

Dependable Grid Services

Dependability Interdisciplinary Research Collaboration

www.dirc.org

Introduction

Services provided remotely over computer networks are subject to failure (tardy response, no answer, etc.) for diverse reasons, ranging from resource starvation and network instability through to implementation or specification error.

With the advent of web services and the grid, automatic composition of services is now possible, and may reasonably be expected to have a multiplicative effect on these failures since breakdown of a single operation in a composition could jeopardise the entire procedure.

A mechanism is needed to manage failures in this kind of environment.

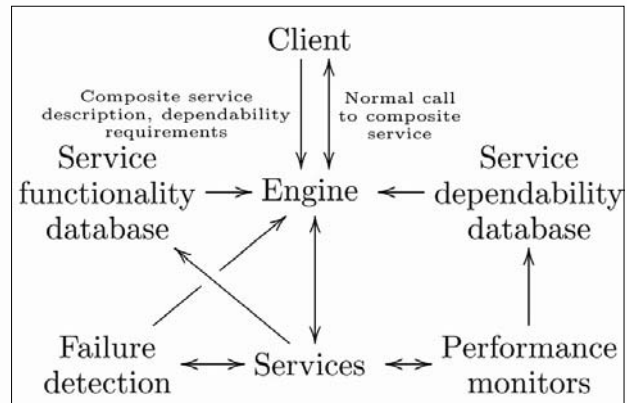
To that end, we are implementing a system which will deliver an end-to-end service to a required level of dependability if that service may be expressed as a composition of other multiply available services (or resources) of known dependabilities.

"Dependability" here is limited to qualities which may be improved through diversity, recovery and requirement matching: most usually availability, reliability (chance to succeed), time to complete, and possibly accuracy.

To provide a fault-management mechanism, we need to address a number of research problems. These include service dependability specification, composition description, service monitoring and failure detection.

Overview

Our solution is structured as follows:



Operation comprises two phases:

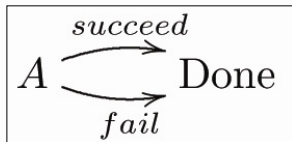
1) firstly, given a composite service description and dependability requirements, the engine constructs (by referring to the service functionality and service dependability databases) an "instantiated dependable composition" of specific identified services which, in conjunction with a set of rules for prevention of or reaction to failure, will satisfy the client's requirements;

2) secondly, it executes this augmented composition, detecting service failures as they occur and reacting appropriately in order to provide the required level of dependability.

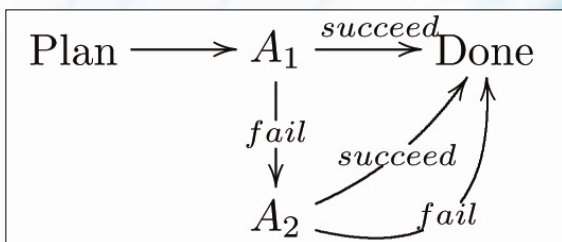
While at the moment these two steps are distinct, it may be possible to improve system effectiveness in future by merging them, thus allowing it to adapt to changing circumstances during composition execution.

Engine operation

Given the trivial service \composition" of a single service of type A as follows:

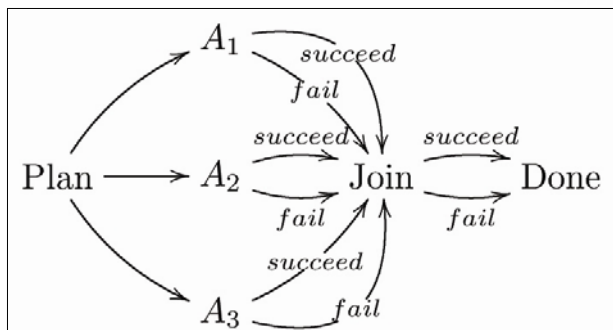


...where "Done" indicates communication of results | be they success, failure, or something more nuanced | to the client, the following might be a simple augmented instantiation:



...here, during the planning phase two specific services of type A (A1 and A2) are chosen with A1 as primary and A2 as backup. If A1 fails, A2 is called; if A2 also fails, then failure is admitted. Obviously this process could be continued forever, but the planning phase would establish in advance how many fallbacks were necessary to satisfy the client's requirements within cost and time constraints.

Another option follows:



...here A1, A2, and A3 are all run in parallel; "Join" could be chosen to mean

waiting for the first successful reply and then finishing (favouring availability and (to some extent) speed), or it could mean a vote, wherein at least two identical results were required before a successful outcome was recorded, favouring reliability instead. Different clients would get different solutions according to their requirements, and the engine will plan these solutions using estimation based on dependability data for the available services.

Testing

In order to help verify correctness of operation of the engine, we are also developing a framework for injecting failures into arbitrary networks of real grid services| thus we can script scenarios which test its ability to produce a defined level of service in context of networks with "known" metrics.

Flexible framework

While we intend to implement solutions for all elements of this framework, it is clear that many of them are potentially subject to wide variation: the two service databases may in fact be a single entity (MDS perhaps); service monitoring may be undertaken by third parties, or services may be trusted to advertise their own metadata honestly; all monitoring may in fact be supplanted by service level agreements; the finer nuances of failure detection may be service-specific so the framework must admit specialised detection mechanisms. Clear separation of design components will admit all of this flexibility. The greatest such issue arises in the core engine: what language describes the composition, and how is that then executed?

Several candidates (none open source) exist, and we hope to achieve an implementation which can be "ported" to a new composition engine with a minimum of effort.