Efficient Dynamic Resource Specifications

Kris Bubendorfer, Peter Komisarczuk and Kyle Chard
School of Mathematics, Statistics and Computer Science
Victoria University of Wellington
PO Box 600, Wellington 6001, New Zealand
{kris.bubendorfer, peter.komisarczuk}@mcs.vuw.ac.nz

ABSTRACT

In the effort to reach beyond 3G, researchers have been actively looking at utilizing new models for network based services. Small mobile, pervasive and ubiquitous devices will benefit from networked services and computation provided by utility computing providers and the virtual organizations that lease resources from them. As an additional factor, we believe that it is critical that the mobile, pervasive or ubiquitous devices be able to dynamically manipulate their resource specifications when obtaining services and resources from the utility computing and communication network. This requires a simple, manipulatable, and preferably modular resource specification structure. This paper presents the Resource Description Graph (RDG). The RDG is used to represent available and required resources for hosts and applications in a directed acyclic graph. The RDG has many desirable properties including inherent security, expressiveness, modularity, and composition. We show that the computational time to match RDG resource specifications, thirty resource types and constraints, is less than 1ms — demonstrating that the RDG is a practical approach to resource specification with a low computational overhead.

Keywords

Pervasive, mobile, ubiquitous devices, resource specification, utility computing

1. INTRODUCTION

The early innovators in the fields of pervasive [1] and ubiquitous [14] computing envisioned a world of small lightweight wearable computing devices, that perform many heavyweight functions, such as realtime language translation or crime scene fingerprint matching. The underlying computing power and information resources required to perform these tasks were to be provided by a computing utility, somewhat akin to a power or water utility, rather than by the device itself. The computing utility would enable the device to be flexible, by offering access to many services, whilst being lighter, with smaller power and storage requirements. However, the ideal of a computing utility has not yet arrived, firstly the devices themselves are now considerably more powerful themselves, and capable of immense functionality even when operating without network connectivity. Secondly, through data services such as GPRS, full internet connectivity can be provided separately to the provision of computation. A significant non technological hurdle is that of social acceptance.

Nonetheless, in the efforts to reach beyond 3G, researchers have been actively looking at utilizing new models for network based services. One such model is the internet Virtual Organization (iVO), where the iVO creates services on demand utilizing resources leased dynamically from Utility Computing Providers (UCP) [12]. The basic UCP model is further extended by including the leasing of communications services, resulting in Utility Computing Communications Providers (UC^{2}P).

One currently evolving model of utility computing is based on grid computing [7], and the GRASP [6] project is exploring the grid paradigm based around the Application Service Provider model. However the UC^{2}P infrastructure, as envisaged, demands the NGG - Next Generation Grid which needs to encompass more resources and provide greater flexibility in terms of mobility, resource allocation and economy based resource allocation than the current Grid implementations.

The predominant Grid toolkit is Globus [10], which has come some way to providing an NGG through the Open Grid Service Architecture (OGSA) and enabling web services though the Open Grid Service Infrastructure, Web Service Resource Framework (OGSI, WSRF). However its communication services model does not encompass the potential services that could be required in a general purpose UC^{2}P scenario, and its resource allocation does not provide for speedy resource allocation and agile pricing mechanisms, which are required in iVO operations.

Within a large scale distributed systems, such as UC^{2}P systems, efficient negotiation for and allocation of resources plays an important role in the performance of the system. Recently much emphasis has been placed on the computational advantages of large scale grid systems which provide mechanisms to share a wide pool of resources such as computers, software, and peripherals amongst many users and organizations [8]. Describing the resources offered by the distributed system and requested by the participating parties is vital in ensuring efficient allocation of the resources, providing quality of service (QoS) guarantees and therefore maximizing the overall performance of the system. However, an important aspect of a resource specification mechanism is finding an algorithm to quickly and efficiently match requested resources to distributed system components. For example, the XML encoded resource specifications that feature in the later Globus RSL standards are encumbered by the processing overheads incurred by parsing the XML. This is a minor inconvenience when considering the large relatively static scientific computations that were envisioned by the designers, where processing may occur over days, if not months. However, when we change the model to include iVOs that alter their configurations dynamically depending on client demands, and the overhead of communicating the specifications over wireless links to pervasive and ubiquitous devices with limited computation, power and connectivity — the issue of encoding of resource specifications becomes a significant bottleneck in UC^{2}P systems. WirelessXML minimises transmission overheads, but does not minimise processing overheads.

As an additional factor, we believe that it is critical that the mobile, pervasive or ubiquitous devices be able to dynamically manipulate their resource specifications when obtaining services from the UC^{2}P network. This requires a simple, manipulatable, and preferably modular resource specification structure. There is little emphasis on resource specification performance analysis in the Globus literature, especially with respect to generating and processing real-time resource specifications within distributed systems.

This paper presents an efficient resource specification mechanism called the Resource Description Graph (RDG). The RDG is a single rooted directed acyclic graph (DAG) that is passed between hosts in a simple textual format. The RDG is able to encode resource constraints and availability for a host thorough a sequence of edges. Similarly an application can provide an RDG to a host or group of hosts which represents a valid combination of resources required for the application to run. This specification consists of a sequence of edges terminating at a vertex marked as an *accept state*. There may be many resource combinations leading to a single accept state representing a set of compromises or alternatives from the view of the application requesting the resources. The RDG has several advantages; in particular expressiveness and versatility; for example, it can be decomposed into subgraphs as processes split or migrate between hosts and can also express mobile object aggregation. The performance results presented in this paper demonstrate that this RDG based system has rapid specification and processing capabilities that can provide fast allocation of resources and thus enable rapid negotiated mobile object migration.

The purpose of this paper is to demonstrate that a resource specification can be encoded and processed in a format that is compact, modular and most importantly can be constructed and matched – with the limited computational power available from small mobile, pervasive or ubiquitous devices.

1.1 Origins

The RDG was developed as the vehicle for communicating resource requirements within a mobile object middleware architecture called NOMAD (Negotiated Object Migration Access and Delivery) [2, 3]. Like the Grid, typified by the Globus initiative [8], NOMAD is a distributed computational system, which consists of a collection of loosely coupled cooperating virtual machines (depots) that are capable of hosting distributed applications. Nomad utilizes an economic resource management model, of which details can be found in [4].

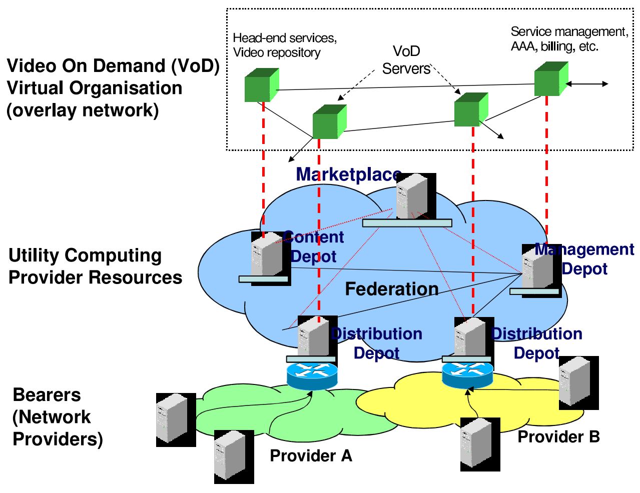
Video On Demand (VoD) Virtual Organisation (overlay network)
Head-end services, Video repository
VoD Servers
Service management, AAA, billing, etc.
Marketplace
Utility Computing Provider Resources
Content Depot
Management Depot
Federation
Distribution Depot
Distribution Depot
Bearers (Network Providers)
Provider A
Provider B

Figure 1: Internet Virtual Organization Supported in NOMAD

NOMAD offers support for iVO applications made with mobile objects that can migrate as required, Figure 1. A federation is a meta depot, permitting a resource provider to construct manageable resource clusters from a set of depots. Figure 1 illustrates an example from cellular networking where mobile devices and their users can obtain services from a local service gateway to optimize performance, see [12]. In such example applications the resource specification needs to provide versatile and extensible specification of required resources and any alternatives; this is accomplished through the RDG.

1.2 Related Work

The majority of recent related resource specification work is in the area of grid systems. Network resource specification work is encompassed within the grid grid resource specification. Two resource specification schemes are mentioned here specifically: Condor classAd language and the Globus Resource Specification Language (RSL). Condor is a high throughput computing environment where idle resources on a network are discovered and allocated to applications. It was conceived as a cycle stealing scheduler which pays particular emphasis to the computer owners rights. Condor uses the classified advertisement language (classAd) to describe jobs, workstations, and other resources [13]. The classAd language is a symmetric, semi structured declarative language designed to allow easy matching of resources and requests in order to correctly execute jobs on the Grid. A classAd is a mapping from attribute names to expressions and two classAds match if each has attribute requirements that evaluate as true. As with the RDG, customer agents use the classAd to request resources and a machine (a resource) can use a classAd to advertise its resources. Matching is performed by the Condor central manager to determine the compatibility of jobs and workstations. The classAd language is available as a stand alone package for other applications and currently it has a native and an XML form [13].

Currently Globus (the de facto standard for Grid computing) specifies an XML based language called the Resource Specification Language (RSL) to describe resource requirements [11]. The RSL was changed from the Globus Toolkit 2 to an XML based language in Globus Toolkit 3 to ensure optimal portability. The RSL is used by Brokers, the Globus Resource Allocation Manager(GRAM) and the Globus Architecture for Reservation and Allocation (GARA) [9], which is a mechanism designed to enable end-to-end QoS guarantees supporting resource discovery and dynamic (a reservation for some future time) or immediate reservation. GARA extended the generic Globus resource management architecture by introducing a generic resource object, which encompasses network flows, memory blocks, disk blocks, and other entities. XML based resource specifications are encumbered by the processing overheads incurred by parsing the XML resource specifications, and are probably not suitable for lightweight, mobile software and devices.

All three resource specification systems, classAd, Globus RSLv3 and the Nomad RDG are all sufficiently descriptive and flexible to be used by systems other than those for which they were developed.

2. RESOURCE DESCRIPTION GRAPH

We propose the Resource Description Graph (RDG) as an efficient means of representing combinations of resource requirements for an application and resource availability at a host. The RDG enables resource allocation across a set of loosely coupled computational and communication resources.

From the viewpoint of an application, an RDG describes a group of tasks (encoded as accept states), each with a set of resource combinations that represent resources required to satisfy each task. The edges of the RDG encode the resources and the vertices the state. There may be alternative resource allocations for each task — these represent viable alternative sets of resources that can be utilized to perform the required application function. A path from the root to an accept state is called a sentence, and an accept state may therefore be traversed via multiple sentences.

Anytime that a set of resources needs to allocated over a set of providers we face an *NP*-complete optimization problem [2]. By constraining the combinations of resources within the RDG we minimize the *NP*-complete Combinatorial Allocation Problem (CAP)[1] within a system. In an economic resource allocation system, such as NOMAD, each sentence is valued, and the preferred (lowest cost sentence) for each accept state is chosen.

The resources available at a host are likewise encoded in an RDG, which we have termed a *Resource Profile*. A resource profile does not utilise accept states — but is instead used to record the current availability of resources and any future resource commitments at a host. The point here is, that certain resources are only available in set combinations — but with variation in their weightings, e.g. both disk space and disk IO are required to store data, the exact weighting between the two resources depends on the application. The RDG is a good fit for describing the resource constraints of both applications and hosts - a handy situation, as resource allocation is essentially an exercise in constraint satisfaction.

[1]To allocate items in order to maximize total utility

2.1 RDG Structure

Figure 2 shows part of a simple content distribution virtual organisation RDG. The iVO RDG specifies a requirement for a number of servers with Constant CPU rate (CCU), memory and disk to perform, say, video stream serving.

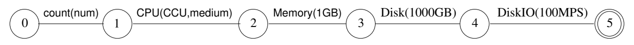

Figure 2: Example iVO RDG, double circles indicate an accept state

It is possible that the iVO could also utilise a slightly different set of resources to provide its services. Figure 3 shows an RDG that encodes two different sets of resources that can be used interchangeably by the iVO. A resource provider, may satisfy either the resource sentence (a) {0,1,2,4,5,6} or (b) {0,1,3,4,5,6}. The alternative sentence (b), utilizes a Variable CPU rate (VCU), and so requires additional memory for caching. The principle here, is that an accept state relates to a service, and not the resource sentences by which that service can be provided.

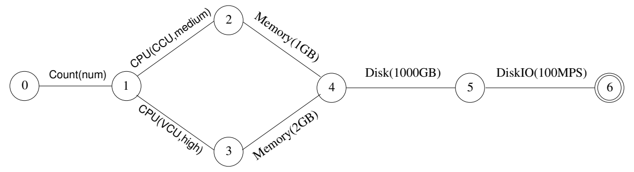

Figure 3: Alternative iVO RDG, double circles indicate an accept state

Different structures in the graph map to logical combinations of AND (sequences), OR (Branches to separate accept states) and XOR (branches that terminate in the same accept state). If an edge belongs to two different sentences that terminate in different accept states, then the host must provide multiple units of that resource if it wishes to host both tasks. However, if the two sentences terminate in the same accept state, as in figure 3, then only one unit of each resource is required.

2.2 Edge Expressions

Classes of service such as those used in networking (e.g. ATM CBR, VBR, ABR, UBR[2]) are used to represent the expected behaviour of a resource consumer. Processes do not usually require constant CPU, Memory, and I/O. For this reason classes have been defined for CPU, memory, disk, as well as bandwidth. The defined classes for CPU are Constant, Variable, and Available CPU Utilisation (CCU, VCU, ACU). The class of service is encoded in the value parameter of the edge, for example, constant CPU bursting of 200 MFLOPS is encoded as {CPU(CCU, 200)} or can represent non specific values, such as high, medium or low CPU power. The mappings for determining availability are again included with the host.

2.3 Graph Modularity

The structure of an RDG makes composition and decomposition relatively straightforward as subgraphs may be removed and manipulated as required. An example would be

[2]Asynchronous Transfer Mode, Constant Bit Rate, Variable Bit Rate, Available Bit Rate and Unspecified Bit Rate

an application migrating some part of its functionality to another host. The application can remove the associated part of its RDG to create a sub RDG to send with the migrating process. Similarity if the process returns, the RDGs can be rejoined.

Another advantage is that common shapes of resource requirements (or *modules*) can be created and used by applications to quickly characterise a common problem. For example when expressing a simple computation CPU, Memory, and disk will all be required. The common modules can be combined to form a complete RDG for a given application at a reduced computational cost.

2.4 Resource Matching — Modules

The ease of joining and separating graph components makes the use of modules a viable option for improving matching efficiency. The biggest limitation of the RDG is the potential for inefficiency when graphs become large. One solution to this is to break large graphs down into smaller subgraphs. We have identified three orthogonal resource categories that form natural modules for matching:

- **Computational** resources may be categorised as performance or availability resources. Performance resources are measurable like CPU, exchanged disk I/O and exchanged memory I/O, these resources are generally related to a required application or mobile object demand. Availability resources are the amount of a resource required for execution, e.g. an amount of memory or number of licenses.
- **Network** resources include parameters such as, bandwidth, jitter, loss rate and locality. Other networking aspects can also be considered, such as firewalls, provision of Application Level Gateways, access to certain network bearers, network load sharing features and redundancy etc.
- **Existential** resources include the availability of J2ME, CODECs, particular software libraries, or licenses. Hardware existential resources include properties such as CPU architecture (e.g. SMP - symmetric multiprocessor architecture), printers and other peripherals.

Host resource profiles are decomposed into these modules and the modules are then compared against an application's RDG.

3. EXPERIMENTAL RESULTS

The experiments in this section give the results from two matching algorithms. The first algorithm called *path comparison* (PC) extracts each path (or sentence) and then compares the individual paths. This simple algorithm results in multiple comparisons against certain resources. A solution to this was to utilise the modules outlined in section 2.4 — giving the *module comparison* (MC) algorithm. Details of these algorithms can be found in [5].

The two key variables in matching RDGs are size and complexity. The following experimental results compare the matching algorithms over a range of different application RDGs and host resource profiles each with varying complexity. We have analyzed system traces as a basis for forming typical application RDGs. All tests were averaged over 100,000 trials and run on a Pentium 4 1.8 GHz with 512MB of RAM using Java 1.4 and Microsoft Windows XP.

3.1 Application Complexity

Application complexity is based on the number of paths to an accept state and the lengths of these paths. In practice application RDGs tend to be small with minimal branching, however we examine the impact of both these aspects on performance.

The first experiment looks at the impact of the path length on the computational time. The resource profile used is more complex than we measured from captured traces [5], it has at least two paths through every module and has over 20 different resources.

Figure 4 shows the relationship between path length and computational time. This graph illustrates the improvement of the MC algorithm over the PC, and provides a feel for the performance of RDG matching that would be achieved in a wireless scenario.

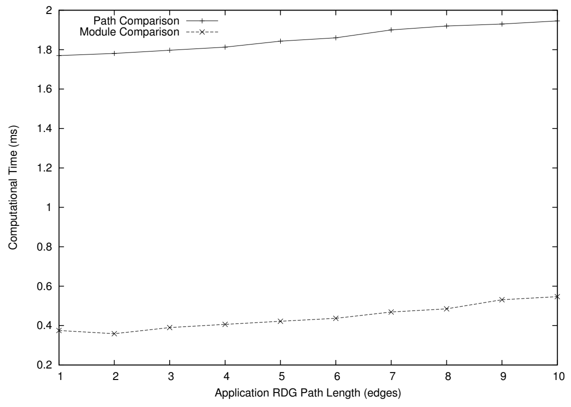

Figure 4: Application path length vs computational time

The results exhibit a small linear increase with length, the dominant cost is that of computing the resource profile paths.

The second factor in RDG complexity is the number of paths. In this experiment all the paths contain the same six resources in the same order. The resources in the paths were chosen to give an equal distribution across all three modules. Figure 5 shows the performance of the two algorithms as the number of paths increases.

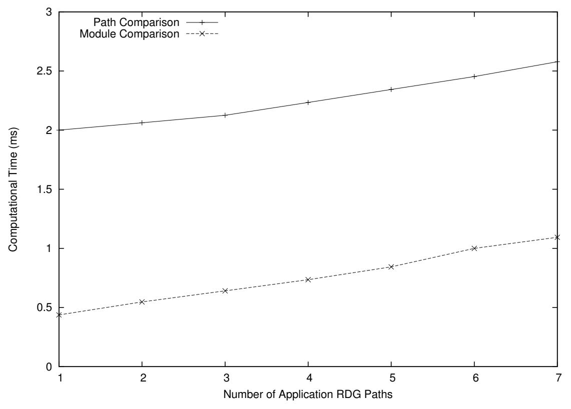

Figure 5: Number of application paths vs computation time

3.2 Host Complexity

Like application complexity, host complexity is effected by the number of paths and the length of these paths. However, the paths are typically short, and do not contribute significantly to the cost of matching (in the region of 20 to 90 microseconds). The final experiment discussed in this paper, looks at the effect of profile branching on the computational time of resource profiles. Additional experiments, results and data are presented in [5].

Figure 6 shows the computational time of PC and MC. It is clear that the MC algorithm outperforms the PC algorithm as the number of branches increases. At 12 paths it takes over 5ms which is 10 times larger than the MC algorithm.

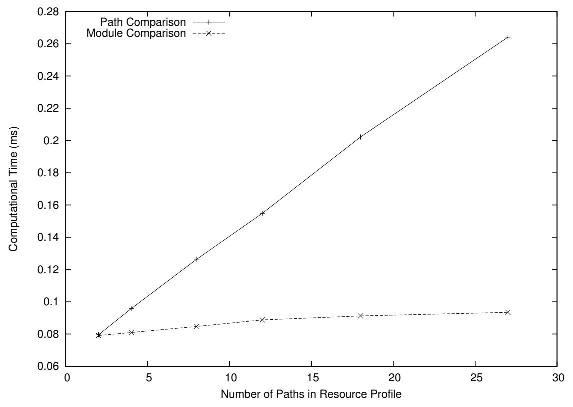

Figure 6: Resource profile branching vs computational time (with pre-computation)

3.3 Analysis

The experiments have shown that we can achieve computational times for average sizes of RDGs and Resource Profiles of less than 1ms. This computational time will be perfectly acceptable in most real situations, take for example a wide area distributed system — the latency required to transport the RDGs would far exceed the computational time required to evaluate them.

4. CONCLUSION

Resource specification is a vitally important ingredient in enabling mobile devices, software and adhoc networks to obtain, exchange or purchase execution and network resources.

This paper presents an efficient resource specification mechanism called the Resource Description Graph (RDG). The RDG is a single rooted directed acyclic graph that is passed between resource consumers and providers in a simple textual format. The RDG is able to encode application resource constraints and host availabilities thorough a sequence of edges. The RDG is a good resource specification mechanism that is sufficiently flexible for use in supercomputers and computationally limited devices such as PDAs. The RDG achieves this flexibility through the use of modules (subgraphs) to simplify composition and comparison.

Efficient matching was expected to be a major limitation of the RDG, due to the expense of finding subgraph isomorphisms. However, the experimental results presented in this paper quantify the low cost of matching RDGs. The use of modules results in excellent computational times of less than 1ms for realistic RDG examples.

This paper demonstrates that a resource specification can be encoded and processed in a format that is compact and modular. Most significantly, RDGs can be constructed and matched – with the limited computational power available from small mobile, pervasive or ubiquitous devices.

5. REFERENCES

[1] J. Birnbaum. Towards Pervasive Information Systems. Distinguished Lecture Series, UVC, December 1994.

[2] K. Bubendorfer. *NOMAD: Towards an Architecture for Mobility in Large Scale Distributed Systems.* PhD thesis, Victoria University of Wellington, 2001.

[3] K. Bubendorfer and J. Hine. NOMAD: Application Participation in a Global Location Service. *Proceedings of MDM 2003, Lecture Notes in Computer Science, number 2574, pages 294-306*, January 2003.

[4] K. Bubendorfer and J. H. Hine. Resource Discovery and Negotiation in the NOMAD System. In *to appear in Proceedings of ACSC2005, The Twenty Eigth Australasian Computer Science Conference*, volume 27, Newcastle, NSW, Australia, January 2005.

[5] K. Chard. Efficient resource descriptions, 2004. Honors Report, MSCS, Victoria University of Wellington.

[6] T. Dimitrakos, D. M. Randal, F. Yuan, M. Gaeta, G. Laria, P. Ritrovato, B. Serhan, S. Wesner, and K. Wulf. An Emerging Architecture Enabling Grid Based Application Service Provision. In *Seventh International Enterprise Distributed Object Computing Conference (EDOC'03)*, pages 240–251, Brisbane, Queensland, Australia, September 2003.

[7] P. Eerola, B. Konya, O. Smirnova, T. Ekelof, M. Ellert, J. R. Hansen, J. L. Neilsen, A. Waananen, A. Konstantantinov, and F. Ould-Saada. Building a Production Grid in Scandinavia. *IEEE Internet Computing*, 7(4):27–35, July-August 2003.

[8] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan and Kaufmann, 1999.

[9] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *Proceedings of the International Workshop on Quality of Service*, 1999.

[10] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[11] Globus. The Globus Resource Specification Language RSL v1.0, 2000.

[12] P. Komisarczuk, K. Bubendorfer, and K. Chard. Enabling Virtual Organisations in Mobile Networks. *IEE 3G2004 Conference, London, October*, 2004.

[13] M. Solomon. The ClassAd Language Reference Manual Version 2.1, October 2003.

[14] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36(7):74–84, July 1993.