Formalising the Grid Environment

The 1st Step to Automate Grid Application Assembly using Deductive Synthesis

Alan Bundy Alan Smaill
Bin Yang
Centre for Intelligent Systems and their Applications
Division of Informatics
University of Edinburgh

August 15, 2003

Abstract

In the emerging e-Science, a Grid computing environment is coming into shape. However, the features of "rapid customised assembly of services" and "autonomic computing" have yet been adequately addressed in existing Grid prototypes [Atkinson *et al.*]. Our project is set up to apply deductive synthesis to automate Grid service assembly, using proof planning technology, provided that Grid services and applications can be specified in a suitable logic.

1 A Nontrivial Assembly Example

An econometrist is trying to recongnise hidden patterns in volatility time series of different stock markets using Neural Network, then apply a certain type of statistical tests on the computing results, and expect an informative plot on his own screen. What he needs to do in the visionary eScience Grid in the senario is to just parameterise and connect three Grid Application objects, *Database*, *Neural Network* and *Statistical Tester*, that are highly abstract, top-level implementations of real world tasks and accessible directly to Grid users.

One of the Grid Applications in the the scenario, the Neural Network, can be realised with two elementary Grid Services:

- an *Algorithm Library* which is a collection of platform-specific executable codes for functions and algorithms in specific domains, in this case, econometrics,

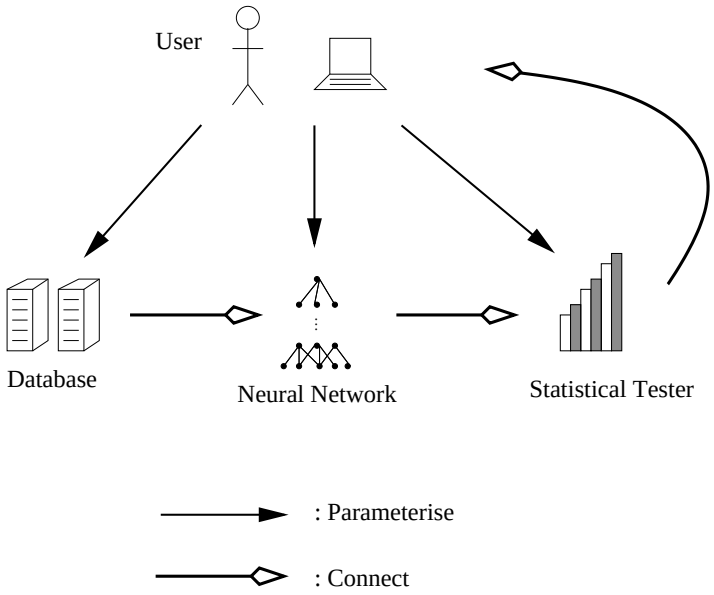

Figure 1: User customised Grid Applications

- a general *Computing Unit* that can be some supercomputer or a cluster of supercomputers that take platform-specific codes and execute them.

In a similar fashion, the Statistical Tester can be decomposed into a *Statistical Test Function Library* and a Computing Unit. To interpret and plot the test result, a *Visualisation Unit* is also needed, see Figure "Assembled Grid Services".

2 Formal Specifications of the Grid

2.1 1. Situation Calculus and Petri Nets

In the situation calculus, situations s are evolving to the next situation by taking an action a. A built-in function $do(a, s)$ represents the result of performing a in the situation s. Thus a function $OwnBook(bookName, s)$ means we own a book named "bookName" in s [Reiter 01]. The situation calculus offers an approach to formalise the Grid Applications and Services. For example, we can define the Computing Unit as:

$$computing(compResult, do(a, s)) \equiv$$
$$\{(a = InputCode(Executable) \wedge (CodeLib(Executable, s)))\}$$

which means that the general-purpose Computing Unit will produce *compResult* only if the action *InputCode* has been done to input *Executable* produced by a general *CodeLib* in the situation s. All other Grid Services in this scenario can be defined in this way.

The synthesis of Grid Services will be possible if there is an ontology that regulates the naming of all actions and thus enables composite Grid Services to realise its pre-

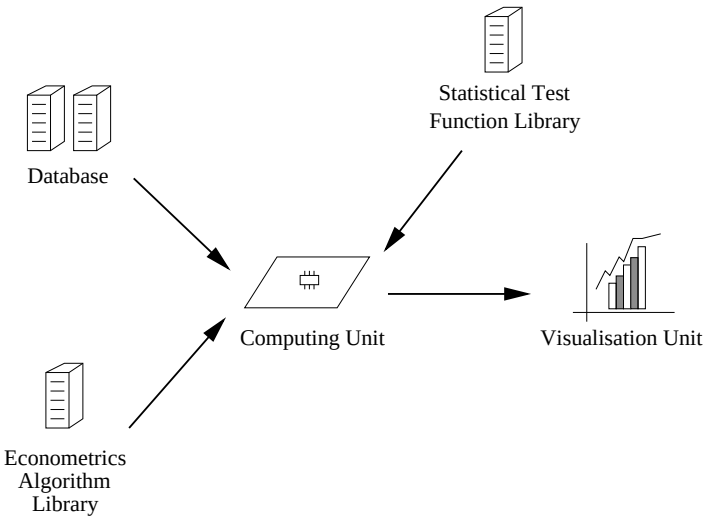

Figure 2: Assembled Grid Services

or post-actions. Grid Applications can be defined as composite functions with existing knowledge of their functionality, and then broken down into simple functions that are correspondent to Grid Services.

The situation calculus can then be mapped to Petri Nets and visualised using the simulation and modelling environment KarmaSIM [Narayanan 99]. Roughly speaking, a Petri Net is a bipartite graph containing *places* (drawn as circles) and *transitions* (drawn as rectangles). Places hold *tokens* that enable their executions and represent predicates about the external or internal state. Transitions are active components, and have *Input I* and *Output O* which both map to a or many places. During execution, token is removed from a place, passed to the transition through its Input and deposited, maybe multiplied, in new place(s) from the transition Output. With these settings, we could have a diagram, Figure 3,[1] of the composite Grid Services that would be generated automatically by KarmaSIM.

2.2 2. Event Logic

Event Logic is a mathematical structure based on constructive Type Theory designed to express the key features of a distributed computing system at the level of abstraction appropriate for specifying interactive behaviours [Bickford & Constable 03]. The logic has only seven axioms and can be modified and extended to capture those features in the Grid environment, such as the concept of Virtual Organisation, collaborative computation and so on. It will be much earlier and more straightforward to apply deductive synthesis technique in a classical constructive Typed logic formalisation.

[1]All Petri Nets functions of Grid services have no arguments of situation because execution order was assigned and indicated using the numbered arrows.

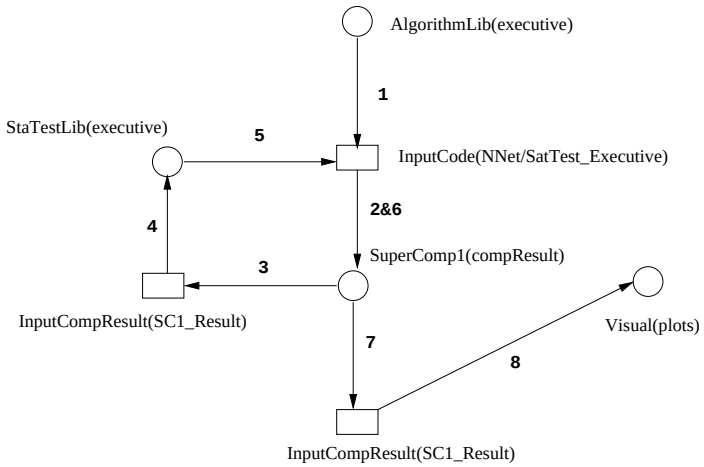

Figure 3: Structure of the synthesised Grid example

3 Work in Progress

We have been looking for an appropriate formal specification of Grid environment, using first the situation calculus and Petri Nets, and very recently the event logic that seems more prospective.

References

[Atkinson *et al.*] Malcolm Atkinson, Jon Crowcroft, Carole Goble, John Gurd, Tom Rodden, Nigel Shadebolt, Morris Sloman, Ian Sommerville, and Tony Storey. Computer chanllenges to emerge from escience. http://www.nesc.ac.uk/news/Vission.pdf. UK eScience Research Agenda.

[Bickford & Constable 03] Mark Bickford and Robert L. Constable. A logic of event. Tech Report TR2003-1893, Cornell University, 2003.

[Narayanan 99] Srinivas Narayanan. Reasoning about actions in narrative understanding. In *IJCAI*, 1999.

[Reiter 01] R. Reiter. *Knowledge in Action: Logical Foundation for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.