Resource management for global public computing: many policies are better than (n)one

Evangelos Kotsovinos
Deutsche Telekom Laboratories

Iulia Ion
International University in Germany

Tim Harris
Microsoft Research Cambridge

Abstract

The federation of authority in global public computing systems poses major resource management challenges, as different stakeholders may have different views on how server resources are to be apportioned. Ultimately, the complexity of declaring and managing potentially overlapping federated policies often leads to the absence of high-level resource management systems from global public computing platforms. In this paper we propose a practical system that allows the different stakeholders to independently express federated policies, provides mechanisms for resolving potential constraint overlaps automatically, and reaches decentralised resource allocation decisions. We demonstrate experimentally that the system scales gracefully, introduces only a very low performance overhead, and is suitable for operating in realistically large and complex settings.

1 Introduction

Global public computing systems such as Grids [1], PlanetLab [2], and XenoServers [3] are characterised by the distribution of management responsibility among multiple individuals and organisations — collectively termed *entities* or *stakeholders*. However, while *federation* is crucial for scalability and cost-efficiency, it introduces important resource management challenges related to *expressing federated policies* and *managing policy overlaps*, often leading to the absence of high-level resource management facilities from such systems.

Let us consider the following example: server Sergei, a member of such a platform, reserves some of its resources for private use. Sergei's LAN is administered by Lou, who limits the network bandwidth made available to Sergei's non-local users to 100Kbps. Additionally, Indy, an infrastructural authority — a XenoCorp in XenoServers, or PLC in PlanetLab — limits the network bandwidth given to commercial and academic users to 500Kbps and 300Kbps respectively.

Determining the amount of network bandwidth to be given to users who are both remote and academic or commercial at the same time is a non-trivial question of resolving such a policy overlap. Simply selecting the minimum reservation or limitation is inadequate; the amount of bandwidth to allocate to a high-paying commercial user who happens to connect from a remote network may be the minimum, maximum, average, or any other amount dictated by *high-level* institutional or contractual arrangements. Additionally, contrary to role-based access control [4], resource allocation decisions are *quantitative* — how much access to grant a user to a resource — instead of binary — whether to grant access or not.

Mandating that policies be declared only by infrastructural authorities (for consistency) is both technically and organisationally inadequate, as it presents a management bottleneck and reduces the distribution of responsibility, which is crucial for scalability.

In this paper we present our design, implementation, and initial evaluation of a scalable *role-based resource management* (RBRM) system for defining and combining federated policies for resource management in a decentralised way. Our work makes the following contributions:

- **Federation of policies.** Our system embraces the distribution of responsibility, by allowing stakeholders to maintain different, subjective policies and views of users' properties.
- **Overlap resolution.** Unlike prior systems, which generally assume policy consistency, we allow overlaps and provide a mechanism for explicit automatic overlap resolution.
- **Scalability and performance.** Our framework is demonstrated to be scalable and of almost unnoticeable overhead in a realistic deployment setting.
- **Openness.** Our system is not tied to a particular language or deployment scheme, making it applicable to a number of global public computing platforms.

Our framework and the mechanisms for resolving policy overlaps are described in Section 2. Experimental evaluation results demonstrating the system's expressiveness, scalability, and minimal overhead are presented in Section 3. Section 4 positions our work in the relevant research context, and Section 5 concludes.

2 Role-based resource management

RBRM operates as follows: *Users* request resources from a server, and servers accept, negotiate, or deny resource reservation requests based on policies and current *resource availability*. For grouping users, entities declare *roles* and *role entry conditions*, which determine which users are members of which roles. Entities specify *constraints* on resource allocation for the different roles. Finally, *constraint relationships* can be used to indicate how to resolve overlaps in policies. Roles, entry conditions, constraints, and constraint relationships are collectively referred to as *policy elements*. We describe how these elements are defined and how they interact in the following section, and explain how they can be safely declared in a federated manner in Section 2.2.

2.1 Policy description

This section describes the syntax and usage of the policy elements that can be declared. `This` font denotes defined entities and policy elements, and *this* font refers to variables and describes the generic format of policy elements. The inference notation used denotes that if the condition that is defined above the inference line holds, then the action defined below the inference line is taken. Variables are always bound to the value they get above the inference line, and the wildcard $*$ denotes "any authenticated entity" or "any value".

The `User` object represents properties and methods exported by the user in question. All policy elements are signed by the entity that creates them ($Declarer$, $Elector$, or $Constrainer$), in order to prevent forging and repudiation.

Role declarations. A role is a group of users to which common policies apply. A role declaration identifies a role called $Name$ created by the entity $Declarer$. This hierarchical structure allows for scalable and decentralised naming. The format of $RoleDeclaration$ statements is $Declarer : Name(Parameter1, Parameter2, ...)$, with parameters bound when users enter the role. Figure 1 shows the roles declared in the example introduced in Section 1.

Role entry conditions. These specify membership of other roles, user properties, or other conditions that a user has to meet to be deemed a member of a given role. A generic form of a $RoleEntryCondition$ is:

$$\frac{RoleMemberships \quad Expressions}{RoleMembership}$$

To express $RoleMembership$ statements, the notation $Elector \rightarrow RoleDeclaration$ is used to indicate that the specified elector entity asserts that the user in question is a member of the specified role. A user's membership of a role is not global truth, but truth according to the $Elector$ — this is inspired by previous RBAC approaches [5]. This allows different electors to maintain different views of role memberships, and present these views to other entities by declaring the corresponding entry conditions on them. How much these other entities value each elector's view is entirely up to them. This distribution of control benefits scalability, as discussed earlier.

As in Figure 1, Sergei defines that users connected to any network apart from his one should enter the `Remote` role defined by Lou. Since this definition is independent of any previous role memberships, only the expression involving the user's network is specified above the inference line. Similarly, entry to the `Commercial` and `Academic` roles is based only upon user credentials.

Constraint definitions. A *constraint definition* is associated with a role to express a reservation or usage limitation on a resource, and applies to all members of the role. It is expressed as follows:

$$\frac{RoleMembership}{Constrainer \rightarrow ConstrKind(Resource, Parameters)}$$

where $Constrainer$ is the entity imposing the constraint. $ConstrKind$ takes the form of one of `limEach` (limit each member), `limGrp` (limit all members collectively), or `rsvGrp` (reserve for all members collectively)[1]. $Parameters$ indicate the extent of the limitation or reservation on a given $Resource$. The constraints defined in our example are shown in Figure 1.

Constraint relationships When more than one constraint is applicable to a user request for a certain resource there is an *overlap*; the system needs to determine how much of the resource can be made available to the user. To facilitate this, we introduce *constraint relationships*, giving a series of pattern-matches for existing constraints, and then a replacement constraint to be generated in their place:

[1] Note that reservations are applied on a per group basis rather than allowing "reserve for each member" as a kind of constraint — that latter kind of reservation is not possible without being able to enumerate the group's membership, which is complicated in a decentralised, federated model.

$$\frac{ConstraintDefinition(s) \quad Expression(s)}{ConstraintDefinition}$$

In our example, in an overlap between `Commercial` (or `Academic`) and `Remote`, the two constraints are replaced with a new one limiting access to the average of what the two overlapping constraints dictated (300 Kbps in this case). Similar relationships to the one expressed in Figure 1 can be defined for more general resolution strategies such as "take the minimum of a set of `limGrp` constraints given for any resource".

2.2 Declaration of policies

For scalability and simplicity, all policy enforcement takes place in a decentralised manner on individual servers, and a policy element is only applicable on the server on which it has been declared. Each server comprises an *authentication* layer, a *tentative declarations* set, a *policy filter*, and a *confirmed declarations* set. The tentative declarations set contains all policy element declarations attempted on a server. The confirmed declarations set is not accessible to any entities other than the server itself and contains the policy elements that are taken into account when considering resource allocations on the server. The policy filter controls which of the tentative declarations are to be copied to the confirmed set.

Any authenticated entity can independently declare policy elements on servers, subject to the policy elements having been properly *signed* by that entity to be non-repudiable and unforgeable, by placing them in the tentative declarations set of that server. Role declarations and entry conditions declared by arbitrary entities offer a federated view of the role memberships but cannot directly affect the resource allocations taking place on a server. Thus, it is safe to subsequently copy all properly authenticated role declarations and entry conditions to the confirmed declarations set without further checks.

On the other hand, the association of role declarations and entry conditions with *constraints* does affect resource reservations and can only be performed subject to a further *authorisation* check carried out by the policy filter. This requires that the server owner defines defines simple authorisation rules on the policy filter, allowing constraints and constraint relationships from entities that the server trusts or has long-term relationships with — such as certain XenoCorps — to enter the confirmed declarations set. This involves the server endorsing the policy elements in question by placing itself as the *Constrainer* and signing the elements. The set of confirmed declarations is then passed on to the RBRM policy evaluation procedure.

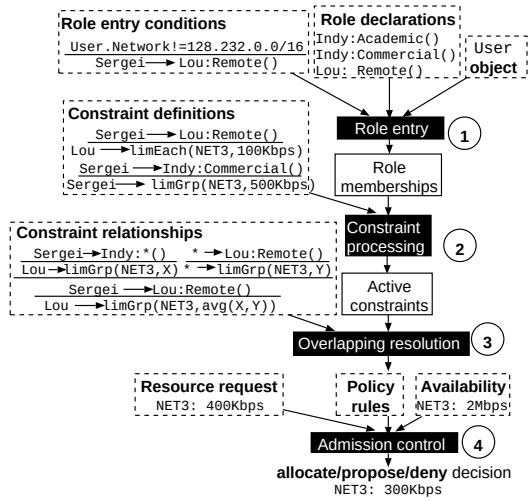

Figure 1: Policy evaluation process. A member of the *Remote,Commercial* roles requests 400Kbps. The two overlapping constraints (100 Kbps, 500 Kbps) are replaced by one granting access to the average (300 Kbps).

2.3 Policy evaluation

Given a set of policy elements and a user request the process starts with the *role entry* step, which determines which roles a user is a member of based on role entry conditions and user properties — operation 1 in Figure 1.

The next step — *constraint processing* — associates the roles the user shares with the constraints that apply to them. Constraints that are not associated with any of these roles are ignored further on, since they are unable to affect the admission control decision, reducing the set of constraints to the — potentially overlapping — *active constraints* — operation 2 in Figure 1.

The *overlap resolution* step checks the set of active constraints against the constraint relationships, resolving overlaps and replacing them by single constraints — operation 3 in Figure 1. The algorithm starts with deriving the sets of overlapping constraints by looking for active constraints attempting to impose different reservations or limitations on access to the same resource, for each resource. Then, for each of these sets, the algorithm uses any constraint relationships that are applicable to resolve overlaps, until there are no more sets of overlapping constraints. This produces a set of non-overlapping constraints (policy rules) applicable to the request.

Finally, the *admission control* module checks the current resource availability and usage against the policy rules to determine the *maximum allowed allocation* that can be made on each resource for the user in question — operation 4 in Figure 1 — and determines whether to grant or negotiate the request. In our running example, the system proposes the minimum between what the policy allows for the user, namely 300Kbps, and the available amount of resource NET3, 2Mbps.

Running the constraint solver in a decentralised manner and not on a central server facilitates scalability, fault-tolerance, and inherent incentive compatibility; nodes that choose to employ sophisticated RBRM policies contribute the local server resources required to run the constraint solver. Our goal is not to pursue a "global optimum" but rather to allow server owners and other stakeholders to define how resources are to be apportioned in a decentralised, federated way.

Conflicting constraint relationships. A *conflict* can occur when no constraint relationship is applicable to a set of overlapping constraints or when several relationships propose different replacement constraints. In case a conflict does occur, the approach our prototype takes is to either select the one that imposes the minimum replacement reservation or restriction or — if that cannot be clearly determined — request manual intervention.

Such conflicts will be relatively rare, as constraint relationships will be relatively few and defined infrequently, for the following reasons. Contrary to role memberships and constraints, which may depend on volatile properties such as the time of day or server load, constraint relationships express long-term, organisational relationships between the entities involved. Such relationships may denote the order of authority ("policies of entities of type A are more important than policies of entities of type B") or the general rules based on which overlaps are to be resolved ("when policies of entities of type A overlap, take the average").

We believe that this is simple yet sufficiently effective; a carefully defined set of constraint relationships allows completely automatic overlap resolution in most cases, requiring manual intervention only when there is a conflict that cannot be overcome by selecting the minimum reservation or restriction. In contrast, without constraint relationships manual intervention would be needed for resolving every constraint overlap, which occurs much more frequently than conflicts of constraint relationships.

3 Evaluation

We implemented a prototype RBRM framework and integrated it in the admission control stage of XenoServers — the point where a user requests the deployment of a new service in direct communication with the server, after authentication of the request.

The prototype is written in Java, supports the use of regular expressions, dynamic variable binding, and a large set of mathematical functions in the definition of constraint relationships, providing a very high degree of expressiveness. At the same time, it allows defining general constraint relationships such as such as "take the average of a set of `limGrp` constraints given for any resource", reducing the number of constraint relationships that need to be defined for resolving possible conflicts.

Our results demonstrate that the system can be used to reach resource allocation decisions in a large-scale, complex, realistic scenario, without introducing a noticeable time overhead in the admission control process.

3.1 Experimental setup.

The following parameters determine the complexity of the problem: *number of roles*, *number of constraints*, *number of overlaps per user request*, and *number of requests per second* — all *per server* that participates in the RBRM scheme. By altering the number of roles, we created scenarios named "10", "20", "30", "40", "50" — we anticipate that a typical server will very rarely have more than 50 roles declared. To consider the worst case scenario, we take the number of constrains equal to that of roles, i.e. we assume that each role has one defined constraint. We use different values for the number of overlaps per user request and the number of requests per second to obtain results regarding the scalability and performance of the framework. All experiments were run on a 3 GHz P4 with 512 MB of RAM, connected to a 100 Mbps LAN.

3.2 Experiments

The proposed resource management scheme needs to be lightweight enough so as to not slow down the operation of global public computing platforms considerably. It also needs to scale gracefully when policies get complex and the number of requests increases, and to be effective in expressing realistic resource management policies applied to existing systems.

Performance. It is important that integrating our RBRM framework with an existing global public computing system does not cause a significant performance degradation of the admission control process, as this would be noticeable to the end users. To demonstrate this, we measured the average time required by each subsequent step in the RBRM policy evaluation process in all scenarios, for single requests each generating 1 overlap on a given server.

The results of this experiment, shown in Figure 2(a), demonstrate that our framework can reach such a complex resource allocation decision in less than 150ms for scenario "50", most of which is needed for determining the role memberships of a user — phase 1 in Figure 1 — with only 25ms needed to resolve the overlap. This shows that it is possible to incorporate our RBRM framework in the admission control process of

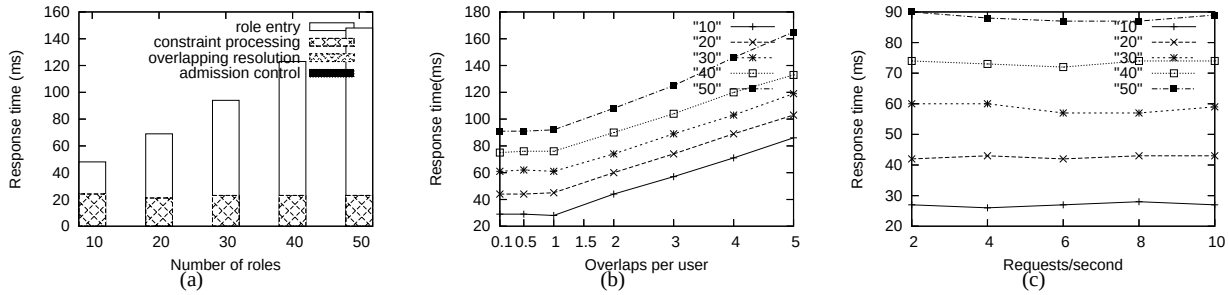

Figure 2: a) Execution time for each processing step. Response time when increasing b) the number of overlaps per user, and c) the number of requests per second

a real global public computing system without incurring a performance hit noticeable by the users — 150ms for a large system that manages up to 50 user roles.

Scalability. We performed two experiments. In the first one, whose results are shown in Figure 2(b), we focused on the effect of *policy complexity* on the system. We measured the time taken by the RBRM framework to reach a decision in all scenarios on one server and increased the number of roles and that of constraint overlaps generated by each request. The results show that the system scales gracefully, each role adding 1.6ms and each overlap resolution adding 15ms.

In the second experiment, Figure 2(c), we examined the effect of *popularity*. We measured the time needed to reach an admission control decision on one server when increasing the number of requests per second to the RBRM system. For normal request rates — i.e. since a request takes 150ms, we can have up to 6.5 per second — response time remains constant.

The results of both experiments demonstrate that our framework is able to scale to deal with both increasing complexity of policies and growing popularity, making it highly suitable for global public computing platforms. Scalability in terms of network usage was not measured as no inter-server communication is incurred.

Effectiveness. We examined various realistic resource allocation policies, of the kind suggested on public computing mailing lists and fora, such as the PlanetLab architecture mailing list[2] and the GGF's Policy Research Group[3]. In these sources we were not able to find a real-world resource allocation policy that our system was not able to express. As a few examples, using the policy description mechanisms described before we could express the following constraints — and their combinations: "Limit bandwidth to X Gigabytes per month to remote users", "limit bandwidth of traffic on TCP port 80 to X Gigabytes per month to all users", "allow dedicated use of only Y IP addresses by each local user", "guarantee bandwidth to X% of total bandwidth to premium users", and "reserve X% of CPU for users connected to the local network".

[2] arch@lists.planet-lab.org

[3] https://forge.gridforum.org/projects/policy-rg/

4 Related work

Since early RBAC research [4], several such frameworks have been devised. [6] defines roles as sets of rights and duties, and considers relationships between roles and meta-policies for resolving conflicts. [7] applies RBAC to open, large-scale systems. [8] proposes object-centric — similar to RBRM's concept of constraint definitions — and environment-centric. Ponder [9] associates positive and negative security policies with roles, and supports meta-policies for conflict resolution and role inheritance. Challenges of conflict resolution in the context of RBAC have been identified [10, 11]. PERMIS [12] provides a privilege management infrastructure and defines a hierarchical RBAC policy language.

The eXtensible Access Control Markup Language (XACML) [13] can be used to define access control policies and protocols, but does not support quantifying access to resources. This is also true of [14], an XACML-based policy management and authorisation service. Additionally, this service does not deal with overlapping or conflicting policies that are likely to appear in an open, federated environment.

Akenti [15] relies upon decentralised use-condition certificates, created using a public key infrastructure, which define what conditions a user has to meet in order to get access to a resource. Attribute-based access control [16], which makes access control decisions based on user attributes, is supported by our framework by defining single-property roles such as "Remote" and associating constraints with them.

Our framework draws on some of the techniques developed in RBAC, but differs fundamentally in its use of

quantitative policies, its emphasis on *federated control*, and the introduction of constraint relationships to control the way in which *overlapping policies* are combined.

Our system is complementary to work on service requirements negotiation [17], policy-based resource allocation [18], and negotiation and agreement protocols [19]. While such approaches focus on methods for specifying and agreeing on service requirements, RBRM provides a scalable system for federated policy specification and management. Our work is orthogonal to — and generic enough to be integrated with — low-level mechanisms for enforcing resource management decisions [20, 21] and incentive schemes for spreading resource contention [22, 23].

Condor [24] supports simple access control policies, but does not deal with *federated* and *overlapping* policies — only the machine's owner can define the policies used to manage a resource. Moreover, there is no mechanism for *grouping* users — policies need to enumerate individual users — or for *quantifying* access to resources — only accept/deny decisions can be made. We plan to experiment with deploying RBRM in Condor platforms to address the above challenges.

5 Summary

This paper has presented a role-based resource management framework that embraces the federated nature of global public computing systems, and maintains the principle of distribution of responsibility that lies at their core. By doing so, and by having proven an efficient and scalable solution, it offers global public computing systems a viable alternative to the current common practice of not providing any high-level resource management schemes at all.

In the future, we plan to perform wide-area deployment and testing of our framework on the XenoServer platform and other global public computing systems, and to devise tools that will simplify the process of declaring policy elements and distributing them to the servers. Additionally, we plan to investigate the expressiveness of our policy definition framework more comprehensively.

References

[1] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. GGF Working Draft, June 2002.

[2] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Comp. Comm. Review*, 33(3):3–12, July 2003.

[3] S. Hand, T. L. Harris, E. Kotsovinos, and I. Pratt. Controlling the XenoServer Open Platform. In *Proc. of the 6th Intl Conf. on Open Arch. and Net. Prog. (IEEE OPENARCH)*, April 2003.

[4] D. Ferraiolo and R. Kuhn. Role-Based Access Controls. In *Proc. of the 15th NIST-NCSC Conf.*, 1992.

[5] R. Hayton. An Open Architecture for Secure Interworking Services. Technical Report UCAM-CL-TR-399, Univ. of Cambridge, Computer Lab., June 1996.

[6] E. C. Lupu, D. A. Marriott, M. S. Sloman, and N. Yialelis. A Policy Based Role Framework for Access Control. In *Proc. of the 1st ACM Workshop on RBAC*, 1996.

[7] R. J. Hayton, J. M. Bacon, and K. Moody. Access Control in an Open Distributed Environment. In *Proc. of the IEEE Symp. on Sec. and Priv.*, May 1998.

[8] M. Covington, M. Moyer, and M. Ahamad. Generalized Role-Based Access Control for Securing Future Applications. In *Proc. of the 23rd Nat. Inf. Sys. Sec. Conf. (NISSC)*, October 2000.

[9] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proc. of the Policy2001 Workshop*, January 2001.

[10] E. C. Lupu and M. Sloman. Conflicts in Policy-Based Distributed Systems Management. *IEEE Transactions on Software Engineering*, 25(6):852–869, 1999.

[11] S. Reiff-Marganiec and K. J. Turner. Feature Interaction in Policies. *Computer Networks*, 45(5):569–584, 2004.

[12] D.W. Chadwick and A. Otenko. The PERMIS X.509 role based privilege management infrastructure. *Future Gen. Comp. Sys.*, 19(2):277–289, 2003.

[13] Organization for the Advancement of Structured Information Standards (OASIS). eXtensible Access Control Markup Language (XACML) Specification Set v1.0, February 2003. Available from `http://www.oasis-open.org/committees/xacml`.

[14] M. Lorch, D. Kafura, and S. Shah. An xacml-based policy management and authorization service for globus resources. In *Proc. of the 4th Intl Workshop on Grid Comp.*, 2003.

[15] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based access control for widely distributed resources. In *Proc. of USENIX Security*, 1999.

[16] P. Bonatti and P. Samarati. A unified framework for regulating access and information release on the web. *Journal of Computer Security*, 10(3):241–272, 2002.

[17] C. S. Yeo and R. Buyya. Service level agreement based allocation of cluster resources: Handling penalty to enhance utility. In *Proc. of the 7th IEEE Intl Conf. on Cluster Comp.*, 2005.

[18] C. Dumitrescu, M. Wilde, and I. Foster. Policy-based resource allocation for virtual organizations. Technical report, Grid Physics Network, 2003.

[19] P. C. K. Hung, H. Li, and J. Jeng. WS-Negotiation: An Overview of Research Issues. *HICSS*, 01:10033b, 2004.

[20] C. A. Waldspurger. Lottery and stride scheduling: Flexible proportional-share resource management. Technical Report MIT/LCS/TR-667, Massachusetts Institute of Technology, 1995.

[21] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose schedular. In *Proc. of ACM SOSP*, 1999.

[22] O. Regev and N. Nisan. The popcorn market — an online market for computational resources. In *Proc. of the 1st Intl Conf. on Inf. and Comp. Econ.*, 1998.

[23] A. AuYoung, B. Chun, A. Snoeren, and A. Vahdat. Resource allocation in federated distributed computing infrastructures. In *Proc. of the 1st Workshop on Operating Systems and Architectural Support for the On-demand IT Infranstructure*, October 2004.

[24] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for High Throughput Computing. *SPEEDUP Journal*, 11(1), June 1997.