Languages not Formats: Tackling Network Heterogeneity Head-on

Timothy Roscoe, ETH Zürich, Switzerland

1 Introduction

The networking community's response to all forms of heterogeneity in systems has traditionally been what might be termed *fixed abstraction:* provide single static representations that abstract away from the differences between resources, domains, hardware, etc. At domain boundaries, these fixed representations (for example, an RSVP resource description, or a BGP route advertisement) serve as the basic units of cross-domain communication.

From the viewpoint of distributed computing (in particular federated systems) from 15 years ago, this approach is hopelessly naive. Such a fixed schema, the argument goes, forces different administrative domains to adopt the same worldview, tends to encourage implementations to all look the same or be supplied by a single vendor, and makes it hard for any individual domain to innovate, unless the innovation is sufficiently incremental that it can be represented in the existing framework.

However, in a world which has set great store by the end-to-end argument, standardizing communication abstractions has been more highly valued than decoupling administrative domains. Arguably, this dogma is responsible both for the success of the Internet, and the ossification and stagnation it faces today.

Revisiting long-dead research into distributed federation concepts points out a different way forward: abstract *less* of the communication reality, and instead use a *richer* representation to allow systems to comprehend more of the heterogeneity they face, and handle it appropriately.

2 Example: GENI RSpecs

An excellent contemporary case of what can go wrong with even carefully designed fixed-schema descriptions is the Resource Specifier or RSpec in GENI [5]. The example is significant for two reasons: it is a crucial element of the current GENI architecture, and it aims to capture a much richer range of resource objects than previous work in networking.

RSpecs are specified in XML, using a schema defined in XSD. They have three uses: *advertising* the availability of resources, *requesting* resources that match a particular set of criteria, and *promising* resources, representing a commitment of resources to a particular client.

GENI's RSpecs are based for the most part on those used internally in the successful Emulab testbed [7] for describing resources. However, the Emulab environment constitutes a single administrative domain, and indeed until recently a single physical room. The format of the Emulab resource description has evolved over the years as Emulab's functionality has expanded.

The difficulties facing the designers of GENI's RSpec are twofold: firstly, capturing upfront the richness of possible GENI configurations (almost arbitrary combinations of links, virtual machines, tunable radios, forwarding hardware, etc.) in a single XML schema, and secondly, expressing a complex request (which might be satisfied by many configurations, with different levels of utility to the client) in the *same* format using wildcards.

In both cases, the challenge is imagining in advance all the possible things that must be expressible in an RSpec. The resulting format can express these things, but little else without explicit extensions. This circumscription of expressivity is a property of formats, but not of *languages*.

3 Back to the Future: ANSA

In juxtaposition to GENI's decision on a static format for resource advertisements, requests, and commitments, it is instructive to look at the ANSA trading model [3].

The Advanced Networked Systems Architecture or ANSA was a synthesis of state-of-the-art techniques in distributed computing in the late 1980s, and brought together results from a variety of previous research systems. ANSA strongly influenced ISO RM-ODP [4], TINA, and CORBA, and the ANSAware middleware was deployed in a number of commercial settings.

ANSA was designed from the outset to encompass multiple, federated administrative domains, with possibly heterogeneous implementations, protocols, and local architectural features. Trading was the term used in ANSA for the process by which service providers announced (or *offered* resources like services, clients requested resources, and offers and requests were matched together.

ANSA's trading model may look somewhat outdated in

our post-XML world. For example, advertisements ("offers" in ANSA terminology) only consisted of resource type and an unstructured set of name-value pairs.

However, what is interesting is that ANSA viewed viewed resource requests as a *constraint satisfaction problem*. Instead of a simple template, ANSA clients submitted a simple form of linear program over resource properties, in a declarative language. The trading service used this to return the k best resource offers it knew about, with the meaning of "best" being supplied by the client.

4 Languages not formats

For some reason, designers of networked systems have always stuck to fixed formats for communication between domains, but we argue that this position is pointlessly conservative today. We advocate using constraint satisfaction, but going beyond ANSA and using rich, declarative languages to represent both resources and requests for them.

Extensible formats are in an important sense *closed*: there are clear limits on what can be expressed. Extensions require agreement between domains before they can be interpreted. In contrast, languages supporting abstraction are *open* – a language is generative of an open range of possible descriptions, and unforeseen concepts can be expressed without extending the language itself.

Indeed, language-based representation of resources, policies, etc. is appearing in some areas of distributed systems and networking: RDF might be viewed as a (rather primitive) logic language, and security policies are increasingly expressed in a rich, declarative form because security researchers have realized that expressivity leads in the bigger picture to stronger and not weaker security.

The area opened up by the realization that inter-domain communication is more of a linguistic problem than one of protocols or formats opens up a much more fruitful area for progress.

Expressing state, requests, and policies in a language entails runtimes to interpret such a language. A runtime must ensure safety and liveness – a linguistic expression must not cause a system compromise, nor should it tie up resources in an uncontrolled manner. We expect such challenges to be met with a combination of automated analysis techniques and resource control mechanisms familiar in the field of OS design. We favor the use of declarative logic languages because they are both concise and amenable to both sets of techniques.

Furthermore, a *lingua franca* for inter-domain communication and representation is not in itself a solution to our problems. There will be a need for conventions that allow the language to be used conveniently between domains without requiring much prior exchange of information. The key observation, however, is that a language approach gives us far greater richness of expression and freedom to represent complex structures or policies than the limited formats we are used to designing.

5 Conclusion

The early designers of networks and associated protocols may have considered the use of constraint satisfaction techniques and rich data descriptions using logic languages (Prolog appeared as early as 1972 [2]), but either this didn't happen or it was ruled out as an approach, most likely because of complexity and lack of processing resources. As a result, today we have an impoverished set of ways of describing computer networks.

It is time to reconsider this position. Today the world is a more complex place, and the radical heterogeneity of hardware resources, application requirements, and service offerings cannot be handled by fixed abstractions. Processing resources in even quite simple network elements are now quite capable of efficiently executing the unification algorithm on quite complex expressions. If the GENI vision becomes a reality, either as a platform for network architectures [1] or a way to do away with networks *per se* entirely [6], a richer and more expressive representation of the world is a basic requirement for communication between domains of all kinds.

References

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. *Computer*, 38(4):34–41, 2005.

[2] A. Colmerauer and P. Roussel. The birth of Prolog. In T. J. Bergin and R. G. Gibson, editors, *History of Programming Languages*, volume 2. ACM Press, 1996.

[3] J.-P. Deschrevel. The ANSA Model for Trading and Federation. ANSA Architecture Report APM.1005.01, Architecture Projects Management Limited, July 1993. `http://www.ansa.co.uk/ANSATech/93/Primary/100501.pdf`.

[4] International Telecommunication Union. *ITU-T Recommendation X.901 / ISO/IEC 10746-1: Information tec hnology – Open Distributed Processing – Reference model: Overview*, 1st edition, December 1998. Available online at `http://standards.iso.org/ittf/PubliclyAvailableStandards/c020696_ISO_IEC_10746-1_1998(E).zip`.

[5] L. Peterson and J. Wroclawski. Overview of the GENI Architecture. GENI Design Document 06-11, GENI Facility Architecture Working Group, January 2007.

[6] T. Roscoe. The End of Internet Architecture. In *Proceedings of the 5th Workshop on Hot Topics in Networking (HotNets-V)*, Irvine, CA, USA, November 2006.

[7] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.