The Semantics of Workspaces

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

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

***Abstract*—Developers frequently expend time and effort reinventing common abstractions to support distributed applications, both for user-facing services and interoperability. This leads to a lack of consistent semantics for users and developers, leading to inefficiencies and even dangerous misunderstandings. We summarize a modern retake of well-known concepts, for generalizing multi-tenant application semantics, in highly distributed systems. By implementing some standard infrastructure, one could easily enable both the self-documentation and friendly coexistence of multiple applications, at any scale.**

I. Introduction

Human workspaces are a familar part of everyday life, and the virtual imitation of workspaces for individual users has been imagined as desktops and login accounts on multi-user communities. Workspaces have also been proposed as a technology abstraction for fully automated distributed applications, summarized in [1]. The technical meaning of a workspace is to combine the notion of a 'namespace' or private context, with access controlled or scoped resources. The semantics of workspaces are about specialized and protected roles, and cooperation between them.

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 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 pervasive use in all levels of society, from the home to industry, for what we are currently calling 'The Internet of Things'.

The workspace concept, we are advocating, follows on from [6] and summarizes a general 'multi-party interest' abstraction for distributed cooperation, based on promise theory models [7]. A workspace is a set of distributed agents[2] with overlapping goals and resources, that are engaged in some human purpose (running an application or a service). In contemporary state of the art, a workspace could represent a scalable cluster or (distributed) application containers, associated with some role or function, for a set of cooperating applications and services. In the future it might represent a variety of packagings, from virtual machines to unikernel collectives.

[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 [7].

II. Context and semantics in workspaces

In the human realm, workspaces enable specialization and compartmentalization. Consider the following examples

> *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.*

Each of these contexts are associated with special semantics, i.e. formal and informal interpretations about how we should interact with them, and what the interactions represent, when the system is in a good or a bad state, and so on. In these 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 [8], [9]). These specialized spaces enable us to cope with scale, by limiting scope both to manage our 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,

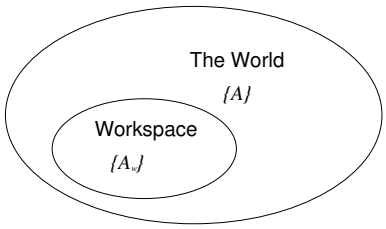

Fig. 1: Consider the set of all information resources, including humans, and an arbitrary subset of it.

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.

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 visit. The current state of the art is analogous to electric circuits prototyped with breadboards and loose wires. Two particular issues stand out: containment of functionality, and interoperability of components. For an intuitive Information Technology (IT) experience, one does not simply wire together devices and software releases into an IP circuit; we need to start by putting them into a more complete managed operating system, with standardized semantics. To do this we need to understand how semantics scale [10].

III. Semantic labelling of resources

To make specializations work like human abstractions, we can design semantics to represent a few basic things before attempting complex logical reasoning [7]:

- Requirements or expectations ('use' promises).
- Behaviours (service promises).
- Dependencies, associative cooperation, and processes.
- Collective state representing context.

Practically, semantics fall into three practical uses:

- Self identification (promises).
- Interactions (dependencies).
- Observation, calibration (assessment).

Semantics are formed from a notion of short term state, representing current state or *context*, and a long term state composed of intentions and relationships, or *promises*, that are activated or modulated by current context. This describes the proven model of CFEngine [11], which is now being rediscovered in modern scheduling tools like Kubernetes [5]. It is an ontology-free non-deterministic state machine approach to associative knowledge representation [12].

Standard IT doctrine is to build systems from the top down, by authority and obligation [13]. In this view, ontology is imposed from the top, and all users are expected to conform. This leads to fragility and inconsistency. Today, business agility is changing this, forcing bottom-up thinking, autonomy, and self-management, with no master authority: hierarchy is used to aggregate observations and unify experiences rather than to disseminate control. This is in accordance with the principles of promise theory [7].

For more than a decade, researchers have tried repeating the top-down schema approaches like SNMP's Management Information Base (MIB), and Common Information Model (CIM), even applying the semantic web technologies to model system behaviours. These efforts consistently failed, because they were based on top-down branching logic [12], [14], leading to an exponential 'many-worlds' or 'split-brain' problem. Instead, a scalable workspace concept can be approached from a bottom-up (promise oriented) perspective. The key technology for implementing and scaling both performance and functional semantics is the *nameservice* (i.e. generalized index, or directory) [10]. This is a unifying technology for transparency, service discovery, coordination, and scaling.

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 *workspace* is, roughly speaking, a namespace that may contain private resources, with access control, like a walled garden[3], with a distributed access perimeter. It is formed from a cluster of agents, which provide and consume services, with a set of high level abstractions, and protections. Its aim is to both collate local information, and limit the exposure of its members and their interactions to the world beyond, and vice versa. Workspaces are associated with security and privacy, but also with shared semantics, for internal interoperability and external services, as well as the labelled bundling of shared resources. Such semantics cannot realistically be standardized beyond workspaces.

IV. 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 [15], Spring [16], Amoeba [17]. 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.

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

System security generally has very weakly defined semantics, based on generic capabilities such as access control and encryption. This weak understanding of what is meant (and intended) by security is probably why it is so easy to blame security when unexpected exploitations of system behaviours take place.

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. This includes a common approach to shared services, like access control, communication, storage and computational services. The cost of cooperation is that namespaces make it harder for specialized agencies to discover and understand one another, or to collaborate in order to promise greater goods. A naming service that represents extensible, schemaless semantics could solve this quite easily. However, present day directories services (DNS, LDAP, Consul, etc) are not up to the task [18]–[20]. DNS is antiquated and top-down. Modern services like Consul and 'etcd' are limited in support for multitenancy and context. The hierarchical structure to support workspace semantics may be very generic [12]:

```
namespace
   bundle / concern
      context::
         promise
```

Again, this is very close to the CFEngine model, which can be applied directly to the pressing issue of bringing predictable environments into a workspace abstraction. Maintaining a delocalized perimeter with access control, by distributed cooperation, is also a surprisingly difficult construction, but it is within our technical capability. Labelling the semantics, in a schemaless, but nonetheless partially standardized way brings enormous flexibility.

V. Summary and Remarks

Two specifically desirable properties of workspaces 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 of these are properties that also promote stable semantics, without imposed standardization, by bringing predictability to systems and limiting scope. This we may add to this a third: application and service semantics.

Building up a unified context from simple and flexible schemaless primitives is an approach that has been proven via CFEngine, and can be modernized for a more mature generation of technologies. The long-term decentralization or break up of what we call 'the cloud' to fully include mobile and embedded devices in a single framework [21], [22]. 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 physically more homogeneous, as technology develops, and domestic and industrial applications are identified and developed; however, it will become semantically more diverse, as local ecosystems are enabled to flourish without restriction or restraint.

References

[1] P. Borrill, M. Burgess, M. Dvorkin, and H. Wildfeuer. Workspaces. Concept description, 2015.

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

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

[8] 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.

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

[10] M. Burgess. Spacetimes with semantics (ii). *http://arxiv.org/abs/1505.01716*, 2015.

[11] M. Burgess. A site configuration engine. *Computing systems (MIT Press: Cambridge MA)*, 8:309, 1995.

[12] M. Burgess. *New Research on Knowledge Management Models and Methods.*, chapter What's wrong with knowledge management? The emergence of ontology. Number ISBN 979-953-307-226-4. InTech, 2012.

[13] Mark Burgess. An approach to understanding policy based on autonomy and voluntary cooperation. In *IFIP/IEEE 16th international workshop on distributed systems operations and management (DSOM), in LNCS 3775*, pages 97–108, 2005.

[14] M. Burgess. *In Search of Certainty: the science of our information infrastructure*. Xtaxis Press, 2013.

[15] 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.

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

[17] 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.

[18] Hashicorp. Consul, service discovery and configuration made easy. https://hashicorp.com/blog/consul.html, 2014.

[19] CoreOS. Etcd, a highly-available key value store for shared configuration and service discovery. https://coreos.com/etcd/, 2014.

[20] B. Reed and F.P. Junqueira. A simple totally ordered broadcast protocol. pages 2:1–2:6, 2008.

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

[22] 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.