Workspaces

Paul Borrill
EARTH Computing, Inc
Palo Alto, CA 94306
paul@borrill.com

Mark Burgess
Independent
Oslo, Norway
mark.burgess.oslo.mb@gmail.com

Mike Dvorkin
Cisco Inc.
San Jose, CA
Mike.Dvorkin@cisco.com

Herb Wildfeuer
Cisco, Inc.
San Jose, CA
hwildfeu@cisco.com

***Abstract*—We summarize an old concept in a new guise: a multi-tenant and multi-user abstraction for ubiquitous systems, spanning datacentres industrial, agricultural, and domestic scenarios. Workspaces modernize current cluster and gateway abstractions, for managing multi-interest separation of concerns, both as a mapping between operations and resources, and as a unified approach for viewing distributed managed systems within a unified platform.**

I. Introduction

Delegation is a powerful way of scaling intent, cooperation, and responsibility within a distributed infrastructure [1]. Delegation goes hand in hand with the concept of roles and services. The challenge, in applying this to information systems, is to scale functional agency to multiple parties, in possibly non-contiguous collaborating regions.

Mechanisms for separation and federation of roles are well known in the form of multi-user operating systems, and, more recently, in some multi-tenant infrastructures like cloud providers: credentials are offered to different parties to authorize access to environments within a supervised resource framework [2]. These are currently best developed in the world of datacentres, where cluster managers and task schedulers have been designed to work within highly specialized and monolithic environments[1]. However, the tools can go further in simplifying IT infrastructure for use in all levels of society, from the home to industry.

The workspace concept follows on from [1] and summarizes a general 'multi-interest' abstraction for distributed cooperation, from a promise theory perspective [6]. A workspace is a set of distributed agents[2] with overlapping goals and resources, that are engaged in a context of human purpose. In contemporary language, a workspace could represent a scalable (distributed) container, or wrapper, for a set of cooperating applications and services.

II. Contexts and workspaces

What problem would workspaces solve? Consider a few human contexts as examples: some for work and some for pleasure.

> *A home, a shop, a line of business in a bank, a playground, a squash court, office, factory, building, a city district, emergency channel frequency, hot and cold water pipes, a dining room, a drinks cabinet or coffee station, etc.*

When we (i.e. users) approach such contexts, we suspend our sense of generality about the wider world, and our interest shifts to a more focused set of ideas, names and services, associated with the purpose of the space (this is sometimes called an ontology [7], [8]). Specialized spaces enable scaling by limiting scope to manage resources and expertise with greater focus and agility. Each context thus has an 'ontology', possibly with its own dictionary or preferred set of names, concepts, and services (see figure 1).

Resources, dependencies, and services, which furnish these contexts, may be bundled even when they are not directly part of a workspace's primary purpose, to help facilitate a simpler user experience. For example, when visiting a sports club, there are showers (one does not have to set up and tear down water or electricity supplies each time a tenant uses the gym); they are provided as a service, with the appropriate plumbing already taken care of, even though they have nothing to do with the game of squash. By the same token, parking, locker space, and balls may be used and reused by players for each game, but score records and competition results for all players would be stored for long term, and displayed for all to see. These human examples illuminate organizing principles that have evolved in the real world, and can be applied in an infrastructure context, catering for servers, devices, and applications (virtual devices), in an intermittently connected environment.

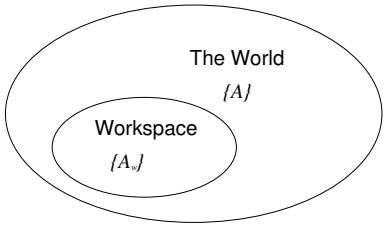

Fig. 1: Consider the set of all information resources, including humans, and an arbitrary subset of it.

At the time of writing, information technology lags quite a long way behind this standard: it offers component mechanisms for assembling catered environments with Application Programmer Interfaces (APIs) for Do It Yourself self-service. We expect every tenant to be a plumber, an electrician, and a housekeeper, rebuilding everything for each hotel visit. The current state of the art is analogous to electric circuits

[1] Mesos [3], Borg [4], Kubernetes [5] are systems designed for unconstrained resource environments, for example.

[2] An agent is an autonomous system player that expresses intended behaviour, either directly or by proxy, and whose default state is one of independence [6].

prototyped with breadboards and loose wires. For an intuitive Information Technology (IT) experience, one does not simply wire together devices and software releases into an IP circuit; rather, we need to start by putting them into a more complete managed operating system from the beginning.

In a seamless world, workspaces would encapsulate their utilities and consumables transparently, like rented spaces, and present a convenient, tailored service-interface to them for registered members. Providers a workspace don't need to update the entire phonebook (as in the DNS registry) for changes and facilities, only a local guest services menu (for a single context), such as one finds in hotel rooms, and conference centres, where some are private, some are shared, and some meetings simply happen in the lobby. The 'world', in figure 1, becomes a menagerie of mutually reachable autonomous agents: persons, devices, users, etc., that can join private meetings by membership registration, and by subscribing to their services.

III. Semantic containment of resources

There are plenty of concepts associated with *separation of concerns* in information technology (IT): scope, namespaces, workspaces, closures, partitions, sandboxes, are a few. There are also constructs for *aggregation of concerns*: contexts, containers, unions, packages, classes, etc. We may represent these structures purely within a framework of agent promises to see them as distributed systems [9].

Scope is a concept commonly used about variable visibility in programming languages. Containment is about packaging resources, for bundling resources for process execution, and limiting their footprint. Then, a context is a label an identifiable pattern of circumstances.

Namespaces are used in directory-based filesystems, IP subnets, class libraries, and also in the Domain Name System (DNS). Namespaces are associated with contextualization or 'disambiguation' of names. They place names into contexts that qualify them. All parts of a namespace may be accessed from the outside by providing a so-called *fully-qualified name*. Examples include directory filesystems, in which a fully qualified name includes a directory path from the root filesystem, or the Domain Name System (DNS) names, where fully qualified names include a domain-name or namespace identifier in addition to an IP-address alias. e.g.

```
section:name
class.member
directory/name
name.parentdomain.tld
IP subnet prefix . IP endpoint number
```

A namespace thus simulates a local horizon for the contextual relevance of identifiers. It creates labels for sections into which names can be grouped, and allows the same name to be adapted for different contexts with different meanings. If an agent (e.g. a software application) expresses a state that can be said to match a particular context (e.g. being 'in' a directory belonging to a file system), its default visibility will be only the range of resources and meanings within the same context. This does not mean there is an impediment to discovering names beyond the horizon, but one generally has to go through a door to see beyond them.

A *workspace*, on the other hand, is defined as a collection of agents that maintain a 'protected space'[3], with a distributed access perimeter. It is like a 'cluster' of agents, providing and consuming services, with a set of high level abstractions, and protections. Its aim is to limit the exposure of its members to the world beyond, and similarly of the world to what goes on inside the workspace. It is like a login account in a multi-user operating system. Access to enter it, and for information to cross the perimeter, must be granted. Workspaces are thus associated with security and privacy, as well as the labelled bundling of shared resources.

It would be not be accurate to think of workspaces as distributed firewalls, implying a disjointedness: they cannot be fully disjoint if they are to succeed in enabling collaboration. We know, from experience of multi-user systems, that a society of users will want to form user groups to share certain resources, some of which will span multiple organizational structures, because user concerns are multi-dimensional.

Imagine, as an example, a city in which building tenants each have their own billboards and facade lighting. Under normal circumstances each tenant would have its own umbrella workspace, with sub-divided workspaces to allow building engineers and business owners to manage their own complicated lighting issues. Now suppose that, as part of festivities, the city asked to coordinate the facade lighting of the buildings to create a common display. A new cross-company umbrella workspace could then be established to offer a cooperative channel between the engineers of the private tenants, without opening each other's full business systems to one another. One could even imagine emergency measures to use the lights of a city to guide a damaged aircraft to a landing strip, as an extension of the runway lighting. Flexibility to collaborate, with clear lines of autonomy, is the key idea.

IV. Localization for accessability

It is efficient to keep resources as close as possible to their point of usage to minimize latency and cost. Currently, public cloud systems transport vast amounts of data across networks to provide processing and storage capacity to remote users. This has scaling limitations as well as latency penalties that can be solved by workspace locality. More recently, cameras and other consumer 'things' are able to talk with smartphones, as an anchor point, rather than going directly to the cloud, avoiding pointless data costs, while allowing forwarding at will. Such an approach is easily extended, by policy and self-organization, to furnish a hands-free approach for multiple devices. A role for workspaces is thus to bring flexibility, as well as branding intent, based on context, specialization, and policy-guided relationships.

Even now, considerable resources are available in most geographical locations, if only they could be consumed as easily as cloud services. This suggests that we should look for locally integrated (so-called 'hyperconverged') platforms that promise resources in proximity, but are still managed by utility providers[4].

[3]This is like a distributed firewall, but without a single gateway.

[4]Today, we might call this the "Uber" of infrastructure.

A hierarchy of embedded resources, including caching and redirection, would improve service performance, and enable a simpler security architecture too. This type of approach is already familiar as the approach employed in Content Delivery Networks for scaling of publish-subscribe 'pull' models[5].

V. Principles for multi-tenant platforms: membership and containment

Operating system research in the 1980s led to several fully distributed operating system projects with workspace abstractions, including Plan 9 [10], Spring [11], Amoeba [12]. Workspaces were user login accounts, with access controls. Multi-user platforms typically enable two forms of support for work delegation, based on trusted users and trusted software:

1) Separation and fair sharing of dedicated resources (user login accounts by name identifier), with possibilities for group collaboration. Agents (humans, devices, programs, etc) play the role of users.
2) Role-based privilege escalation through *trusted programs* (setuid, sudo, capabilities etc). Role Based Access Control (RBAC) is a way of maintaining minimal exposure and of separating concerns, based on patterns rather than name identifiers.

These capabilites are based loosely around two well-known models of 'security' that loosely represent their basic mechanisms: the Bell-LaPadula model [13] and the Clark-Wilson model [14], respectively. In a distributed system, promises about security must be made and kept in each agent individually, as no agent can make a promise on behalf of another. This exposes a much wider surface to the world. Thus we have to address the perimeter problem.

The goal of a workspace, like a multi-tenant, multi-user platform, is to create a supervised abstraction, for agent collectives, that makes a collection of cooperating agents have the same kinds of properties as a single atomic agent, despite being delocalized. Maintaining such a delocalized perimeter with access control, by distributed cooperation, is a surprisingly difficult construction. Indeed, this is no doubt why so many mistakes are made in security.

VI. Summary and Remarks

We still think of management as a brute force operation, wielded from a central controller, thus we assume that embedded devices are too weak or too ad hoc to play a role in their own management. It seems no longer sufficient to arrange abstractions suitable for datacentres alone. Cloud computing assumes that management must accompany centralization to a remote datacentre, but this is for provider rather than customer convenience. Two specifically desirable properties of a solution are:

- *Modularity* or logical isolation of different interests, to avoid contamination, disruption, and to be able to understand systems.
- *Locality* so that workspaces will be able to place resources (compute and storage) close to the source of action, like 'local service branches' or customer service points, wiht low latency.

Both these arguments suggest a long-term decentralization or break up of what we call 'the cloud' to encompass mobile and embedded devices. Users are already bridge the cloud, with domestic and mobile devices, but today that cloud is quite inhomogeneous with respect to resource distribution. It will become more homogeneous as technology develops and domestic and industrial applications are identified and developed [15], [16]. As this happens, the distinction between datacentre, embedded services, and users will gradually vanish altogether.

[5]Read (pull) and write (push) systems scale very differently. IT has been slow to move away from push-based thinking for both read and write that have scaling limitations.

References

[1] P. Borrill, M. Burgess, T. Craw, and M. Dvorkin. A promise theory perspective on data networks. *arXiv:1405.2627 [cs.NI]*, 2014.

[2] M. Schwarzkopf. Operating system support for warehouse-scale computing. Master's thesis, Cambridge University, October 2015. http://www.cl.cam.ac.uk/ ms705/pub/thesis-submitted.pdf.

[3] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[4] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, pages 18:1–18:17, New York, NY, USA, 2015. ACM.

[5] Brendan Burns. How kubernetes changes operations. *;login:*, 40(5), 2015.

[6] J.A. Bergstra and M. Burgess. *Promise Theory: Principles and Applications*. $\chi tAxis$ Press, 2014.

[7] J. Strassner. A model driven architecture for telecommunications systems using den-ng. In *Proceedings of the 2nd International Conference on E-Business and Telecommunication Networks (ICETE)*, pages 118–128, 2004.

[8] J. Strassner. *Handbook of Network and System Administration*, chapter Knowledge Engineering Using Ontologies. Elsevier Handbook, 2007.

[9] D. Aredo, M. Burgess, and S. Hagen. A promise theory view on the policies of object orientation and the service oriented architecture. In *submitted to Science of Computer Programming*.

[10] R. Pike, D. Presotto, S. Dorwood, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom. Plan 9 from bell labs. *Computing systems (MIT Press: Cambridge MA)*, 8:221, 1995.

[11] J. Mitchell. Introduction to "an overview of the spring system". *Sun Microsystems Laboratories: 10 Years of Impact.*, 2001.

[12] F. Douglis, M.F. Kaashoek, A.S. Tanenbaum, and J. Ousterhout. A comparison of two distributed systems: Amoeba and sprite. *Computing systems (MIT Press: Cambridge MA)*, 4:353384, 1991.

[13] D.E. Bell and L. LaPadula. Secure computer systems: Unified exposition and multics interpretation. *MITRE technical report, MITRE Corporation, Bedford Massachusetts*, 2997:ref A023 588, 1976.

[14] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, page 184, 1987.

[15] The internet of things - concept and problem statement, January 2013. version 5.

[16] M. Burgess and H. Wildfeuer. Federated multi-tenant service architecture for an internet of things. https://tools.ietf.org/html/draft-burgess-promise-iot-arch-00, October 2015.