Proc UK e-Science All Hands Meeting 2003, © EPSRC Sept 2003, ISBN 1-904425-11-9

ICENI Dataflow and Workflow: Composition and Scheduling in Space and Time

Anthony Mayer Steve McGough Nathalie Furmento William Lee
Steven Newhouse John Darlington

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

Abstract

With the prevalence of component based and service oriented architectures used to support e-Science activities, we examine different views of application composition supported within these systems, which tend to be spatial composition in the former case, and temporal composition (workflow) in the latter. We consider the advantages of each view; spatial composition enables, dynamic programming, while temporal composition provides information useful for performance analysis. ICENI uses a spatial composition view to allow maximum flexibility, but provides an inferred temporal composition to support scheduling optimisation. We describe the graph based language used to annotate component behaviours, and discuss optimisations derived from estimating execution time - with or without loops - and resource sharing. Two examples drawn from e-Science pilot projects are used to illustrate these techniques.

1 Introduction

1.1 Component and Service Oriented Architectures

In order to deliver e-Science, applications require transparent configuration and deployment within a grid environment. This requirement is driving the increasing interest in component based programming models and approaches to application assembly. e-Science applications typically require multiple software resources even where the core application is a monolithic code - visualisation, collaboration, database access, file transfer support and numerous other auxiliary features are often called upon, hence the seamless deployment of a composite application is at the heart of the e-Science enterprise. Existing component based systems designed for Grid and high performance support include the Common Component Architecture [1] effort in the United States, and Triana [9] and ICENI [5] within the UK.

A parallel but in many cases separate development path in the Grid community has been the adoption of the service oriented mechanisms pioneered and developed in the B2B sphere of E-Commerce. Mechanisms such as Jini, Jxta and Web Services enable the process of publishing and discovering resources, whether software, hardware, or information, on a very large distributed network, typically the internet, and to cope with issues such as variable performance and unreliability. The Global Grid Forum has ratified the first standards document, the Grid Service Specification [10], for a Grid based service architecture (OGSA) based on Web Services.

ICENI (Imperial College e-Science Networked Infrastructure) features an implementation independent service oriented architecture, providing a core API which can be mapped either onto Jini or a prototype Jxta implementation. Additionally all services can be presented as GT3-based Grid Services by using a gateway service [3]. On top of this core API is a component based runtime framework, which uses open extensible XML metadata descriptions of components to support the deployment process, together with higher level services, such as client-side tools, domain and identity managers, and scheduling and launching services.

1.2 Relationship between Services and Components

As ICENI provides component based programming model together with a service oriented architecture, it is well placed to serve as a case study for the intersection of component and service based systems, in particular the representation of the composite application. The issue of representing how a desired application or activity is to be composed from its constituent parts, whether they be components or services, is of importance, particularly since service based models tend to favour a *temporal* view of composition, and component systems a *spatial* one. We shall examine how both views are expressed in ICENI and other systems, and the benefits from using both orthogonal models to understand the composition.

2 Views

The composition we describe is not of the runtime structure, the architectural plumbing that supports the interaction between components or services, which in any case is very specific to each system, whether it be SOAP messages or sockets, or some other technology. Nor in terms of the abstract description of a component or service, which in either case is almost always based on a interface / ports model, but rather in terms of the context in which the units are being assembled.

There are at least two orthogonal views of an application composition,

Spatial Composition With this view of composition, all the units that make up the application are represented simultaneously, with detail representing how they relate and interact with each other. There is no ordering between the units. It is implicit that all the units exist concurrently, and while they may exist on the same machine, perhaps on the same processor, they require independent and distinct resource commitment (memory space, some proportion of the processor's time etc), which is not shared. Connections in a spatial composition represent movement of data (whether through calls or streams).

Temporal Composition Under this viewpoint, all units are ordered with respect to their temporal dependence. Concurrency, where it exists, is explicit. A temporal composition may include instances that at no time exist simultaneously, and thus may share resources. Connections in a temporal composition represent sequencing of activities; it is a workflow view.

These two aspects of description exist for any composition, but a particular *view* of a composition illustrates only the single model.

3 Expressing Spatial Composition

Spatial composition is analogous to declarative programming in that no procedural aspects are explicitly described. It is a form of expression that is extremely easy to represent using a visual programming paradigm, and within ICENI we utilise two different visual programming tools to support spatial application composition. The first is a dedicated java Swing based tool for application launching, the second is a plugin to Sun Microsystem's integrated Netbeans environment which provides additional facilities and views to the user.

In both cases a component is represented by a rectangle on screen, and an application a set of rectangles, connected between ports by pipes that represent potential interactions between the components. This model is familiar to scientists, as it resembles a wiring diagram for electronics, flow diagrams in hydraulics, and organisational and mechanical processes. It is a highly intuitive form - and inherently functional.

This is not to say that a spatial view of composition can only be represented by a visual programming model. The underlying document that represents a composition in ICENI, known as the *execution plan*, is derived from the visual representation. This script has three essential elements, "create new component service instance", "reference to existing component service" and "establish a new link between two component services". There is no ordering information in the execution plan. Of course the statement to create a new component instance is extremely detailed, and is subsequently augmented with annotations providing additional information by scheduler and

application mapper services (the latter choosing an implementation for the abstract component chosen). Nevertheless, the same spatial composition is represented in a textual form. While creating execution plans by hand in with a text editor is cumbersome and unsuitable, there is no reason why a spatial composition could be expressed with a clear and elegant scripting system, or even using an already known scripting system such as Perl or Python.

The spatial view of composition is used by other systems comparable to ICENI, such as Triana, and is the model used by the Common Component Architecture group.

3.1 Enhancements to Spatial Composition

This simple expression system can be augmented with the use of operators that can be applied to components or links. Within ICENI we make use of the *quantity* operator, which can be applied to a new component instance, to represent the creation of an arbitrary number of new instances. This is not represented within the visual environment, in order to reduce the complexity of the diagram presented to the user, and thus the cognitive load they experience. An example is the parameter sweep discussed below in Section 3.3.1. The application mapper expands the execution plan automatically to produce the required number of component instances.

Within ICENI components are connected by abstract communication links which exist between ports. A user-defined component may only have ports that accept one-to-one connections. Multiple connectivity is provided by special collective communication components, each with a particular behaviour. One-to-many collectives include broadcasts, splitters (in which large data set messages are divided and routed to the appropriate receiver), routers and filters. Many-to-one collectives include gatherers, buffers, and funnels (which simply act to route messages to a single receiver).

When a user defined component with a quantity operator is connected to another user defined component with a quantity operator, the connection is only deemed valid when the quantities are the same. In this way NxN connections are allowed, but NXM are not - the complexities of the mapping require user guidance and cannot be automated. If such an NxN mapping occurs, the ports are connected pairwise.

Where a user defined component with a quantity operator is attached to a single multiway port on a special collective component, the mapping is also allowed, and is performed in a one-to-many fashion. This is demonstrated in Figure 1 below.

3.2 Comparative Advantages of a Spatial Composition

The spatial composition view really comes into its own in terms of dynamic component assembly. An existing application can be viewed in terms of a spatial composition, and can then be modified easily by adding new components, deleting or adding new links etc. This is significantly more difficult in a temporal composition, as the act of adding to the composition occurs at some moment in time, and as such only modifications to future plans are possible. As the user examines the application, the content that is meaningfully modifiable changes, and worse, any changes could have different semantic impact if made at different times. Thus transparent, dynamic programming requires a spatial composition.

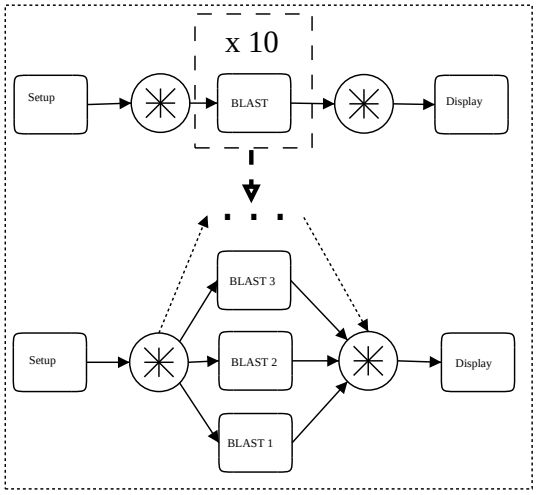
Within ICENI all deployed components appear individually as Grid Services, but additionally each application submission also appears as an additional Grid Service, representing the composition. This parent service can provide its own execution plan to a client tool, which enables the e-Scientist to modify the application at run-time. This has been of particular interest in the development of collaborative visualisation environments in connection with Access Grid, allowing multiple parties to connect to a complex composite application, adding their own visualisation and steering components during the process of the simulation. It is to support capabilities such as these, essential for deliver of e-Science, that ICENI uses a spatial composition model as its primary method of expressing application structure.

3.3 *e*-Science Applications Expressed with Spatial Composition

3.3.1 Example A: Parameter Sweep

Parameter studies are a common form of e-Science application: within the London e-Science Centre experiments are being performed to deploy highly distributed BLAST searches [1]. Figure 1 illustrates the spatial composition of such an application, with the use of two simple collective communicators. To avoid excessive drag-and-drop, the user connects a single BLAST representation between the splitter and the collector, and states the number of BLASTs required. The application mapper expands the spatial representation automatically.

Figure 1: Example A: Parameter Sweep's Spatial Composition



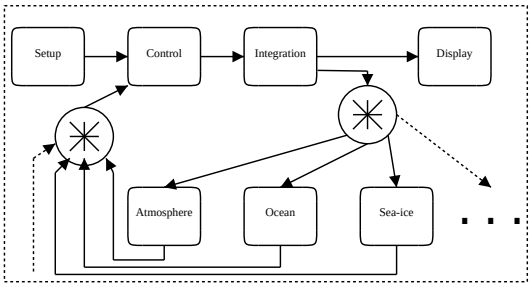
3.3.2 Example B: GENIE

Grid Enabled Integrated Earth system model (GENIE) [2] aims to simulate the long term evolution of the Earth's climate, by coupling together individual models of the climate system. The constituents may include models for the Earth's atmosphere, ocean, sea-ice, marine sediments, land surface, vegetation and soil, hydrology, ice sheets and the biogeochemical cycling within and between components. GENIE aims to be a modular and scalable simulation of the Earth's climate, allowing for individual models in the system to be easily added or replaced by alternatives.

Figure 2 illustrates the organisation of the application. The simulation components communicate with each other through an integration component. This component also performs tasks requiring data from all the simulation components (e.g. describing the heat exchange between the surface of the ocean and the base of the atmosphere), and is designed to be extendable to allow further simulation components to be added to the system in future.

A control component manages the flow of information between each of the components in the GENIE framework. It also allows communication of the simulation data with external resources such as visualisation and steering components.

Figure 2: Example B: GENIE's Spatial Composition



4 Expressing Temporal Composition

Temporal composition is typically expressed in a procedural fashion, in which the ordering of interactions is important. Temporal composition is essentially the *workflow* of the application.

There are numerous modelling mechanisms which support the description of changing state and interaction of application components. Petri net based models, clearly express the ordering and dependency of application activities, and allow a wealth of previous research and literature to be utilised in

[1] as part of the Proteome Grid, an e-Science Pilot Project funded by the Biotechnology and Biological Sciences Research Council

[2] an e-Science pilot project funded by the Natural Environment Research Council

exploiting the information. Conversely, state transition systems provide an alternative graph based model, with nodes representing complete system states, and arcs transitions [11]. Such graph based models allow both a textual and visual representation, in the manner of a flow chart, of the composition.

Application workflow is usually described either with a textual scripting language that is used to provide the "glue" that links units together, or a special bespoke workflow language, of which a great many exist and are discussed in the literature. For Web Services, there have been a number of efforts to produce a workflow description language, which in effect describes the temporal composition of web services. These include BPEL4WS (Business Process Execution Language For Web Services) [2], WSCI (Web Services Choreography Language) [6] and other earlier efforts. Though no standard has yet been agreed upon, experiments have begun with Grid Service Workflow Languages, such as GSFL [7]. All of these efforts are textual in representation.

4.1 Workflow within ICENI

ICENI's primary view is a spatial one, as described above. Nevertheless there is value in being able to describe the temporal composition of an application. Thus the temporal dependency is a *derived* view, in that the application must first be defined with a spatial view, and additional information, provided by the component developer (and stored as metadata describing the component's behaviour [8]) is used to provide a workflow view of the application.

Within ICENI we use a model similar to that of YAWL (Yet Another Workflow Language) [12], but simplified in certain respects. Each component has attached workflow information information, which consists of a graph in which the directed arcs represent temporal dependence i.e. a node's behaviour occurs after those which have an arc directed to it. Each node represents some behaviour, and the behaviour happens in an ordered fashion, beginning with the *Start* nodes, and finishing with the *Stop* nodes. The types nodes include:

Activity Represents computation and communication that occurs within the bounds of a component. An activity has duration, and possesses one child and one parent arc.

Send, Receive Communication through the component's interface is indicated with such nodes. Send has a single parent arc, receive a single child arc.

Start, Stop These nodes represent the creation and destruction of threads. Though they have no duration, they can occur at arbitrary points in a component's or application's lifecycle, as new threads spawn or are terminated. Start has a single child arc, stop a single parent.

AndSplit This node represents concurrency, in terms of spawning a new thread or threads. All the children of this node are activated simultaneously. AndSplit has a single parent arc, but possibly many child arcs.

OrSplit This node represents choice within the workflow. Each arc may be given a condition, or a probability, that they will be followed. The child arcs of an OrSplit are exclusive - only one can be followed. OrSplit has a single parent arc, but possibly many child arcs.

AndJoin The counterpart to AndSplit, this represents the end of concurrent execution. All parent arcs must be activated before the child arc is activated. In this was the AndJoin acts as a barrier synchronisation. Naturally it has possibly many parent arcs and a single child arc.

OrJoin Similar to AndJoin, the OrJoin requires only one of its parent arcs to transition before the child arc is activated. Thus it allows multiple branches of workflow to converge to a single set of actions.

Once the components and links of an application are determined by the spatial composition, the execution plan is parsed, and each component's workflow is connected to produce a complete application workflow. The application graph is assembled due to the association between *Send* and *Receive* nodes with the component's ports. As the ports of two components may be linked within the execution plan, an additional arc is added which makes the *Send* the parent node of the *Receive* on the corresponding component.

The application workflow is presented in graphical form upon submission of the application, and is allows the e-Scientist to examine the Activities of the application. By attaching each node to a line representing it's containing component, the effect is similar to that of a UML activity diagram - a model that is readily recognisable by end users.

4.2 Comparative Advantages of a Temporal Composition

A temporal, workflow view provides information unavailable to a spatial view, concerning, naturally, issues such as the concurrency structure of the application, execution times, and scheduling. Within ICENI, once the temporal view is derived from the spatial one, information can be extracted and passed to the scheduling service, a key part of the ICENI stack, which can utilise this in decision making regarding placement of new component instances.

4.2.1 Execution Time

The most information that can be gleaned from the workflow graph is the maximum expected execution time - found by searching for the critical path through the graph from any start node to all stop nodes. This can be determined recursively with a simple algorithm.

By determining the expected expected execution time on a number of different platforms (assuming that the Activity Nodes have annotations giving performance models for different platforms) composite models of application performance can be compared, and this comparison used by the scheduling service [4]. Note that the mapping of components to resources selected by the scheduler need not be the one with the quickest estimated execution time, as other information may also be brought into play, such as reservation, cost, user constraints and so forth, as required by the particular scheduling algorithm in question . The most important feature of this process is the that performance model is given for the application, rather than the components, and it may well be the case that due to synchronisation issues a particular component's performance is of no relevance to the entire application performance, implying that a cheaper (and more time consuming) resource would be optimal.

Issues arise with critical path analysis where the workflow graph is not acyclic. This is in fact quite frequent in scientific applications, occurring whenever two components are call each other, or there is a loop in the application between components. In many applications loops can be concealed behind the boundary of the component interface, (in which case they are considered part of the corresponding Activity Node), but in applications where components represent aspects of a larger loop, this is impossible. In this case the entire body of the application is in loop.

Loops are dealt with by elimination: cycle detection is a relatively straightforward graph algorithm. Once the cycles in a workflow have been identified, they themselves can be given a performance model in the same manner as the complete application. Their contribution to the overall application performance is a multiple of the estimated loop execution time.

There are then two possibilities: either the user provides an estimated number of loops (this can be stored as component meta-data, or provided at submission) to determine the complete model, or the loop is considered in isolation. This latter case occurs where the loop is certain to dominate the execution time of the application: for example, in the GENIE case the application requires only an initialisation phase before beginning the loop. Where the loop dominates, the loop performance model is taken in place of the application performance model for purposes of scheduling, ignoring the multiple for loop iterations - the scheduling decision for a single iteration will be the same for all (if non-loop activities are ignored).

4.2.2 Interleaving Resources & Deferred Deployment

In addition to providing a composite performance model, the temporal composition may may act as a guide to the sharing of resources between components that do not operate at that same time. Where one component's activities have ceased, the resources assigned to it may be redeployed to support another component, and if necessary, returned to the initial component. This is especially useful in cases where a small number of components perform most of the work in an application - in which case the other component's resources can be reassigned. The actual process of assignment is carried out by the scheduling algorithm, and

depends upon issues such as reservation possibilities, queue lengths and the like, so it is impossible to determine what is pragmatically possible from the workflow graph alone. However the workflow information does at least allow these optimisations to be made, by indicating at which phase of the application's execution a component requires resources.

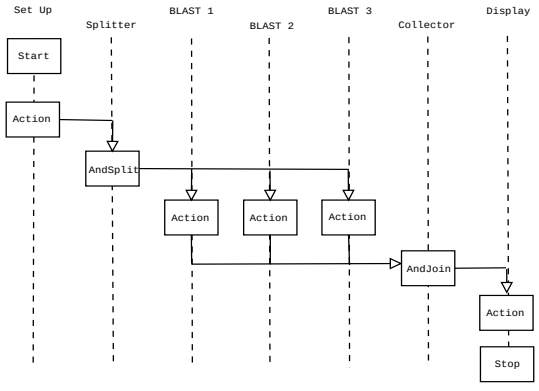
4.3 *e*-Science Applications in ICENI: Temporal Composition

Both of the examples given above for spatial composition have derived workflow models which represent the temporal composition of the application. In order to simplify the diagrams, we have omitted the send and receive nodes, which are present between all communications between components.

4.3.1 Example A: Parameter Sweep

The parameter sweep's workflow model is extremely simple, and acyclic. The critical path is the entire application. Opportunities for deferring the final display activity exist - and the initial set-up component's resources could be reassigned following the completion of its activities.

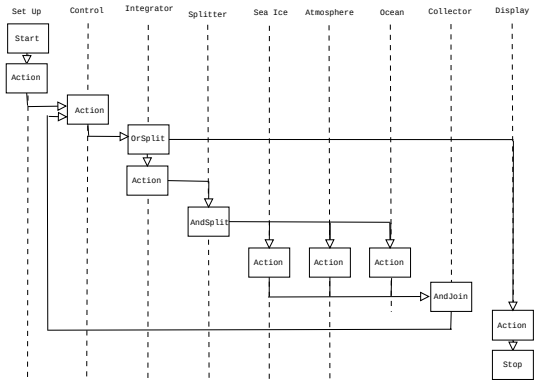
Figure 3: Example A: Parameter Sweep's Temporal Composition



4.3.2 Example B: GENIE

The GENIE application workflow is very similar to that of the parameter sweep, but features a main loop. In the case of GENIE, the user indicates, once the workflow is analysed and presented, that the loop is the main body of the program. The scheduling system then considers deployment choices using the performance model for a single loop as a guide. Opportunities for deferred deployment are restricted to those components that do not feature in the main loop, which leaves only the display component at the end. Similarly the set-up component's resources may be redeployed following startup.

Figure 4: Example B: GENIE's Temporal Composition



5 Implementation Status & Further Work

At this present time the workflow language is fully supported within the ICENI component meta-data system, and complete workflow view is presented upon application submission, along with a service view (indicating the location of the components-as-services within their federated domains) and the spatial view, used for dynamic application construction.

The workflow graph provides methods which offer information on critical paths, and opportunities for optimisation, and our current set of scheduling services utilise this information.

Future implementation work will enable the workflow view to become more interactive, both by instrumenting the services so that the flow of activities is visible to the user in real time, but also so that the application builder can add their domain specific knowledge with regards to loop completion and importance to facilitate optimisation.

A further possibility will be to allow the user to describe their application entirely from

a temporal viewpoint, using either a graph based tool, or an existing workflow language. The units of composition would be composites of activities and other nodes, and appear independently of their software component, which could then be inferred. This is the converse of the existing system, and would allow the a spatial composition to be determined by a temporal one.

6 Conclusions

In this paper we have shown that

- Spatial Composition, the typical model for component based systems, often graphical in nature, is useful for dynamic assembly of applications
- Temporal Composition, the typical workflow description for service oriented architectures, can provide useful information for application scheduling.
- ICENI uses as spatial composition view, based upon component meta-data as its primary model, and infers the temporal composition from the user defined spatial composition, by using a graph based workflow language
- This behavioural information may be used to support optimisations in scheduling, by providing composite performance models and indicating opportunities for resource sharing.

References

[1] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a Common Component Architecture for High-Performance Scientific Computing. In *The Eighth IEEE International Symposium on High Performance Distributed Computing, HPDC'99*, August 1999.

[2] BPEL4WS Consortium. Business Process Execution Language for Web Services, May 2003.

[3] N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington. ICENI: An Open Grid Service Architecture Implemented with Jini. In *SuperComputing 2002, Baltimore*, Baltimore, USA, November 2002.

[4] N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, and J. Darlington. Optimisation of Component-based Applications within a Grid Environment. In *SuperComputing 2001*, Denver, USA, November 2001.

[5] N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, and J. Darlington. ICENI: Optimisation of Component Applications within a Grid Environment. *Journal of Parallel Computing*, 28(12):1753–1772, 2002.

[6] Intalia, Sun Microsystems, and BEA Systems san SAP. Web Services Choreography Interface (WSCI) 1.0 Specification, 2002.

[7] S. Krishnan, P. Wagstrom, and G. Laszewski. GSFL: A workflow framework for grid services, July 2002.

[8] 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.

[9] Ian Taylor, Matt Shields, Ian Wang, and Roger Philp. Distributed p2p computing within triana: A galaxy visualization test case. In *IPDPS 2003*, 2003.

[10] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman. Grid service specification. Open Grid Service Infrastructure WG, Global Grid Forum, June 2003.

[11] S. Uchitel, R. Chatley, J. Kramer, and J. Magee. LTSA-MSC: Tool support for behaviour model elaboration using implied scenarios. In *Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, April 2003.

[12] W. van der Aalst and A. Hofstede. Yawl: Yet another workflow language, 2002.