The Arusha Project: A Framework for Collaborative Unix System Administration

Matt Holgate – University of Glasgow
Will Partain – Arusha Project

ABSTRACT

The Arusha Project is an independent open-source project centered on the premise that the best hope for Unix system administration at modest-sized sites is through large-scale Internet-wide collaboration. We present a simple *object model* as a thinking tool, and an XML-based *configuration language* as a concrete notation for expressing system-administrative facts. We show how this framework allows a gentle evolution from current practices, but gets us quickly to very powerful ways of working.

Introduction

Fifteen years ago, people would stare at you blankly had you suggested a major operating system might be written by a few hundred people scattered around the world in their spare time (more or less).

Today, people stare at you blankly if you suggest that many thousands of Unix system administrators[1] might manage their sites *collaboratively*, with no organization to 'set standards' and no single concept of what 'good' administration is. Welcome to the central idea of the Arusha Project!

The Arusha Project (ARK[2]) is an independent 'itch-needing-scratched' open-source effort that sprung up in Glasgow, Scotland, in 1998. Whether it will see its vision of rampant collaboration fulfilled is a mostly sociological matter.[3]

For system-administrative collaboration beyond a well-tuned mailing-list, a more formal *lingua franca* (a 'trade language' used between people of diverse mother tongues) is essential. The core ARK technology, its *configuration language*, is our contribution to this end. The ideas behind it are innovative, and its design is a delicate brew of minimalist choices.

This paper outlines the context that we care about, followed by a sneak preview of our final results. We then set out some key engineering choices, plus a word or two about the wider collaboration picture. The ARK configuration language is the main technical content, followed by a review of what we gain, collaboratively speaking.

The View From Mount Meru[4]

The Arusha Project is hugely informed by its target context, so it is worth explaining that in some detail.

Picture: To your left you see a pile of just-delivered Unix workstations (and servers, and networking bits, and . . .), and to your right you see a marauding herd of users with work to do. You, the glorious administrator, are to assemble the pile of boxes to your left into the most important tool of the mob to your right, helping them toward staggering, competition-wilting job effectiveness.

Now move the clock forward by ten years. Most of those original computers have been retired, replaced spasmodically by others (budget gods willing). That known-to-work-together pile of equipment has turned into a made-to-work-together hodge-podge. Duct tape has been used. People, too, have come and gone.

The administrator's goal will not have changed. The amalgam of equipment should still comprise a single 'tool' that is perfectly tuned for its users' effectiveness. (Better than that, actually: a good system will continuously *anticipate* the workplace's needs a year or two ahead.) All with great uptime, no unplanned outages, perfect backups, and apple pie for all.

Our picture provides a formidable administrative challenge in the context we see: 4-400 hosts, with a comparable number of users in an innovation-oriented workplace.[5] Some employers (e.g., investment banks) may spend enough money to collect a surfeit of heavy-hitting administrators. Other employers may be wise or lucky enough to hire clever LISA readers. But, all too often, you will find one-or-a-few under-resourced administrators doing their best but not winning. If you hear the phrase, "We spend all our time fire-fighting," then you have found our target audience.

Our beleaguered fire-fighters will be doubly frustrated if they are cursed by wanting to aim high:

[1]Hereafter, just 'administrators' (and 'administration,' 'administrative,' . . .).

[2]'ARK' is Arusha's airport code (*Arusha* airport, not Kilimanjaro International, JRO). We use 'ARK' and 'the Arusha Project' interchangeably.

[3]*Peopleware* argues convincingly that 'technology' and 'process' are always less important that 'people' issues in a technical endeavor [DeM87].

[4]Arusha is a town in northern Tanzania that sits at the foot of Mt. Meru, which is 14,979 feet high, the fifth highest mountain in Africa. What better place for a general overview?

[5]Such a workplace benefits from a rich and arguably over-provisioned computing environment, with considerable information flow and exchange.

wanting to build a wonderful system. (And is it fun to do anything else?)

We now outline some of the main problems (labeled with **P-***n*) and goals (**G-***n*) that face our overeager fire-fighters. (The labels are so we can refer back to them.)

P-1 *Complexity is unavoidable:* No matter how you go about it, 4-400 Unix hosts, well stitched together, will make for a *complex* system; and managing complexity well is hard work.

P-2 *Mediocrity is natural:* A modern Unix system is composed of dozens, if not hundreds, of subsystems (kernel, DNS, sendmail, RAID, printing, . . .), each of which is a complicated beast. You can often make a Good Living Indeed by being seriously good at *just one* of these subsystems. An administrator who spends a morning with one of these subsystems has to accept that s/he is probably mediocre at it.

P-3 *Wayward buses are a threat:* It is all too easy for essential site knowledge to live *only* in the head of one administrator, but if s/he is run over by the proverbial bus . . .

P-4 *Isolation:* In our target context, an administrator often works alone, or a small team works at a level far below 'critical mass.'[6] Solutions, scripts, documents, etc., are *unlikely* to receive any independent scrutiny; this is *not* a recipe for robust and powerful systems.

P-5 *Wheel reinvention:* This tendency is fairly natural in an isolated systems ecology, and human hubris tends not to help. Also, the site-specific nature of scripts that administrators write weakens their value as reusable components.

P-6 *Administration is more than fiddling with* /etc*:* Just as there is more to software development than design and coding, there is more to administering a Unix system than managing hosts, users, and software applications.
Administrators have lots of other 'things' to manage: vendor records, maintenance contracts, old purchase orders, licenses, serial numbers, spare-parts inventory, helpline numbers and addresses, e-mail about all of the above, and so on.
One may choose to have no formal management of such 'things,' with the attendant risks. However, as soon as you turn such information into bits to be cared for, you have entropy (or 'bit rot') to worry about. Phone numbers change, new releases come out, the purchasing department 'improves' their procedures, etc.

P-7 *Systems have a multi-decade lifespan.* Computing infrastructures tend to be evolving, long-lived structures. It is not rocket science to spend lots of money this year and have a great system for users *this year*. What *is* a real achievement is to keep entropy at bay for ten or twenty years and still have a sweet system running, budget vagaries and turnover vicissitudes notwithstanding.

P-8 *Administrators are not, by and large, programming-enthralled.* Administrators admire a good Perl script as much as the next guy, and are more than willing to roll up their sleeves and sling a little code. But tell them that they 'only' need to grasp (say) Hindley-Milner type inference [Mil78] to make some Great Leap Forward, and they will (mostly) respond, "No thanks."

P-9 *Working examples to learn from are rare.* Many administrative tasks, e.g., setting up a mailing-list server, are an amalgam of small tasks. The README file typically says, "You could do this, or perhaps that." What you really want to know is what did someone, anyone, *actually do* to complete the job. Even better would be several real, working, non-obsolete examples to study and work from.

G-1 *Administration should be 'site-at-a-time.'* If your administrative gestures (running a command, clicking a GUI button, . . .) have small effects, e.g., adding a single line to a file on one host, then it takes many gestures to do the job (low productivity). We want high-impact gestures that make something true for the *whole site*. But more important than automated ways of doing things is *thinking about* the system in 'site-at-a-time' ways. (This is the same impulse as the 'infrastructures' work [Tra98].)

G-2 *All added-value matters.* Recall our initial picture of just-delivered boxes to the left, users to the right . . . We consider *all 'value-adding' activities* that lead to a useful-to-users system to be 'system administration' and therefore within the Arusha Project domain.
Setting /etc/resolv.conf correctly for all hosts is in play (nothing new there . . .), but so is the hpux-admin article about kernel tuning that you saved. So are notes of a phone call to Sun support. All are part of the total 'added value' that makes the system what it is.[7]
This goal says that we need something analogous to the 'methodologies' of the software domain. The observation there is that software development is much more than just frenetic coding, and all of the many other artifacts and activities that go into making software (requirements gathering, unit testing, project plans,

[6] We define a team at 'critical mass' as having enough collective brainpower to get on top of the problems immediately at hand, and to have enough cycles left over to engage with the wider world (e.g., read LISA proceedings) and to do some speculative projects, to explore new technologies, etc.

[7] Aside: this paper concentrates on what we do *to the boxes* on the left in order to induce great work in the user population. Actually, we consider it equally within an administrator's remit to Do Things *to users* (training, cajoling, altered work practices, new ways of thinking, etc.), if that makes the total result better.

etc.) need to be 'managed' and dealt with inside some overall 'process.'[8]

G-3 *Simplicity.* In this line of work, simplicity of design and implementation is invariably rewarded.

G-4 *A system should have its 'source code.'* We want as much of administrators' 'added value' to be expressed as bits-on-disk as possible, if only to avoid the Wayward Bus problem (**P-3**). We view the bits created directly by administrators (scripts, notes, e-mail, web pages, etc.) to be the *source code* for the system.

Of course, the idea of 'source code' implies something stronger: with nothing but the source code and raw vendor-supplied hardware/software, the overall system should be entirely reproducible.

G-5 *Once and Once Only.* Just as duplicate code is suspect in programming, a system's source code should have the same "once and once only" property.[9] If a site has a hub-and-clients sendmail configuration, that should be expressed in *one* piece of 'source,' and *not* in files scattered hither and yon. It is good to look in a single directory and be able to say, "That is all there is to know about our sendmail setup"; it is downright perverse to cut-and-paste sendmail.cf fragments from host to host. That is malignant source-code duplication.

And if saying something once per site is good, once per planet is even better.

G-6 *It's great for sites to be different.* A common impulse in administration is to 'standardize' (hardware, software, people, . . .), nearly always as a way to cut costs. The notion is not entirely without merit.

'Standardizing' often bumps into an organization's *Immovable Local Realities* (ILRs). These are less-than-ideal local facts or components that simply must be factored into any system solution. Examples might include: poor cable ducts, a cantankerous old plotter that is essential to the enterprise, some key software whose supplier has long-since gone out of business. ILRs often guarantee that administrators must deal with 'not as standard as we'd hoped' solutions.

Another worry with 'standardizing' is that it increases the risks associated with single-vendor solutions. This year's 'obviously the way to go' can crumble if your vendor loses a few key people, or stumbles into an unfortunate lawsuit. A deeper problem with 'standardizing' is that it is often at the expense of *competitive advantage*. If you are committed to making your users more effective than the competition, you will have to supply something extra, something different that the other guys cannot just order off the web. Decreeing "A Windows PC for everybody!" is not an option that your competition somehow stupidly overlooked.

G-7 *Presentation matters:* While there have been many Unix sites that were rigorously managed, there have been many fewer where this was manifestly obvious to someone other than the administrator who did the work.

Cairo[10]

We have established a setting for ARK, and will shortly describe the ARK configuration language and 'engine.' But, you cry, "Cut to the chase scene! What might I do with this stuff at my site?" Here are some deeply hypothetical examples, barely explained.

Build all packages for all hosts (even if of diverse platforms):

```
ark package install ALL
```

Verify the configuration of all Solaris hosts:

```
ark host verify sparc-solaris
```

Check that local mailing lists only have valid subscribers:

```
ark maillist chk-valid local-lists
```

Any support contracts about to expire?

```
ark support-contract list-expiring ALL
```

Notice how *many kinds* of administrative 'added value' are being managed in a consistent way.

Crucially, we want these (and many other) powerful 'site-at-a-time' commands to incorporate *both* local ways of working *and* top-quality 'patterns' from respected ARK sites around the world.

We hasten to add that this kind of power does not fall ready-made out of an ARK tarball. You must build up an ARK description of your site; but this local effort to make such powerful commands possible is modest. At one of our real sites, an average package (file) clocks in at 20 lines (690 bytes), a host at 32 lines (1140 bytes), and a 'disk chunk' [think of an automount-map entry . . .] at nine lines (295 bytes). Often, even these small files can be copied from a collaborating pal.

Volcanic Ash[11]

Our context and goals make for a big picture. Happily, the Arusha Project is *not* about filling in the whole picture; rather, we provide a *framework* within

[8] Actually, we are profoundly skeptical about the conventional 'processes' and 'methodologies' of the software world. But you are going to have an administrative 'methodology,' whether you call it that or not; so you might as well try for a good one.

[9] "Once and once only," usually written 'OAOO,' is a buzzphrase of Extreme Programming [Bec99].

[10] Arusha lies about halfway along the Cape-to-Cairo overland route, which is the direction most people do it. Cairo is their ultimate goal.

[11] The ground beneath your feet in Arusha is a powdery greyish volcanic ash; we hope the foundations of the Arusha Project, sketched in this section, are more solid.

which cooperating administrators world-wide can set about painting the canvas. Still, even a framework must make some up-front (engineering) choices that affect its shape and scope.

We first review a few simple pragmatic choices, noting how they tie into our problems and goals.

Small sites: Our target is comparatively small sites (4-400 hosts), and scaling issues do not keep us awake at night. Lots of people target the single-host site, and others are better placed to tackle the Big Sites ('enterprises').

High on the food chain (i.e., existing tools): We strongly prefer to build on existing tools (notably the standard Unix utilities), especially for the heavy-lifting parts of administrative tasks (**P-5**).

For scripting, administrators should still be able to use shell/Perl/Python. They *don't* have to learn a new language, throw out their old stuff and start again (**P-8**).

Textual tools must suffice: Administrators sometimes work with the world crashing around them. They must not be forced to rely on an elaborate software scaffold to get any work done.

Our *central choice* is unsurprisingly:

Internet-wide large-scale collaboration: We simply cannot see any other way for administrators at small sites to produce top-quality results. (**P-4**)

Other basic design choices follow from our collaboration imperative.

Separate mechanism and methods: Our 'it's OK to be different' goal (**G-6**) means that the core ARK machinery needs to provide a mechanism that doesn't constrain the methods used in administration. Sites ought to be able to pursue different policies, architectures, and/or methods, yet still be able to express their solutions in an ARK framework.

Reuse must be lavishly supported: Good ways to reuse administrative solutions (across organizational boundaries) are essential; otherwise, sharing to overcome administrator isolation (**P-4**) clashes with the need for site-specific solutions (**G-6**).

Not all-or-nothing: ARK must not be an all-or-nothing proposition. If the only way to 'get into' the Arusha Way is to start building a site over again, administrators will (rightly) walk on by.

Object orientation: We need some extra thinking machinery. For this, we steal a simple form of 'object-oriented' thinking from the software world. It is ubiquitous, universal in application, and is essentially a complexity-managing tool. (**P-1**)

Using XML as the main notation: We need some extra notational machinery. For this, we make simple use of XML [XML]. It has the merit of being a standard, and of perhaps being faintly familiar to administrators. The tidal wave of XML-related tools should make it easier to get presentation right as well. (**G-7**)

XML's extensibility means that different sites can use different tags to encode their unique information. And the way XML can wrap around other programming text means that the 'business end' of a solution can use a specialized tool (e.g., PIKT [Ost]) or remain in a familiar scripting language (**P-8**).

What we value most about XML is its *semi-structured* nature: the level of precision of a description can range from utterly precise (very structured) to exceptionally loose (unstructured) . . . For example, here is an install method that is precise:

```
<install><code>
cd /build/dir/foo
/usr/bin/make install
</code></install>
```

And here is a much looser 'equivalent':

```
<install><code lang="message">
You will need to be in the directory
where foo is built.  Typing
'make install' should do it.
</code></install>
```

XML lets us express all of our 'added value' (**G-2**) but without forcing us to do so rigorously. ARK can then support an evolution from informal solutions (probably text) to precise ones (probably code). (**P-7**)

All of that said, we are *not* that excited about XML.

Tribal Matters

The Arusha Project (ARK) is substantially a social enterprise, and that is not the subject of this paper. But we do need to mention 'teams' and comment upon the old 'but we cannot give away our secrets' chestnut.

Teams

The *team* is the basic social unit in ARK. All code is tied to some team. A *site team* produces and manages the code specific to a site. A *methods team* produces and manages code that is site-independent and which (presumably) promotes particular ways of doing administrative things; the team's members might share an office, or be scattered around the world. And finally, there is at least one *mechanism team*: the base team, also called 'ARK,' provides the ARK engine.

How teams go about their business (what they promote, how they distribute their bits) is entirely up to them. Teams may have profoundly different notions of administration. A healthy Arusha world would comprise many site teams, possibly just the one mechanism team, and perhaps four or five general (how-to-run-a-site) 'methods' teams. There might also be quite a few specialized teams that target a specific domain, e.g., how to run a particular flavor of website.

Collaboration vs. Competition

The ARK collaboration imperative may seem at odds with a goal of seeking competitive advantage through better computing infrastructure.

There *are* some aspects of administration you would not reveal to a competitor: your exact mix of tools, your /etc/hosts.allow file, anything related to your core competence, and so on.

But it is simply delusional to think that your standard-issue Apache configuration (say) somehow represents a rival-clobbering breakthrough. On the contrary, if the configuration was set up by a solo administrator without any peer review (**P-4**), the quicker you expose it so that a fellow ARKer can knock off the rough edges, the better.

Strangely, we have a hunch that Internet-wide administrative collaboration may work better than that within an organization. Inside a company, interactions (between, say, 'satellite' and 'corporate' IT groups) may be clouded by issues of funding, status, or political advantage. Meanwhile, the stranger in Croatia who critiques your Big Brother setup is probably a disinterested party.

Even if publicizing considerable administrative work meets with a collaborative deafening silence, there is still one *certain* beneficiary, and that is the provider: we all do better work if we know others will see it.

Swahili[12]

The single unavoidable piece of Arusha technology is the ARK *configuration language* and the *engine* that interprets it.

Things, Fields, Values (and More)

The ARK configuration language has an XML syntax and can describe any entity an administrator cares about. The following figure shows (contrived) examples of the unavoidable entities (or *things*, in ARK-speak) – teams, hosts, and packages.[13] A team:

```
<team name="glaslil">
<contacts><list>
  <item>partain@dcs.gla.ac.uk</item>
</list></contacts>
<admin-group>sliadmin</admin-group>
</team>
```

A host:

```
<host name="slicker">
<status>active</status>
<ip-address>130.209.242.51</ip-address>
<history><doc format="html"><![CDATA[
<ul>
<li>2001.09.04: Rebooted (matt).
<li>2001.03.10: Add 2nd disk drive.
</ul>
]]></doc></history>
</host>
```

A package:

```
<package name="textutils--2.0.11">
<status>revealed</status>
<hosts-supported><list>
  <item>sparc-solaris7</item>
</list></hosts-supported>
</package>
```

All instances of an ARK 'thing' (e.g., all hosts) are said to be of the same *type*.

Besides the unavoidable types (teams, hosts, packages), a site may choose to manage other things: users, support call-outs, disks, network ports, . . . the ARK engine is domain-agnostic. Some fictional examples of such things might be:

```
<user name="matt">
<status>active</status>
<cron-allow-fragment>
  <constraint>
    <host-spec>slinger</host-spec>
  </constraint>
  <string>matt</string>
</cron-allow-fragment>
</user>
```

```
<support-contract name="hp">
<terms>next day</terms>
<phone>+44 800 555 4321</phone>
<email>none</email>
<expires>2002.03.31</expires>
</support-contract>
```

Every thing has zero or more *fields*. For example, the <support-contract> example above has an <expires> field. Fields are *not* nested.

An individual field, whatever thing it is part of, has a structure drawn from a *fixed* set of *elements* (**G-3**). (That is worth saying again: *All* fields of *all* things have the *same* internal structure!) The most important field elements are:

A value: A *value* can be one of: a string, a list, a table (key-value pairs), a documentation fragment (various formats), or some code (Bourne shell, Python, or Perl). Preceding examples show at least one of each. A value can be nested in obvious ways, e.g., a list of tables of lists of lists of strings . . .

Parameters: Values, most notably *code*, can be parameterized; this is a key reuse weapon, and essential to "once and once only" source code (**G-5**). So, for example:

```
<do-it-now>
  <param name="ECHO">/bin/echo</param>
  <code>$ECHO "I'm doing it now"</code>
</do-it-now>
```

Parameters usually have defaults, as in this example.

Constraints: A field's *constraints* guard its value; if the constraints are not satisfied (or cannot be made so), then the value/parameter-settings/etc. do not apply. Consider:

[12]Swahili is the *lingua franca* (trade language) of eastern Africa. Though it is mother tongue of some people who live on the coast, most Arusha dwellers would speak another language at home and use Swahili in civic life.

[13]All XML examples are slightly simplified compared to Real Life.

```
<used-for>
  <constraint>
    <host-spec>freebsd</host-spec>
  </constraint>
  <string>Web serving</string>
</used-for>
<used-for>
  <constraint>
    <host-spec>rhlinux</host-spec>
  </constraint>
  <string>SETI@Home</string>
</used-for>
```

If we ask for the <used-for> field in the context of a freebsd host, we get one string; if an rhlinux host, we get another.

Constraints are normally intra-type; for example, one package method depends on another. But constraints can also be cross-type; for example, a host method could depend on a user's attribute or a vendor's method.

Prototypes

You should now have some notion of how you might ARKishly describe all of your hosts (for example). For each machine, you would prepare an XML file (<host> ... </host>), each of which might have many fields, e.g., <ip-address>, <serial-num>, <disk-config>, <os>, etc.

This task would be hugely repetitive, because many machines would have the same elements for the same fields. The solution is to create a *proto-host* (a specific form of *proto-thing*) that expresses the common knowledge; e.g., see Listing 1.

Now, all the hosts that have these properties (presumably all 'lab machines') can have this (proto-)host as a *prototype*; for example:

```
<host name="slibber">
<prototypes>
  <prototype team="ours"
            name="lab-machine">
</prototypes>
<ip-address>192.10.168.4</ip-address>
<tagline>
  <param name="what">workstation</param>
</tagline>
</host>
```

'Things' as Objects

The 'prototypes' idea is drawn from the world of *classless* object-oriented programming languages [Bor86]. By 'object,' we mean an opaque entity that presents an interface to the world through public *attributes* (data about it which you can query) and *methods* (code that makes it 'do something').

The idea of a prototype-based object is incredibly simple: you create a new object by first copying another, and then tweaking the new object to make it unique. The object, or objects, that you copy (from) are the *prototypes* for the new object.

This is exactly what we are doing with ARK things. In the example above, the host named slibber is created by first cloning the prototype host lab-machine, and then adding/overriding-with its own unique data, notably the field <ip-address>.

An ARK thing (with prototypes) fits our definition of 'object,' above. A field with a non-code value is an attribute, and a field with a code value is a method.

For example, we can query our host object slibber for its attribute ip-address (a string), or for disk-config (a table). Or we can invoke one of its methods, perhaps restart-sendmail (implemented with a shell script).

A thing can have zero or more prototypes. A proto-thing can itself have prototypes. Exactly how this works out is explained later.

Inheritance

In our example above, the host slibber *inherits* the value its disk-config field from the lab-machine proto-host. Inheritance is a fundamental property of objects which helps to control complexity: we push common/reusable attributes and methods into the base (prototype) objects, which make them widely available to their inheritees. (**P-1**, **G-5**)

Combining this inheritance with parameterized values gives easy reuse of administrative solutions.

In ARK, field fragments are the *only* inherited entities. Thanks to our classless prototypes mechanism, there is no complex type/class structure to worry about as well (**G-3**).

```
<host name="lab-machine" prototype="yes">
  <tagline>
    <param name="what" default="no" />
    <string>A @param:what@ in Lab 4</string>
  </tagline>
  <disk-config><table>
    <entry name="boot">  Quantum 9902</entry>
    <entry name="other"> IBM   4/4432</entry>
  </table></disk-config>
  <os>NetBSD 1.4.3</os>
  <restart-sendmail><code>
  kill -HUP `/bin/cat /var/run/sendmail.pid`
  </code></restart-sendmail>
</host>
```

Listing 1: A *proto-host* expressing common knowledge.

This style of 'value inheritance' is close to that in Couch's Babble [Cou00] and that recently explored by Anderson [And00b].

Textual and Semantic Entities

A quick review: what do we type (textual), and what do we think about (semantic)?

A 'thing' is a semantic entity to which we apply 'object' thinking. The total textual material that comprises that 'thing' comes from potentially-many files, one per thing or inherited proto-thing. In our example, two files: slibber.xml and lab-machine.xml.

Similarly, a 'field' is a semantic 'atom,' the smallest piece of a 'thing' that we can get a hold of. The total textual material that comprises a 'field' is one or more like-named *field fragments*, plus the prototype links that connect them. In our slibber example, its <ip-address> field comes from one field fragment. However, its tagline field comes from two fragments, the *partial* fragment (no value) in slibber.xml and the completing fragment in lab-machine.xml.

Crucial point: even the smallest entity in ARK land, a field, can be built up collaboratively by different teams scattered around the world!

Other Language Complexities?

We have glossed over some details of the configuration language, mostly unexciting details of what you can do with a <code> value. We are sometimes asked for further details about 'modules,' or include files, or DTDs, or other suspected language facilities; in short, 'there must be more to it' . . .

No, *that is all there is* (**G-3**). (For a complete rundown on field elements, see the language manual [ARK].)

Prototypes As Matching and Naming Mechanisms

A prototype name can be a *pattern* against which we match 'real' (non-prototype) things. When we had a <host-spec>freebsd</host-spec> constraint, it simply meant that any real host which has the freebsd proto-host as one of its prototypes will match.

Similarly, *naming* a prototype thing is equivalent to naming all of the real things that have that as a prototype.

Prototypes as Cross-Planet Inheritance

Notice that every <prototype> 'link' must specify a *team* ('.' is shorthand for the prevailing site-team).

This team can be (and ideally will be) a global ARK team. By using the prototype things it supplies, you take advantage of others' expertise that may benefit your site's requirements. You also maximize the pool of people who will be interested in (and critique) your own contributions.

We like the idea that people around the world work to improve our systems while we are asleep.

The Operational View

How do we write code to *use* the ARK objects (things), to access their attributes and invoke their methods? In Python, you write code that looks like this:

```
# create a host "object" for
# our 'slibber':
slibber = ark.host.ArkHostsMgr(). \
             lookup('slibber')

# access and print out its IP#:
print slibber.ip_address()

# run its restart-sendmail method:
slibber.restart_sendmail()
```

This is object-oriented programming at its simplest. What lies behind these rather lovely method calls is the ARK *engine*, which scrambles over <prototype> 'links' to create the illusion of the built-up-by-copying objects.

The prototype links in an ARK 'thing' comprise a directed acyclic graph; the nodes visited by doing a depth-first left-to-right traversal give a thing its *prototype path*.

Operationally, the ARK engine walks along a prototype path. This has the same semantics as actually copying proto-things, but is more efficient.

The executive summary version of the algorithm is: walk the prototype path looking for the field of interest, checking constraints and collecting parameters as you go, returning immediately when you find a field fragment with a value. (For tables, we keep going and 'merge' all of the entries found.)

The Command-Line View

We provide a command-line tool ark, with which we can 'fire' common methods for ARK objects. The syntax is: ark *type method [options] [thing1 [thing2 . . .]]*.

The examples given earlier should now make sense!

Big Game Spotted[14]

We have described enough of the ARK configuration language to show why it is effective 'glue' to hold together world-wide collaborative administrative effort, as well as the merits of our simple object model. We note connections to the problems (**P-***n*) and goals (**G-***n*) of the first section.

Collaborative Wins

Domain- and methods-agnostic: The ARK configuration language has virtually no system administration wired into it. Choosing to express solutions with it in no way compromises local ways of working.

Optional: Managing aspects of your site with the ARK machinery is optional. If you don't want <vendor>s as part of your ARK world, then don't.

Evolutionary: You can start small with ARK, and grow bigger (**P-7**). Also, you can bring non-ARK

[14]Arusha is awash in safari operators, as it is the 'jumping-off point' to all of northern Tanzania's game parks. Most tourists hope to see the Big Five: buffalo, elephant, leopard, lion, and rhino.

bits of your world into ARK play quite easily. Consider this variant on the method we saw earlier:

```
<restart-sendmail><code>
  /usr/local/sbin/restart-sendmail
  </code></restart-sendmail>
```

(We presume the referenced script is pre-ARK.)

Lingua franca, not mother tongue: Though we envisage a strange administrative world of 'objects,' 'methods,' etc., we expect this to materialize mostly by straightforward 'wrappers' around conventional solutions. The 'wrapping' is this funny ARK stuff, but the 'business end' is still your favorite scripting language (**P-8**), or Cfengine, Nessus, RPM, Swatch, or any other tool of your choice.

Simple pragmatics: ARK uses a dull form of XML, and requires little more than a text editor and a Python interpreter. Most fields in most XML files are a few lines long, particularly for site teams, which typically inherit most of their smarts from a global team. (**G-5**)

Simple mental model: The ARK prototype-based 'object' model is as simple as they come, but packs a heavy punch (**G-3**). It means we can think about *all* aspects of a site's administration in a *uniform* framework.

Simple data: Strings, lists, tables, bits of scripting or documents . . . not a lot to get your head around.

Rich reuse: The basic way that a field become reusable is by putting it into a prototype thing. As a field is pushed up a prototype tree, it applies to more and more inheriting 'objects.'

If we then parameterize some aspects of inheritable attributes/methods, their reusability is considerably enhanced. Our experience is that you do this slowly, as the need arises.

A small unit of collaboration, the 'field': Collaboration does not work if the parties have to agree on too much up front. The smallest 'atom' of ARK collaboration is a single field of a particular type of thing. Sites can do quite different things with ARK <host>s, but if they can agree on just a few fields (e.g., <ip-address>, <gateway>, and <dns-servers>), useful collaboration can follow.

ARK is easy to learn: Our experience is that a competent Unix administrator can learn everything they need to know about the ARK configuration language in half a day.

Object Wins

Our dominant mental metaphor, of 'objects' with attributes and methods, is a powerful (and well-known) way of thinking about systems. It attracts benefits of its own.

Universal: The ARK configuration language can describe *any* aspect of a system that you care about. (**P-6**, **G-2**)

Should you choose to record a preponderance of administrative 'added value' with the ARK language, you then have comprehensive 'source code' for your system (**G-4**), which is a key defense against administrators falling under buses (**P-3**).

If many sites record their solutions in the ARK language and make them available to others, then we have a ready source of complete, accurate, probably-automated examples that administrators can study (**P-9**).

Abstracted away from immediate file contents: Because ARK is about 'objects' rather than files and bytes, it tends to operate at a high level of abstraction. 'Site-at-a-time' operations (**G-1**) are the most obvious manifestation of this; for example configuration files that require a different format for each platform can be produced from a single object representation.

Complexity control: The fundamental strength of object thinking is managing complexity. We try to push complexity 'upward' into a prototype, so that many things can inherit from it (spreading the complexity cost over more things) (**P-1**).

If we push the complexity into a *global* team, we hope to find ourselves working with other clever people (in the relevant team) to keep the fancy stuff right. "With enough eyeballs, all bugs are shallow" (Linus's Law, according to Raymond [Ray00]). This is how you can beat the inherent mediocrity (**P-2**) in a small isolated administrator team (**P-4**).

Will They Bite?

So, will administrators working in our context bite and do the ARK thing? As we have said, we are sure the answer is more sociology than technology. One of the biggest obstacles today is IT management that optimizes for 'cost' and has never thought of optimizing for 'effective.'

Administrators will need to learn enough XML to talk the ARK talk; they probably know a little HTML already, so it is not a stretch.

Administrators really should know how to write Bourne shell scripts, ARK or not. Perl and Python are optional.

Administrators are slow to adopt *any* site-configuration tool, because they (rightly) know that it is a decision that will be hard to escape. We've tried to make ARK suitable even for glacially-paced incremental adoption.

Related Work

There is a flotilla of tools that offer comprehensive management of a single host: HP's SAM, IBM's SMIT, Linuxconf, and many others. These are all nearly useless in our context.

The literature (and world) is crawling with 'site configuration' systems; Evard's 1997 paper [Eva97] is a particularly useful review. On the specific matter of configuration *languages* (notations), Paul Anderson's survey [And00b] is a good overview of the territory. He draws on the venerable Edinburgh work on LCFG [And00a], as well as Cern's SUE [SUE], Cfengine [Bur] and others. If you venture over to the world of software deployment, there are *lots* of related things; a good starting point is the work by Hall, et al. [Hal98]. Moving not much further afield and you reach the bewildering land of 'software configuration management' . . . [App]

Our ARK work tends to differ from other 'site configuration' tools in that they make no upfront provision for collaboration across organizational boundaries. Couch's DISTR system [Cou97] is an exception, with collaborative concerns very similar to ours, but limited to file distribution. Another system in a comparable vein is 'PowerAdmin' [Pow], a customizable service that diverse groups within the University of Michigan can 'buy.'

There are many systems where a difference in scale is apparent. In the high-performance computing arena, there is much effort (and money being spent) on *computational grids*, e.g., the Globus Project [Glo], aspects of which are squarely in our domain, notably the Metacomputing Directory Service (MDS). All-inclusive commercial administrative tools, e.g., CA's Unicenter TNG [CA], HP's OpenView [HP] and Tivoli's tools [Tiv] seek to cover the wide ground that ARK does; again, however, such solutions tend to be well beyond the means of our target audience (and often platform-specific).

In his analysis, Evard [Eva97] suggests that the "systems administration community needs stronger abstraction models." We believe the ARK object model contributes here. As mentioned before, both Anderson [And00b] and Couch [Cou00] have dabbled with a similar form of 'value inheritance.' We further note that the FLASH project in Brazil also picks up on object-oriented ideas, in a more complex way [daS98].

There are a few other systems that try to capture constraints among configuration artifacts, as we do. Ganymede [Abb98] is one example: it is a directory service into which 'local smarts' can be programmed. The work by Couch and Gilfix with Prolog [Cou99] is another powerful (and scary) way to tackle such constraints. Again, if you move slightly further afield, you find many comparable systems; one example is the CML2 Linux-kernel configurator [Ray01].

Roads From the Arusha Clock Tower[15]

The Arusha Project (ARK) is not about producing a tool; rather, it hopes to be at the center of a maelstrom of collaborative system administration. The core developers' activity is driven by 'scratching the itches' at their real sites, which may or may not have value to others. We would hope that most Arusha activity will evolve to happen well beyond our purview.

One aspect of ARK that we expect to occupy us for a while is the presentation (or documentation) of a system (**G-7**). We think of it as bringing the literate-programming impulse [Knu84] to the system source code (as represented by the ARK XML files). (We have a first-cut implementation, using the Webware application server [Est01], also written in Python.)

The ARK configuration language is *domain-agnostic*, and so the question arises: in what other fields might it reasonably be applied? For example, the 'chipmake' tool, for describing how to put together a semi-custom chip, was scarily close to ARK in the issues that it had to address [Hol00].

Summary

The Arusha Project (ARK) has a profoundly ambitious goal of many thousands of Unix sites around the world being managed in a collaborative way.

We began this paper with a from-first-principles analysis of our target context, reaching a set of goals we would have for any system-administrative framework. The ARK configuration language provides a basis for meeting all of those goals. It is an XML-notated *lingua franca* with which system administrators can describe the value they have added to a collection of raw vendor-supplied computing equipment. Their descriptions are in terms of *objects* ('things'), with parameterized *attributes* ('fields') and *methods* (fields with code values). We have a *universal* and *as-precise-as-you-like* language for describing administrative activities (because of the semi-structured nature of XML).

Our object 'things' are built up out of *prototype* objects, possibly supplied by other teams, potentially anywhere in the world. Administrators *collaborate* on shared solutions insofar as they use (and work to maintain) common prototype objects.

The ARK 'object' view of administration is lightweight, builds on standard Unix tools, and allows extremely varied uses, from do-just-one-specific-task to run-the-whole-site.

The following ideas are unique to the Arusha Project:

- Describing *all* administrative 'added value' in a *single* notation (the ARK configuration language
- Viewing these descriptions as *objects* which link together *across* organizational boundaries;
- Advocating *world-wide collaboration* as the basic way forward for Unix system administration.

[15]The Arusha town center features a clock tower, next to which is a signpost giving distances to other locations. What with Arusha's place along the Cape-to-Cairo route, the places listed tend to the remote: London, Moscow, Cape Town, etc.

Acknowledgments

The Arusha Project (ARK) is an independent open-source project that has been based in the Computing Science Department at Glasgow University. We have received financial support for LISA-related expenses from the Department (including some under SHEFC RDG Project 85, ''Design Cluster for System Level Integration''), and from Verilab (http://www.verilab.com). We are very grateful to all concerned.

ARK work has origins in the glamake tool (1997), which automatically built open-source software for multiple platforms. The earliest (undistributed) ARK code was written in Haskell [Has]. ARK was set up as a SourceForge project in January, 2000. (Hooray for SourceForge!)

A small army made this paper hugely better than what we started with, including our LISA reviewers and shepherds, and also David Partain, Jonathan Hogg, Rolf Neugebauer, and Tommy Kelly. Thanks, folks.

Status and Availability

The central Arusha Project website is http://ark.sf.net/ . It includes instructions for getting any and all ARK bits.

We recommend that prospective ARK users join one or more of the ARK mailing lists. Also, the website recommends some get-your-feet-wet activities to try before you switch to an ARK lifestyle.

At time of writing, there are a few real production Unix sites fully managed in the Arusha Way, mostly in the chip-design ('technical computing') arena.

Author Information

Matt Holgate is a Research Assistant on the IDEAS project, which is funded by SHEFC RDG 130. He is based in the Computing Science Department at Glasgow University, Scotland. He is also a part-time system administrator for Verilab, a Scottish hardware design and verification company. He graduated with an honours degree in Computer Science from Trinity Hall, University of Cambridge in 1998. Contact: matt@dcs.gla.ac.uk.

Will Partain is a graduate of Arusha School, Rift Valley Academy, Rice University (BSEE), and University of North Carolina at Chapel Hill (Ph.D., computer science). He was one of the original development team for the Glasgow Haskell Compiler, at the Computing Science Department, Glasgow University. Contact: partain@dcs.gla.ac.uk.

References

[Abb98] Abbey, Jonathan and Michael Mulvaney, ''Ganymede: An Extensible and Customizable Directory Management Framework,'' *LISA 1998*, Boston.

[And00a] Anderson, Paul, and Alastair Scobie, ''Large Scale Linux Configuration with LCFG,'' http://www.dcs.ed.ac.uk/home/paul/publications/ALS2000/ .

[And00b] Anderson, Paul, ''A Declarative Approach to the Specification of Large-Scale System Configurations,'' version 1.9, Discussion Document, 2001, http://www.dcs.ed.ac.uk/home/paul/publications/conflang/ .

[App] Appleton, Brad, ''The ACME Project: Assembling Configuration Management Environments (for Software Development),'' http://www.enteract.com/~bradapp/acme/ .

[ARK] ''The ARK configuration language manual,'' http://ark.sf.net/ark-conflang.html .

[Bec99] Beck, Kent, *Extreme Programming Explained: Embracing Change*, Addison-Wesley, 1999,

[Bor86] Borning, A. H. ''Classes versus Prototypes in Object-Oriented Languages,'' *Proceedings of the ACM/IEEE Fall Joint Computer Conference*, pp. 36-40, 1986.

[Bur] Burgess, Mark, *Cfengine*, tool, http://www.iu.hio.no/cfengine/.

[CA] Computer Associates, *Unicenter TNG*, tool, http://www.cai.com/unicenter/ .

[Cou97] Couch, Alva L., ''Chaos Out of Order: A Simple, Scalable File Distribution Facility for 'Intentionally Heterogeneous' Networks,'' *LISA 1997*, San Diego.

[Cou99] Couch, Alva and Michael Gilfix, ''It's Elementary, Dear Watson: Applying Logic Programming To Convergent System Management Processes,'' *LISA 1999*, Seattle.

[Cou00] Couch, Alva L., ''An Expectant Chat about Script Maturity,'' *LISA 2000*, New Orleans.

[daS98] da Silva, Fabio Q. B., Juliana Silva da Cunha, Danielle M. Franklin, Luciana S. Varejão and Rosalie Belian, ''An NFS Configuration Management System and its Underlying Object-Oriented Model,'' *LISA 1998*, Boston.

[DeM87] DeMarco, Tom and Timothy Lister, *Peopleware: Productive Projects and Teams*, Dorset House Publishing Co., 1987.

[Est01] Esterbrook, Chuck, ''Introduction to Webware for Python, 9th International Python Conference,'' Long Beach, California, March, 2001, http://webware.sf.net/Papers/IntroToWebware.html .

[Eva97] Evard, Rémy, ''An Analysis of UNIX System Configuration,'' *LISA 1997,* San Diego.

[Glo] ''The Globus Project,'' http://www.globus.org .

[Hal98] Hall, R. S., D. Heimbigner, and A. L. Wolf, ''Evaluating Software Deployment Languages and Schema,'' *Proceedings of the International Conference on Software Maintenance*, November, 1998; See http://www.cs.colorado.edu/serl/cm/Papers.html .

[Has] ''A Short Introduction to Haskell,'' http://www.haskell.org/aboutHaskell.html .

[Hol00] Holgate, M. and J. Hogg, "Chipmake: An XML-based Distributed Chip Build Tool, 1st ECOOP Workshop on XML and Object Technology," *Sophia Antipolis*, France, June 12, 2000.

[HP] Hewlett-Packard, *OpenView,* tool, http://www.openview.hp.com/ .

[Knu84] Knuth, D. E., "Literate Programming," *Computer Journal,* Vol. 27, No. 2, pp. 97-111, 1984.

[Mil78] Milner, R., "A Theory of Type Polymorphism in Programming," *Journal of Computer and System Sciences*, Vol. 17, pp. 348-375, 1978.

[Ost] Osterlund, Robert, *PIKT (Problem Informant/Killer Tool)*, tool, http://pikt.uchicago.edu/pikt/ .

[Pow] "The ITD PowerAdmin service," http://www.umich.edu/˜gpcc/poweradmin/ .

[Ray00] Raymond, Eric, "The Cathedral and the Bazaar," http://tuxedo.org/˜esr/writings/cathedral-bazaar/cathedral-bazaar/, 2000.

[Ray01] Raymond, Eric, "The CML2 Language: Constraint-based Configuration for the Linux Kernel and Elsewhere," http://tuxedo.org/˜esr/cml2/cml2-paper.html, 2001.

[SUE] "SUE (Standard Unix Environment)," tool, http://wwwinfo.cern.ch/pdp/ .

[Tiv] Tivoli enterprise management tools, http://www.tivoli.com/ .

[Tra98] Traugott, Steve and Joel Huddleston, "Bootstrapping an Infrastructure," originally published in the *Proceedings of the Twelfth USENIX Systems Administration (LISA) Conference*, Boston, Massachusetts, 1998, http://www.infrastructures.org/papers/bootstrap/bootstrap.html .

[XML] "Extensible Markup Language (XML)," http://www.w3.org/XML/ .