

Workflow Expression: Comparison of Spatial and Temporal Approaches

Anthony Mayer Steve McGough Nathalie Furmento William Lee
Murtaza Gulamali 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

We identify two forms of workflow expression, temporal, in which relations may imply temporal dependencies, and spatial, in which they do not. Spatial expressions enable the dynamic modification of the workflow; by raising the level of abstraction they provide flexibility. This flexibility is at the cost of information; temporal expressions can guide performance analysis and scheduling decisions. The ICENI system uses a primarily spatial form of expression, but from this (and component meta-data) infers the temporal information. A case study of an environmental modelling system is presented as illustration of the two complimentary approaches.

1 Introduction: Workflow for Grid Programming

In recent months workflow has emerged as a key requirement for the development of sustainable, usable Grid systems. Many researchers, systems developers and application users are finding they need ways to express the structure of complex multi-unit jobs within a Grid context. Multiple definitions and usages of the term ‘workflow’ have been made, many synonymous, some mutually exclusive. For clarification we consider workflow to be

the organisation of a structured application in an abstract fashion, such that the implementation of the atomic tasks being organised is independent from the organisation itself.

The rationale behind this explicit demand for abstraction is the way Grid forces us to be implementation agnostic; in an uncertain and ever changing environment the precise details of the implementation may be unknown and unknowable at design time, and are left to the grid middleware to assign. This definition is also broad enough to encompass non-computational workflows such as business and factory process management (which is the concern of much of the workflow literature).

Workflows serve two distinct purposes. They allow *expression* of the user’s requirements, and through *enactment* they are used by some agent to compose and synthesise the execution of the underlying implementations. In this paper we are primarily concerned with questions of workflow expression.

2 Spatial and Temporal Expression

We make a distinction between two forms of workflow expression, those that are *spatial* and those that are *temporal*.

Spatial Expressin is defined as a workflow in which no relation holding between two or more units represents a temporal ordering, while

Temporal Expression is a workflow where any relation represents some temporal ordering.

This distinction is observed in practice: the presence or absence of temporal orderings produces very different forms of workflow expression, with different features, uses and qualities.

3 Features of Spatial Expression

Spatial composition is analogous to declarative programming in that no procedural aspects are explicitly described, and the strengths of such models are those of descriptive programming, primarily the independence from execution ordering. Spatial expressions typically feature *implicit concurrency*, and *dataflow interactions*.

It is implicit that all the units within the spatial expression 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. By dataflow interactions, we mean that the interactions between the units typically represent the potential to move data, either through a message call or a pipeline model of communication. As such the graph of interactions maps to a dataflow graph of the computation.

Spatial models of expression are typical to many component based systems, being a natural form of organisation composition between concurrently existing entities. Many systems derived from engineering or signal processing requirements and backgrounds feature a spatial view, such as Ptolemy [10] and Triana [7], as well as more general component based systems, such as the Common Component Architecture [1].

Our case study system, ICENI, is based upon a component model and the method by which composition is expressed is purely spatial with dataflow relations between ports. In terms of manipulation the expression, we currently offer two different visual programming tools, 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. The examples above indicate how the spatial form of expression lends itself to visual programming methods.

3.1 Advantage of Spatial Expression: Runtime Modification

The spatial composition view really comes into its own in terms of dynamic workflow modification. An existing application can be viewed in terms of a spatial composition, and can then be modified easily by adding new components, terminating component execution, and deleting or adding new links. This is significantly more difficult in a temporal composition, as the act of adding to the composition must occur at some moment in time during the composition. As such modifications can only be made to future plans, and the horizon of where meaningful changes can be made is continually advancing. Changes could also have semantic differences if made at different times. Thus we assert transparent, dynamic programming requires a spatial expression of the workflow.

Within ICENI an executing component implementation, deployed on a remote machine, exposes itself as a Grid Service. The composite application is also represented as a service, and presents the workflow of the running application to any authorised user, who can edit the composition (both creation and destruction of components and links) in real time as the application runs. Clearly this facility is only possible due to the enactment of the workflow by the ICENI middleware, but it is also enabled through the use of a non-temporal form of expression.

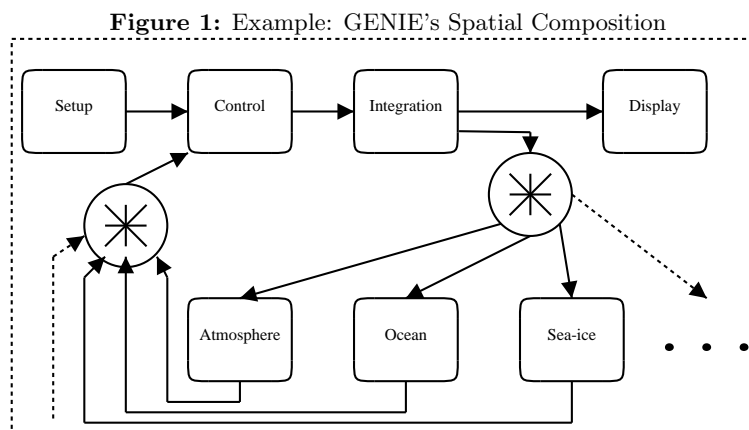
This technology has been of particular interest in the development of collaborative visualisation environments connected to the Access Grid, allowing multiple parties to join and interact with to a complex composite application, adding their own visualisation and steering components to a long running simulation. This model is thus highly applicable to the steering of long running 'Grand Challenge' problems, which need to be steered, visualised and modified without restarting the core simulation.

3.2 Case Study: GENIE Spatial Composition

Grid Enabled Integrated Earth system model (GENIE)¹ aims to simulate the long term evolution of the Earth’s climate, by coupling together individual models of the climate system. The constituents 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.

Figure 1 illustrates the organisation of the application within a spatial composition. 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.



4 Features of Temporal Expression

In contrast to spatial expressions of workflow, temporal expressions feature relations between units that imply a temporal ordering (i.e. unit A performs an action, send result to unit B, unit B performs an action). In contrast to spatial expressions, temporal expressions feature *explicit concurrency* and *control flow relations*.

Concurrency can be made explicit as the orderings of activities are explicit. In this way the potential for components to be executing simultaneously can, in cases where there is dependency, be ruled out. Thus it is possible for distinct units of a temporal expression to exist on the same resources (though not through the same interval in time).

The relations of a temporal expression typically indicate dependence or execution order, thus it is relatively straightforward to construct the flow graph of the execution - in this way the temporal expression can be considered to provide explicit control flow information, in contrast to the dataflow of the spatial ‘pipeline’ model.

Temporal expressions usually take the form of either a textual scripting language 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. High level scripting languages with a procedural style, such as Python and Perl are familiar to many end-users, and fall into this category, as are many domain specific languages.

¹an e-Science pilot project funded by the Natural Environment Research Council

For Web Services, the foundation of contemporary grid computing, 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 WS-BPEL (Business Process Execution Language For Web Services) [2], WSCI (Web Services Choreography Language) [4] and other earlier efforts. Though no standard has yet been agreed upon, experiments have begun with Grid Service Workflow Languages, such as GSFL [5]. All of these efforts are textual in representation.

Because it is extremely difficult to modify a workflow based on temporal expression while it is executing, this form of description is best suited to batch type jobs where there is no interaction with the model, or to successive parameter sweeps where the same workflow is executed many times. In the latter situation the workflow can be modified between execution runs.

4.1 Advantage of Temporal Expression: Execution Planning

The advantage temporal expressions give over spatial expressions is precisely what makes them unsuitable for runtime modification: they contain information about the execution ordering. This information can be extremely useful in determining the performance requirements for the composite application.

Temporal models are extremely tractable to analysis by formal methods. There are numerous formal 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 exploiting the information. State transition systems provide an alternative graph based model, with nodes representing complete system states, and arcs transitions [8]. Such graph based models allow both a textual and visual representation of the composition in the manner of a flow chart. Using such techniques it may be possible to determine the critical execution path of an application, and where there are opportunities for components to be deployed onto the same resources (as they will never execute concurrently).

4.2 ICENI: Inferring Control Flow

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 any temporal information must be *derived* from the spatial expression and from additional component specific data, provided by the component developer (and stored as meta-data describing the component's behaviour [6]).

The language used is similar to that of YAWL (Yet Another Workflow Language) [9], but simplified in certain respects. Each component has attached workflow 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 nodes themselves include *Activity* nodes, which represent the computation, and possess duration, *Send* and *Receive* nodes which indicate communication, and various *Split* and *Join* nodes which support concurrency and synchronisation.

Once the components and links of an application are provided by the user in a spatial expression, this workflow is parsed to component's behaviour is connected to produce a complete temporal expression of the application's workflow.

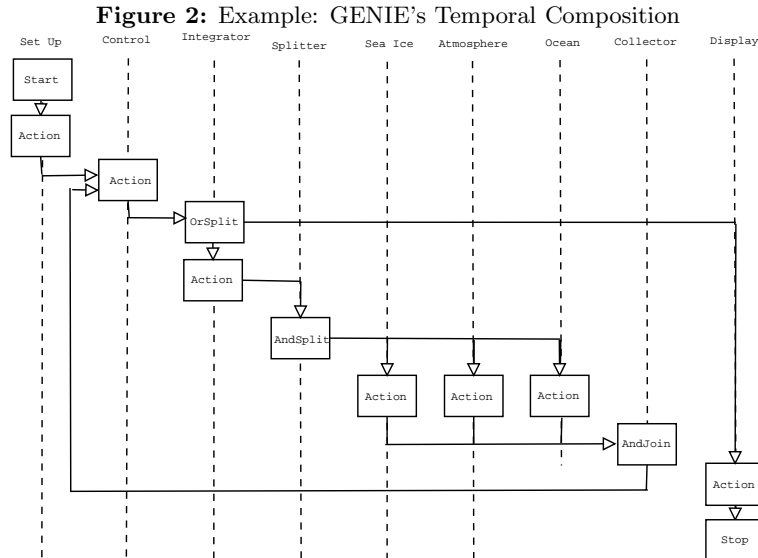
This temporal workflow is presented in graphical form upon submission of the application, and it allows the e-Scientist to examine the activities of the application. By attaching each node to a line representing its containing component, the effect is similar to that of a UML activity diagram - a model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

The critical path analysis is combined with performance meta-data for the activity nodes, to produce multiple performance models for the available choices of implementation platforms. These choices are compared by the ICENI scheduling service [3] to make decisions about deployment (which may be based upon considerations other than the fastest execution time, such as reservation, cost and user constraints).

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 a loop. Where this occurs we consider only the body of the loop for performance analysis purposes. Where the loop dominates the execution of the application, knowing the number of iterations (and thus the execution time) is unnecessary, what is required is a comparison between times with different resource choices. Assuming the number of iterations is data but not machine dependent, it is possible to compare the critical path analysis for a single iteration and make decisions using this.

4.3 Case Study: GENIE Inferred Temporal Composition

The GENIE application workflow features a dominating loop. Scheduling decisions are made according to the execution length of the loop rather than the application as a whole, which in this case is indeterminate. As such the initial setup phase and the final release of results are ignored for purposes of decision making, and only the core simulation components are considered.



5 Enactment

Workflow enactment is usually considered to be in one of two forms:

Orchestration A single workflow engine reads the workflow data, and dispatches method invocations, or execution directives to the underlying implementations to perform the required behaviour.

Choreography The units themselves enact the workflow by invoking methods on their partner units, these partnerships being specified by the workflow.

Both forms of enactment are compatible with both the spatial and temporal forms of expression; indeed it is possible to build both choreography and orchestration systems that support the same languages. ICENI supports a choreography model; the derived temporal view is not used to drive the execution of the components, only to capture information regarding what they are expected to

do, so as to inform the scheduling framework. The components themselves talk to their partners, the partnerships established by the ICENI Grid Container on deployment of the component.

6 Conclusion

In this paper we have shown that:

- Workflow expression is important in providing features that influence their execution.
- Spatial Expression, typical in component models, feature dataflow and implies concurrency, can allow dynamic modification of applications. This is due to the level of abstraction, by omitting temporal information the user's modifications can occur at any time.
- Temporal Expression, the typical in scripting languages, can provide useful information for application scheduling, by providing composite performance models and indicating opportunities for resource sharing. The information present that is lost in a spatial expression is valuable for supporting scheduling decisions.
- ICENI uses a primary spatial expression, and infers the temporal expression from a combination of the user defined composition and component meta-data, by using a graph based language.
- The enactment of the workflow is can be undertaken orthogonally to the temporal expression, which used only to inform scheduling decisions.

ICENI thus serves as an excellent case study in the synthesis of distinct but complementary forms of workflow expression.

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, 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.
- [4] Intalia, Sun Microsystems, and BEA Systems san SAP. Web Services Choreography Interface (WSCI) 1.0 Specification, 2002.
- [5] S. Krishnan, P. Wagstrom, and G. Laszewski. GSFL: A workflow framework for grid services, July 2002.
- [6] 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.
- [7] Ian Taylor, Matt Shields, Ian Wang, and Roger Philp. Distributed p2p computing within triana: A galaxy visualization test case. In *IPDPS 2003*, 2003.
- [8] S. Uchitel, R. Chatley, J. Kramer, and J. Magee. LTSA-MS: 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.

- [9] W. van der Aalst and A. Hofstede. Yawl: Yet another workflow language, 2002.
- [10] Yuhong Xiong and Edward A. Lee. An extensible type system for component-based design. In *6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Berlin, Germany, LNCS 1785*, 2000.