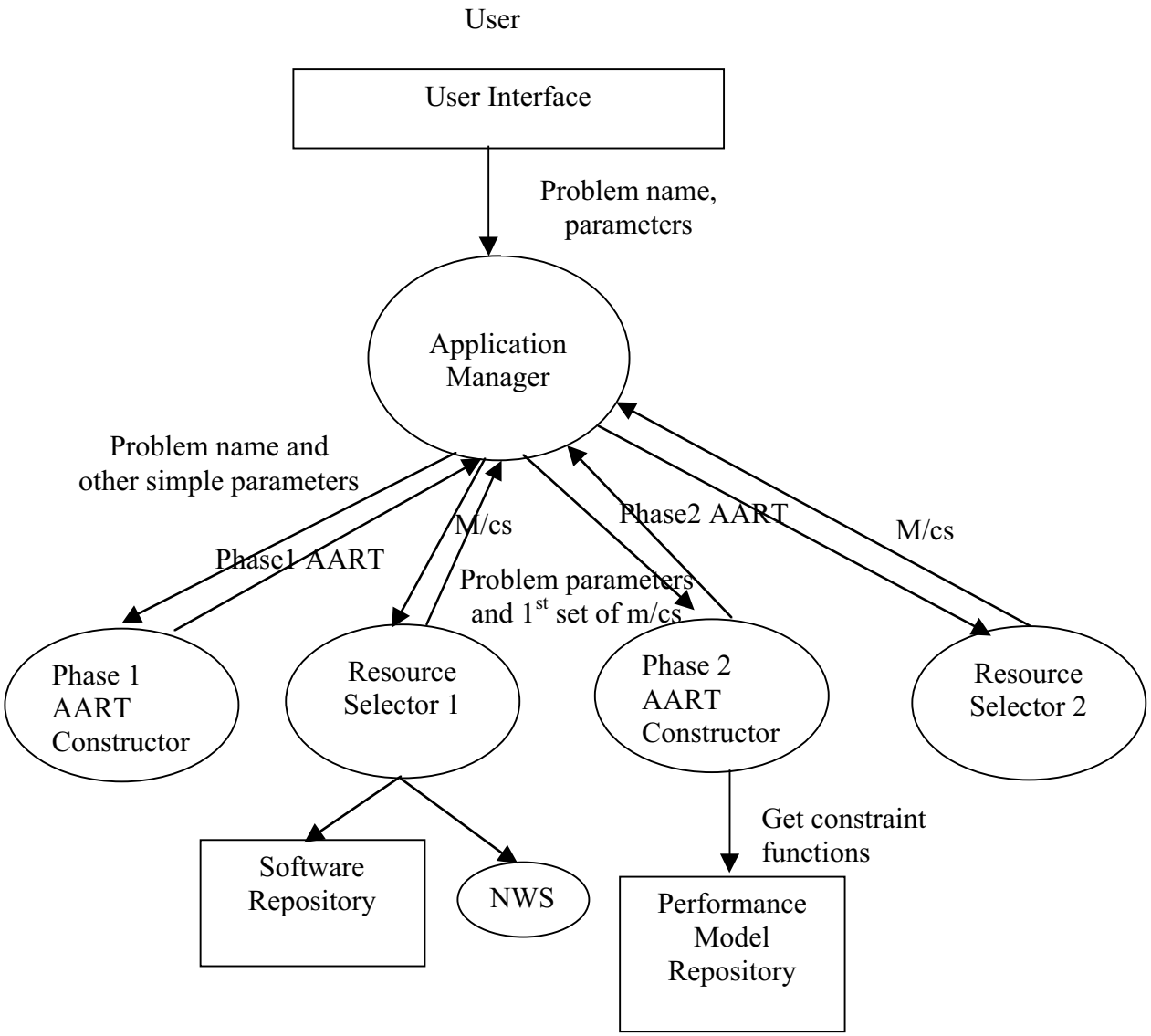
Library Interface Preparation



Library Writer

The library writer integrates a library routine into the grid system through an interface called the Library Writer Interface. This phase has to be completed even before the user submits a problem to be solved on the grid. The interface is a shell around which various grid components are contacted to help in setting up the system. This interface contacts the Machine Repository to find information about processors available in the grid and the computer architecture of these processors. The interface enters into a dialogue with the library writer and helps in ftping the executable into a specific location in the grid machines of specified architecture. The list of grid machines where the software is installed and the path of the executables are stored in the software repository. The Library Writer Interface also helps in ftping the performance model for the library routine into the Performance Model Repository.

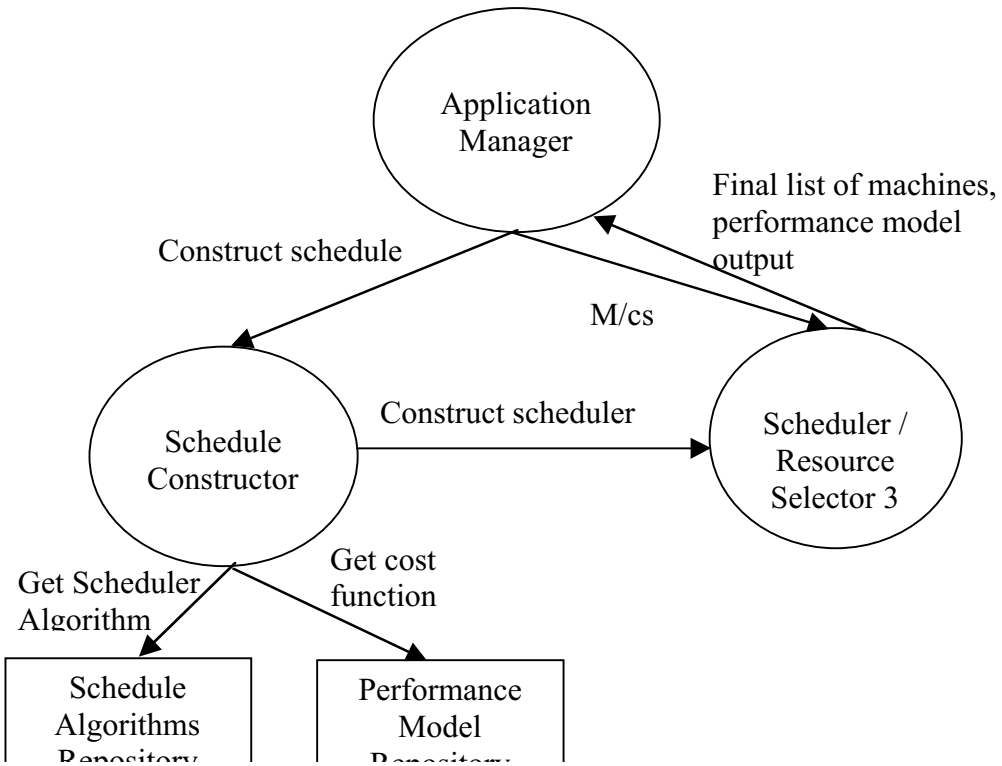
AART Construction And Initial List Of Machines



During a call to the library routine by the user, the Application Manager accepts user parameters through the user interface. In the Phase 1 AART Constructor, an AART with a simple set of constraints is developed. In the ScaLAPACK case, the Phase 1 AART constructor accepts the problem name, and creates an AART that specifies the constraint, "Machines with ScaLAPACK installed". For other numerical libraries, there can be other simple constraints like mesh topology and so on. The Application Manager then passes the Phase 1 AART to Resource Selector 1. The Resource Selector 1 accepts as input, the constraints specified by phase1 AART and returns back a list of machines that have the library software installed, for our example a list of machines that have ScaLAPACK installed. The Resource Selector 1 also fills up the NWS information for the machines it returns. The NWS information retrieved deals with potential processor speed and memory limits, as well as bandwidth and latency between each par of processors.

Phase 2 AART Constructor takes as input the list of machines returned by Resource Selector 1 and the user-passed parameters and returns back a subset of machines in the grid. The constructor uses certain functions from the performance model repository that define the resource needs for the problem. For example, in ScaLAPACK, there is a memory constraint function that specifies the amount of memory needed, given a list of machines and the problem size. Phase 2 AART Constructor retrieves this function from the Performance Model Repository and also uses the machines returned by Resource Selector 1 and constructs the Phase 2 AART that specifies the constraint "machines with ScaLAPACK installed and satisfying minimum amount of memory". Resource Selector 2 takes the Phase 2 AART and returns back a set of machines satisfying the constraints specified in phase2 AART

Scheduler And Final List Of Machines



The Application Manager then invokes the Schedule Constructor. The job of the Schedule Constructor is to construct the final scheduler or resource selector that will return the final list of machines that can solve the problem. To construct such a resource selector, the Schedule Constructor needs two components: the Schedule Algorithm that drives the Performance Model and the Performance Modeler itself. The prototypes of Schedule Algorithm and the Performance Modeler are well defined. In the ScaLAPACK and PETSc prototypes, the Performance Modeler has the same prototype, namely

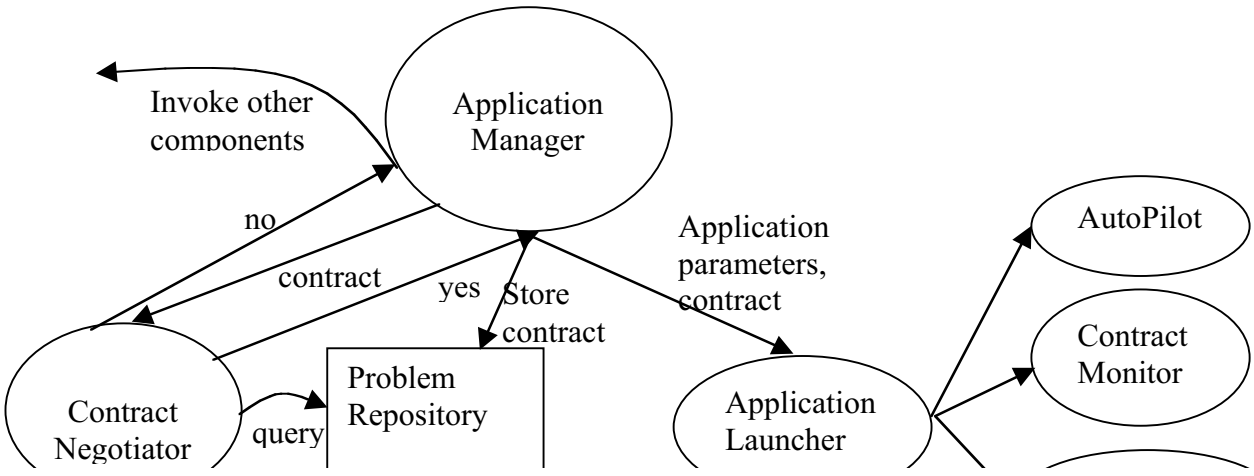
Execution_time (IN machines, OUT time, OUT model_output);

model_output variable is used by the Performance Modeler to pass values back to the Application Manager. These values can be used later when launching the application. For e.g., in PETSc, the model_output contains row distribution information, that depends on a given set of processors. This information is used later when launching the PETSc application. Irrespective of the numerical libraries, the function prototype of execution_time will be the same, only the definitions will differ. In ScaLAPACK, execution_time will return the total time for the LU factorization and solution, in PETSc this is the total time for the sparse iterative solve and in Holly's iterative mesh based application, it is the time per iteration.

Similarly, the Scheduling Algorithm can adhere to a single format. Currently, we have 2 scheduling algorithms: the adhoc algorithm that UTK uses based on doing a simulation of the routine running on a subset of processors and determining a "best time to solution" for the subset and the linear programming based algorithm that Holly uses. Though the functionality of these algorithms differ, they both follow a similar pattern, i.e., they accept a list of machines as input, use their own scheduling code and make calls to execution_time. GrADS can do some preliminary experiments with different scheduling algorithms for different problems and store information regarding best scheduling algorithm for problems in the Scheduling Algorithm Repository.

The Schedule Constructor uses this information in the Schedule Algorithm Repository, gets the best schedule algorithm for the problem, and together with the cost function from the Performance Model constructs the final scheduler / Resource Scheduler 3. The Application Manager passes the processors that it received from Resource Selector 2 to Resource Selector 3 and gets back the final list of processors where the problem will be executed.

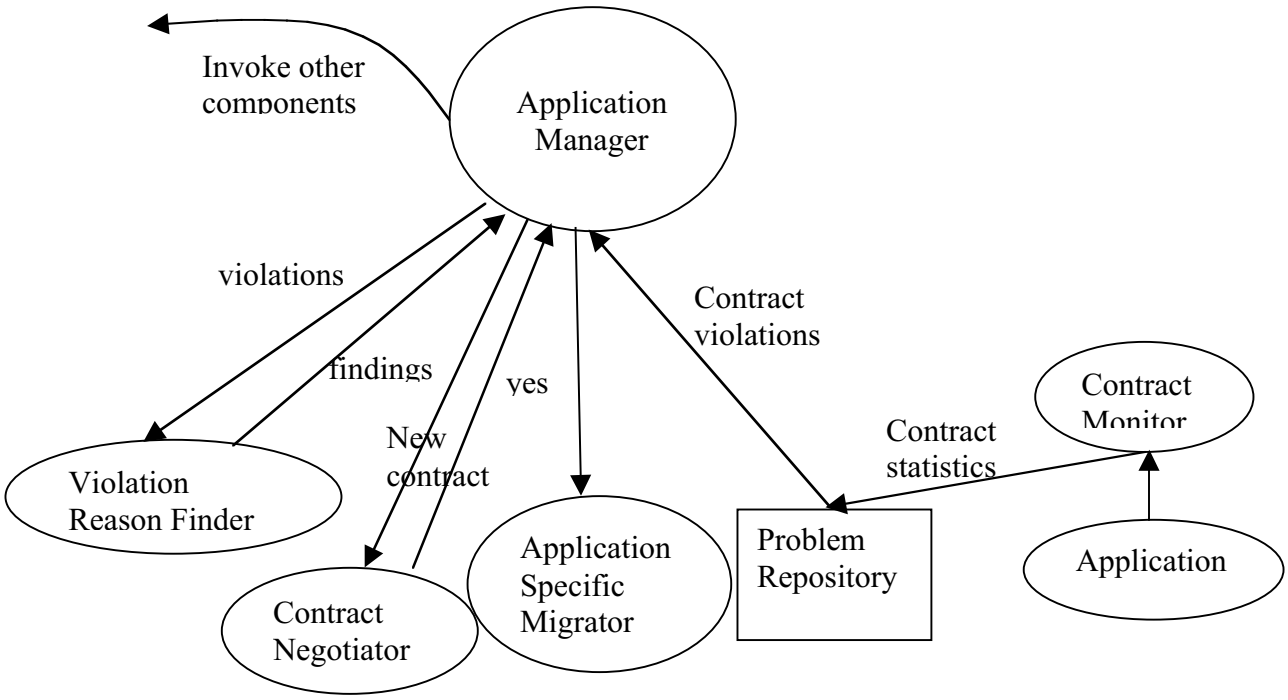
Contract Specification, Validation And Application Launching



The list of machines, the machine parameters, the problem parameters and the expected execution time for the problem on the list of machines form the contract. The Application Manager, after receiving this information from the previous steps, creates the contract and passes the contract to the Contract Negotiator for validation. The Contract Negotiator can either approve or reject the contract depending on other applications running on the grid. The Contract Negotiator acts as the "grid police" overseeing all of the applications running on the grid and deciding if incoming contracts can be approved without seriously impacting the performance of other applications currently running on the grid. It uses the Problem Repository to make its decisions. Sathish's thesis deals with this component.

If the contract is rejected, the Application Manager will invoke the previous steps to generate a new contract. If the contract is approved, the Application Manager stores the contract, and starts the trio of AutoPilot, Contract Monitor, and Application Launcher. We can also think about implementing an AutoPilot service that manages a pool of AutoPilot managers and creates a new AutoPilot manager only if necessary.

Application Execution, Monitoring And Rescheduling



The Contract Monitor collects contract statistics of the application and stores them in the Problem Repository. The Application Manager, either through monitoring or through notifications, detects violations, if there are any. If there is a violation of a contract, the Application Manager invokes the Violation Reason Finder with the contract violations. The job of the Violation Reason Finder is to analyze the violations, determine the reasons for the violations and also suggest to the Application Manager if the application has to be

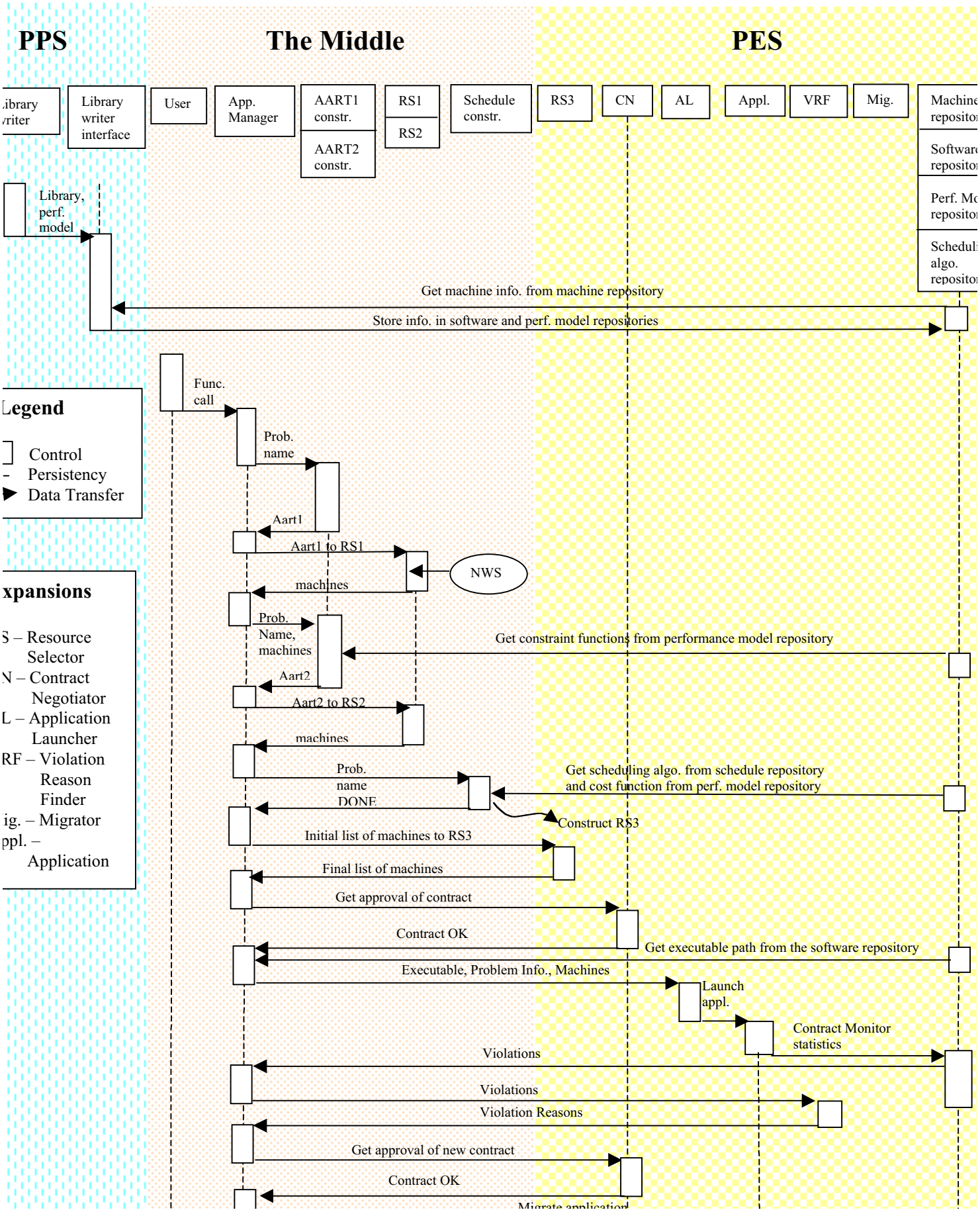
migrated. We guess that this is part of Otto's thesis and also the work being carried out in UIUC.

If the Violation Reason Finder suggests that the application be migrated, the Application Manager invokes some of the previous steps, generates a new contract and supplies the contract to the Contract Negotiator. The Contract Negotiator determines if performance benefits can be obtained by migrating the application to the new resources. If it determines that performance can be improved, it approves the contract and the Application Manager invokes the Application-Specific Migrator to migrate the application to the new resources.

The above steps will be repeated till the application completes and the Contract Monitor notifies the Application Manager that the application has completed. These steps can be summarized by the following diagrams.

Application Manager (the Application Manager will make calls to each routine in columns below in turn.)									
Phase 1 AART constructor *Input:* problem name *Output:* AART1									
	Resource Selector 1 *Input*: AART1 *Output*: machines								
	Software and machine repository, NWS	**Phase 2 AART constructor** *Input:* machines, prob. parameters *Output:* AART2							
		Performance Model Repository	**Resource Selector 2** *Input*: AART2 *Output*: machines						
			Machine repository	**Schedule Constructor** *Input:* Prob. name *Output:* scheduler					
				Schedule Algorithm, Performance model	**Scheduler/ Resource Selector 3** *Input:* Initial machines *Output:* Final machines				

						Contract Negotiator *Input*: contract *Output*: OK / not OK			
						Problem Repository	**Application Launcher** *Input*: Machine and prob. Parameters *Action*: Autopilot, contract monitor, application		
								Violation Reason Finder *Input*: Contract violations *Output:* Reasons for violations, suggestions	
									Application Specific Migrator *Input:* New machines *Action:* Migrate


PPS
The Middle
PES
ibrary
/riter
Library writer interface
User
App. Manager
AART1 constr.
AART2 constr.
RS1
RS2
Schedule constr.
RS3
CN
AL
Appl.
VRF
Mig.
Machine repositor
Softwar repositor
Perf. Mo repositor
Schedul algo. repositor
Library, perf. model
Get machine info. from machine repository
Store info. in software and perf. model repositories
Legend
Control
Persistency
Data Transfer
xpansions
S – Resource Selector
N – Contract Negotiator
L – Application Launcher
RF – Violation Reason Finder
ig. – Migrator
ppl. – Application
Func. call
Prob. name
Aart1
Aart1 to RS1
NWS
machines
Prob. Name, machines
Get constraint functions from performance model repository
Aart2
Aart2 to RS2
machines
Prob. name
DONE
Get scheduling algo. from schedule repository and cost function from perf. model repository
Construct RS3
Initial list of machines to RS3
Final list of machines
Get approval of contract
Contract OK
Get executable path from the software repository
Executable, Problem Info., Machines
Launch appl.
Contract Monitor statistics
Violations
Violations
Violation Reasons
Get approval of new contract
Contract OK
Migrate application