Proceedings of the UK e-Science All Hands Meeting 2004, © EPSRC Sept 2004, ISBN 1-904425-21-6

Workflow Enactment in ICENI

Stephen McGough, Laurie Young, Ali Afzal, Steven Newhouse and John Darlington

London e-Science Centre, Imperial College London, South Kensington Campus, London SW7 2AZ, UK
Email: lesc-staff@doc.ic.ac.uk

Abstract. Workflow specification and enactment is a critical operation in e-science. In this paper we describe how an abstract workflow, specified by an end-user in the form of an Execution Plan, is instantiated within the ICENI environment through the enactment pipeline. The pipeline starts with the workflow specification, includes the mapping of work onto resources through a workflow enabled scheduler which is able to make use of performance results captured from previous executions within the ICENI environment, and ends with the orchestration of the concrete execution plan on the specified collection of distributed resources. We show that ICENI is capable of deploying the specified components of the workflow over the available resources.

1 Introduction

The ICENI (Imperial College e-Science Network Infrastructure) uses an XML based language to describe workflows that are submitted for execution (see [4, 7]). The workflow describes a collection of components and the links between them (see Figure 1). These workflows are called Execution Plans (EP). When EPs are submitted into the ICENI environment they are abstract in nature, we are using the workflow definitions here from the NeSC e-Science Workflow Services Workshop [2].

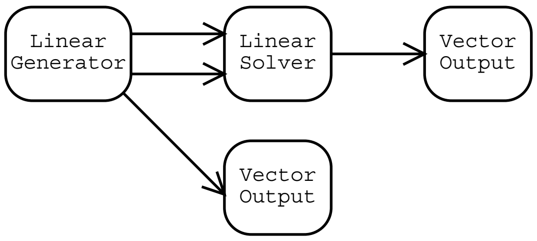

Fig. 1: A simple ICENI Workflow

ICENI describes components within the EP in terms of meaning, behavior and implementation. The meaning of a component is the high level description of the work the component performs and the dataflow of the component. The behavior of a component is the notion of control flow through the component, while the implementation of a component describes the algorithm used to implement the component, the language used and the data-types and the ports (methods) of the component. Each component meaning may have multiple behaviors and each behavior may have multiple implementations. For further details on the ICENI EP and components system see [7].

Unless specifically required the components (or jobs) that make up the submitted workflow are described in terms of their meaning as opposed to their implementation and/or the resources to deploy onto. Thus the workflow at this stage is spatial in its nature and oriented around data flows. For example if a component was required to solve a set of linear equations the description in the EP would merely ask for a linear solver. The user may wish to specify a particular implementation (such as LU factorization), in which case this can be specified within the EP. The user may also wish to specify a particular resource and/or code implementation specified in a similar manner.

The Enacment pipeline can be seen as a pipeline of stages which are performed in order to move from an abstract EP (workflow), through to a concrete (fully qualified components over a suitable subset of the available resources) workflow, and an instantiation of the concrete workflow onto the appropriate subset of Grid resources. By using the metadata provided by component implementations, performance repositories and resources, ICENI is capable of mapping these abstract EPs through to instantiated running applications. This is achieved by matching each component's meaning with an appropriate behavior and implementation description. The matching includes determining which implementation is best to use on the resources that will be available during the lifetime of the workflow, where 'best' is defined by some combination of user and resource owner metrics. Information about the components is combined with performance data in order to compute which components will be executing at the same time. This allows the scheduler to select a concrete workflow with appropriate set of resources that helps to minimise the inter-component interference.

We present here the architecture used to achieve this, consisting of schedulers, performance repositories and launching mechanisms, which can include resources capable of delivering advanced reservations.

In section 2 an overview of the ICENI enactment system is provided. Section 3 describes the scheduling processes to convert an abstract workflow into a concrete workflow. Section 4 details the process of taking the concrete workflow and deploying it across the available resources including a detailed description of the mechanism used to create advanced reservations in ICENI. This section also gives a breakdown of how the ICENI Grid Container facilitates the communication between components.

2 The Workflow Pipeline in ICENI

In this section we present an overview of the ICENI enacment system. This is presented in the order that the enactment pipline will be used in order to deploy a workflow. See figure 2.

2.1 Execution Plan (EP)

The Enactment pipeline starts with the generation of an abstract EP. Abstract workflows can be generated manually or through a graphical interface. ICENI makes use of the Netbeans Application Framework to provide a drag and drop workflow generation tool. This tool discovers, through a OGSI gateway, the components that are available within the ICENI environment and allows the user to construct workflows from these components. The abstract workflows can be checked for correctness at this stage and then submitted to an appropriate scheduler. Netbeans allows the user to inspect the concrete workflow at the end of the scheduling cycle and monitor the execution of the workflow at runtime. This EP is passed to the Scheduling Service which uses the services of the performance Repository, security authority and software repository to select the appropriate resources and implementations for the components.

Once the resources, where components will be deployed, have been determined the process of launching the components onto the resources is undertaken. The ICENI environment provides a number of Launchers each of which is capable of launching over different architectures. Components can be either launched directly onto these resources, or if they are still to be scheduled placed into a virtual space (or “Green Room”) awaiting scheduling at a later stage. The term “Green Room” is taken from the theatrical world; actors will wait in this room awaiting their time on stage. Some of the launchers allow for advanced reservations. The scheduler will make an advanced reservation on these resources and place the current component into the “Green Room” – providing a pre-deployment holding area.

Each resource within the EP will be sent, via its Launcher, a JDML (Job Description Markup Language) document detailing the work that needs to be performed. In the case of an ICENI job this will launch a Grid Container on the resource which will take the concrete EP and instantiate the components that are on its resources.

Deployed components within ICENI run within a Grid Container, which provides the mechanism for inter-component communication and event notification. For each resource used in the workflow by a user there will be one Grid Container which may handle multiple components. The Grid Container is also responsible for orcastrating the components.

3 Scheduling

Most scheduling algorithms do not attempt to search the entire problem space to find the optimal concrete workflow, as this is an NP-hard problem. Instead, heuristics are used to approximate the optimal solution. Thus the aim of most schedulers within ICENI is to map the abstract components to a combination of resources and implementations that is both efficient in terms of execution time of the workflow and in terms of the time to generate the concrete workflow. Components need not all be deployed at the same time: lazy scheduling of components and the use of advanced reservations help to make more optimal use of the available resources for both the current and other users.

The scheduling framework in ICENI is responsible for taking an abstract workflow and determining the appropriate implementations and resources to use. The framework allows for “pluggable” scheduling algorithms which is used to determine these mappings. A number of scheduling algorithms have been implemented including game theory schedulers and simulated annealing schedulers. These schedulers have been programmed to be workflow aware. Thus the scheduling of components depends not only on the performance of a component on a given resource, but also on the affect this will have on the other components in the workflow. Described below are the general steps taken to evaluate a suitable mapping of components onto resources.

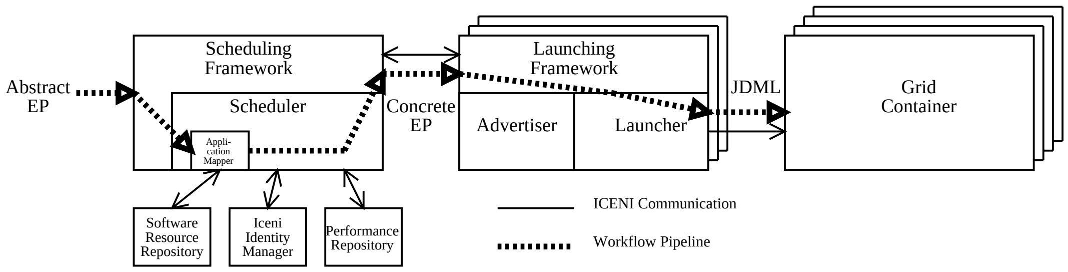

Fig. 2: The ICENI Workflow Pipeline

As the components that make up the abstract EP only describe the meaning of what should be carried out (we define this to include the dataflow between components) the first task of the Scheduling Service is to match these component meanings with component implementations. There may be many component implementations matching any given component meaning. The Software Repository provides the details of all components matching a given meaning. Once the implementations are known then selection can be performed as to which implementation should be used and on which resource.

To select an appropriate implementation on an appropriate resource the Scheduling Framework feeds the EP into the “pluggable” Scheduler, which may be workflow aware. The scheduler can then speculativly match implementations with resources, using the “pluggable” application mapper. The Scheduler can then interrogate the performance repository in order to obtain estimates on the execution times for these implementation / resource combinations. With this information and information gathered from the resources that have been advertised into the ICENI system, via their Launchers, the scheduler can determine an appropriate mapping of the components over the resources. Schedulers may support the notion of lazy evaluation, in which case only those components currently required are scheduled. Other components in the workflow are left unscheduled until required.

A number of equally optimal concrete workflows are selected using the attached heuristic scheduling algorithms. Performance information is then utilised to predict both the duration of the entire application and the times at which each component is expected to begin execution. The predicted component start times for each concrete workflow are then passed to the ICENI reservation service, which responds with a single concrete workflow, including any reservations it was able to make.

Below the stages involved in scheduling are described in more detail, this is also illustrated in figure 3. It should be noted that not all these steps need to be performed completely (or at all) in order to generate a mapping. If all the steps outlined are completed then the result will be an exhaustive search of the available component/resource space. This form of search is often very inefficient due to the size of the problem space. The main aim of the scheduling algorithms used here is to determine how to traverse the component/resource space in order to obtain a “reasonably” optimal mapping without the expense of searching the whole space.

– **Enumerating the possible Concrete Workflows** The first stage of scheduling the workflow is to find appropriate implementations of the components. This is performed by the application mapper. The mapper searches through all known implementations that match each components meaning. This may lead to a large number of concrete workflows which match the abstract workflow. At this stage each concrete workflow is checked for consistency and whether the user is allowed to use it. Consistency checks are made to ensure that the behavior between components match, ie the control flow between components is valid. Further consistency checks are made to ensure that the data types passed between components match. Each component within ICENI has it’s own access policy. This is used to prune the component implementations further by removing those the user does not have access. The ICENI Identity Manager is used to determine those components a user is allowed access to, this is determined by the components access policy.
– **Pruning the Resource Space** Once implementations of components are known the possible resources where the component may be executed can be determined. The metadata stored about each component implementation contains information about the resource requirements for running, along with any pre-required software that needs to be available on the resource. Each resource within ICENI will also have it’s own access policy. This can be used, with the

Identity Manager, to further prune the resource space.

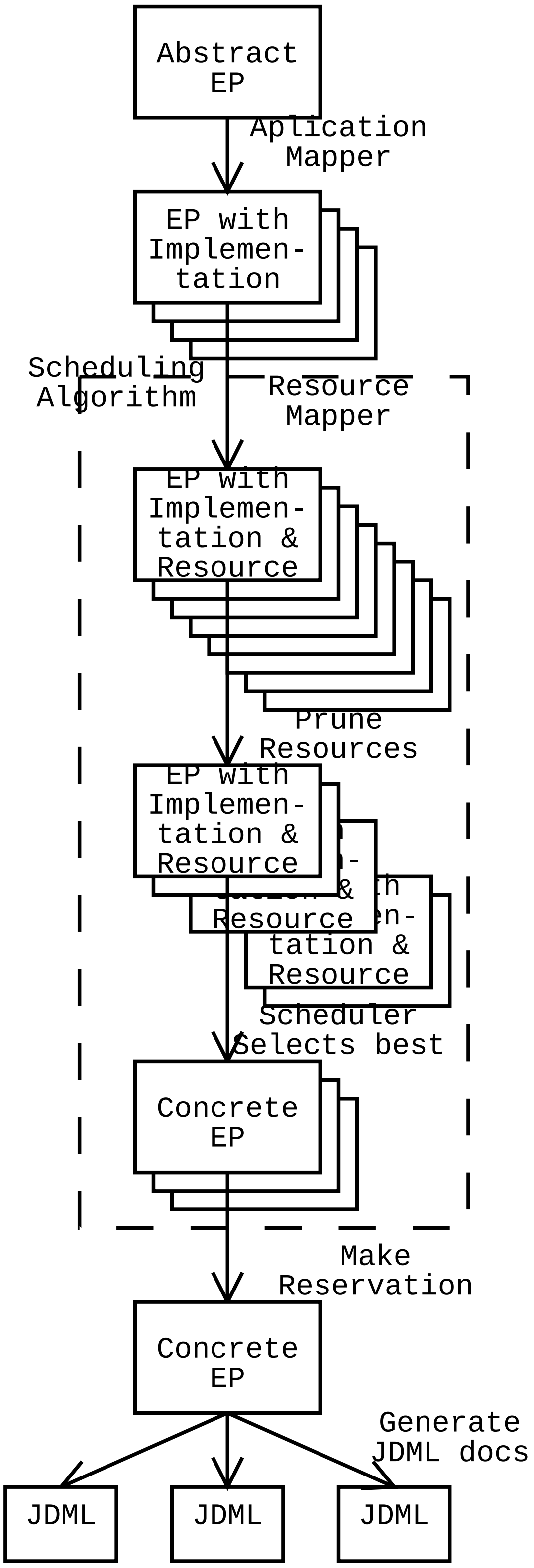

Fig. 3: The stages involved in scheduling

At this stage a set of concrete workflows can be generated by matching possible components with valid resources.

– **Selection of the “best” concrete workflow** The “best” workflow is defined by some user-defined criteria of the factors that are important to them. This could be based around quickest execution time, “cheapest” execution (where resources are priced) or some other metric, or combination of metrics. The techniques for combining these metrics and accurately modeling the users, resource owners and grid managers requirements is an area of current research [9].

  In order to evaluate the metrics for a given workflow it is almost always necessary to determine the execution time of each component in the workflow. This can be achieved by the use of the performance repository. The repository gathers information about the execution times of components on different resources. This information can be interigated during the scheduling phase in order to get a prediction of the execution time of a component.

  A number of scheduling algorithms have been developed for use in ICENI, these include random, best of n random, simulated aneiling and game theory schedulers [9]. These schedulers can be made “workflow aware” so that they take the whole worflow into account when scheduling each component.

Lazy Scheduling / Deployment In many workflows it may be beneficial not to map all the components to resources at the outset. This may be due to the fact that it is not possible to determine the execution time of a given component until further information is obtained by running previous components in the application. It may also be desirable to delay mapping until a later time in order to take advantage of resources and/or component implementations that may become available during the lifetime of the workflow. Certain Launchers are capable of handling advanced reservations (see below), in which case the scheduler will allocate a resource for the component to run on but the deployment of the component will not happen until the reservation is active.

The metadata held about a component implementation indicates whether a component can benefit from lazy scheduling and or advanced reservations. The scheduler may then then decide to ignore these components for the purpose of mapping. When the rest of the components are deployed to resources all components that are not currently mapped to a resource (or to a future reservation on a resource) are instantiated in a virtual space (referred to as the "Green Room"). Components in the "Green Room" are able to communicate with other instantiated components. The only valid operations that may be called on components held in the "Green Room" are those that add configuration data to the component, any call that requires computation will cause the component to be scheduled (see below).

Scheduling of components contained in the "Green Room" can be triggered by one of two events. If an already running component tries to communicate with a component in the "Green Room" with more than a configuration call then the component will trigger a scheduling operation. Alternatively the scheduler, which is aware of the time when a component should be required can preemptivly start scheduling so that the component is made real (just) before it is required. Components which hold advanced reservations will remain in the "Green Room" until the start of their reservation slot. At this time the Launcher will be given the components to deploy there.

4 Reservations and Co-allocation

The Reservations Service attempts to co-allocate all the resources required for the execution of the workflow. It does so by entering into negotiation with the appropriate Launcher for each resource in order to make advanced reservations for the time and duration specified in the concrete workflow. The negotiation protocol is based on WS-Agreement [8].

Advanced reservations can only be created if the launcher/resource supports them, otherwise the scheduler has to resort to best-effort service with regards to the particular resources that could not be reserved. If the negotiation fails, i.e. the resource cannot create a reservation satisfying all of the application's constraints, the reservations service can consider alternative schedules.

4.1 Instantiating Jobs from the workflow

The process of instantiating components onto resources is handled by the ICENI Launching Frameworks. There can be multiple Launching Frameworks within an ICENI environment with each framework representing one or more resources. Each launching framework is capable of deploying jobs through one launching technique using the attached "pluggable" Launchers.

The Scheduler Framework collects all components that are to be deployed onto the same resource and generates a Job Description Markup Language (JDML) document to represent the deployment of these components. These JDML documents are sent to the appropriate Launchers and describe how to instantiate the workflow on that resource through the use of the ICENI Grid Container.

There can be multiple Launching Frameworks within an ICENI environment with each framework representing one or more resources. Each launching framework is capable of deploying jobs through one launching technique using the attached "pluggable" Launchers. On receiving a JDML document, the Launching Framework will pass this document down to the attached Launcher which translates the JDML into a format the local resource or DRM can handle. The Launcher is also responsible for staging any required files to and from the resource. The Launcher will also monitor the running job and pass back any information about the success or failure of the job.

4.2 JDML

JDML is a general job description language designed for deploying jobs onto DRMs. Further information can be found in [1]. The work performed in developing the JDML is now feeding into the GGF [3] standardization process through the Job Submission Description Language working group [5].

Components in ICENI applications are deployed within the ICENI Grid Container. Thus the JDML documents describe how to start up the Grid Container and obtain the EP for the application.

4.3 Launchers

On receiving a JDML document the Launching Framework will pass this document down to the attached Launcher which translates the JDML into a format the local resource or DRM can handle. The Launcher is also responsible for staging any required files to and from the resource. The Launcher will also monitor the running job and pass back any information about the success or failure of the job.

4.4 Advanced Reservation Launchers

Some DRM systems support advanced reservation which can be exposed through the ICENI Launcher. In this case when the scheduler determines that a resource will be required for a certain interval of time a reservation is made with the Launcher through a WS-Agreement procedure based on the current WS-Agreement document [8]. If an agreement to reserve the resource is reached then the components to be deployed to that resource are placed into the "Green Room" until the resource reservation time is met. The scheduler is then responsible for submitting the JDML to the Launcher during the interval of the reservation.

4.5 The Grid Container

The grid container is the last stage of the workflow pipeline. The concrete components that are to be deployed onto the resource where the grid container has been initialized are now instantiated. The grid container is then responsible for the choreography of the overall workflow. It deals with passing data between the components within the applications and the firing of event notifications that can be used by the performance repository for collecting performance results in order to provide better predictions for future workflow predictions.

In most situations the components within a workflow will not all be deployed onto the same resource. As such there will be grid container running on each resource used in the workflow. These grid containers need to discover each other in order to allow the components to communicate. This is achieved through the underlying ICENI communication layer.

In order to monitor the execution of components ICENI uses an event model. ICENI events are fired whenever a component is entered or exited, these events are picked up by a running ICENI performance repository which can use the information in order to collect performance information to be added to the repository to improve future predictions. Events are also used to monitor the completion (or failure) of components.

5 Conclusion

We have shown in this paper how the components, within a workflow, are deployed within the ICENI environment. The deployment of these components can be performed in an efficient manner. although we seek to improve this process through further research, both in the choice of optimization policy and the use of scheduling algorithm to obtain the "best" schedule in a shorter period of time. Or at least a reasonable schedule in a reasonable time.

ICENI contains a complete workflow enactment process, capable of taking a users abstract workflow design and deploying this across a grid. Further techniques are being considered to allow users to define their problem space in a more abstract and general manner.

In the current implementation, users need to have their own launchers running in order to make reservations for their exclusive use. This is because the launcher makes reservations on the underlying Sun Grid Engine for the user running the launcher. This means that whoever has access to the launcher will be able to submit jobs to the launcher and have them run.

Since expecting all users requiring reservations to start up their own launchers, and thus have a local SGE account on the resource that they wish to run their jobs on, is not a scalable solution, it has been proposed that another layer of reservations administration be added, this time within the launcher itself.

In this architecture, the launcher keeps track of what reservations have been made, and are active, and hence whether a user's job should be submitted to the base level reservation or not. Hence, although the base level reservation allows any user to submit jobs via the launcher, the additional layer of administration in the launcher will police the submission to jobs.

An active area of research within LeSC is the performance overheads incurred by the ICENI system. Although these values are typically small, less than 10% of the overall execution time [6], we are striving to reduce this value. We are experimenting with techniques of caching and multi-threading schedulers.

References

1. A common job description markup language written in xml. `http://www.lesc.doc.ic.ac.uk/projects/jdml.pdf`.
2. e-Science Workflow Services Workshop. `http://www.nesc.ac.uk/action/esi/contribution.cfm?Title=303`, December 2003.
3. Global Grid Forum. `http://www.ggf.org`.
4. N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington. ICENI: An Open Grid Service Architecture Implemented with Jini. In *SuperComputing 2002*, Baltimore, USA, November 2002.
5. Job Submission Description Language Working Group. `https://forge.gridforum.org/projects/jsdl-wg`.
6. M. Y. Gulamali, A. S. MCGough, R. J. Marsh, N. R. Edwards, P. J. Valdes, S. J. Cox, S. J. Newhouse, and J. Darlington. Performance guided scheduling in GENIE through ICENI. In *To apper at the UK All Hands Meeting 2004*, Nottingham, September 2004.
7. A. Mayer, S. McGough, M. Gulamali, L. Young, J. Stanton, S. Newhouse, and J. Darlington. Meaning and Behaviour in Grid Oriented Components. In *3rd International Workshop on Grid Computing, Grid 2002*, volume 2536 of *Lecture Notes in Computer Science*, Baltimore, USA, November 2002.

8. Grid Resource Allocation Agreement Protocol. `https://forge.gridforum.org/projects/graap-wg`.

9. Laurie Young. Scheduling componentised applications on a computational grid. MPhil Transfer Report, 2004.