Automating Infrastructure Composition for Internet Services

Todd Poynor – Hewlett Packard Laboratories

ABSTRACT

This paper describes a framework for automatically configuring relationships that tie together the software and hardware infrastructure components for an Internet-based service, which we also call "composing" the service infrastructure. Management and infrastructure software employ the framework to specify, discover, and react to changes in the infrastructure components participating in the service, automatically configuring cross-component relationships based on this information. This plays a key role in increasing the flexibility of a highly dynamic service infrastructure by reducing manual configuration required to redeploy resources. The framework also facilitates enterprises distributed across multiple autonomous administrative domains by automating chores required to tie resources distributed among the domains together into a cohesive service. We advocate extending existing service discovery protocols to distribute the information needed by the framework and suggest this as an area for future standardization.

Introduction

The technology described in this paper arises out of our recent research into automation of deployment and configuration of Internet applications and enterprise infrastructure. Our chief motivation is to assist in manageability and flexibility of the data center. Especially for the more dynamic enterprise infrastructures envisioned for the near future [13, 14, 21], the configuration of the data center hardware and software may be in nearly constant flux to support changing workloads, changing customer sets, and changing hardware resources. Our goal is to redeploy available resources quickly and easily according to demand instead of massively over-provisioning dedicated resources, as has often been the usual practice. Today we find ourselves in the midst of a global slowdown in IT spending, and we can predict that a good many service providers will be interested in "doing more with less."

We are also interested in increasing reliability of management and consequent availability of the enterprise. Today, administrators must often manually perform a number of individual hardware and software configuration steps to effect a consistent change in data center configuration, which is a known source of errors and downtime. For example, a recent report [9] claims 40 percent of Web site downtime is due to "operations – e.g., not performing a required task or performing one incorrectly." Our goal is to pursue the extent to which the data center may instead be automatically reconfigured into the desired consistent state, in keeping with the Internet architectural principle of avoiding manually specified parameters [15].

Our work also targets management of a distributed enterprise hosted in distributed data centers, especially data centers operated under separate administrative domains by distinct service providers. For predictions of future trading of compute power as a "liquid commodity" [8, 10, 19, 22] to be realized, automated mechanisms for discovering and configuring the distributed components together into a cohesive enterprise are needed.

Among our first steps to address these topics is to design a framework for discovery and configuration of relationships among the infrastructure components, such as applications and networking functions, participating in an Internet-based service. Because this framework encompasses the problem of service advertisement and discovery, we suggest that this framework should leverage existing service discovery protocols.

The rest of the document is organized as follows. The next section describes our general approach. The following section discusses leveraging existing protocols for infrastructure discovery and describes a design leveraging the Service Location Protocol [1]. Subsequent sections describe our prototype, related work, and conclusions.

Automated Infrastructure Composition

This section describes our infrastructure composition framework in general, leaving aside for the moment details of any particular technology on which the framework may be based. We first define the concepts modeled by our framework and then describe the framework itself.

Describing an Infrastructure

The model chosen is of an enterprise described at a high level of abstraction by the following:

Resources, generally hardware, such as general-purpose server machines and load balancing appliances. A resource is identified by its network name (TCP/IP hostname).

Services provided by resources. We use the term *infrastructure services* to distinguish these from the higher-level e-services that are built from these components. An infrastructure service may be identified both by a *service type* identifier that describes the standard service protocol and by a name that identifies the particular implementation, used to discover proprietary or version-specific features. For example, the server named server127.xSP.com (a resource) runs an HTTP server (an infrastructure service type) that is Apache 1.3.19 (a specific implementation). Other service *attributes* that aid in configuring communication with the service, such as to identify a non-default TCP port, may be specified along with the service identifiers.

Contexts in which the various infrastructure services act in concert to provide higher-level Internet services; services are tied together through acting in a common context. A context may be identified using an identifier, such as the name of a particular customer in a shared data center, that must match for disparate infrastructure services to participate in the same context. A context may instead be identified by a URL that points to more detailed configuration information about the context using such formats as XML. For instance, the HTTP server on server127.xSP.com serves Web site tom.com (a context), and participates in the set of infrastructure services, also including load balancers and a Web cache array, that cooperate to provide the tom.com e-service.

The context identifiers allow infrastructure services to select the proper remote infrastructure services to tie together into a higher-level service. Through being configured into common contexts, infrastructure services can discover the other participants in their contexts (and ignore those of other contexts). The various resources of a data center may act in different contexts, such as to be partitioned among different applications or to be employed on behalf of different customers, and resources may act in multiple contexts at a time, such as to be shared among multiple customers.

An extension of this model is that of a ''virtual enterprise'' comprised of multiple data centers, each of which may be operated by distinct service providers, as may occur if in the future compute power is traded as a commodity [8, 10, 19, 22]. The virtual enterprise need not manage, nor even be aware of, every resource and infrastructure service in every participating data center, but only the subset of these with cross-data-center interactions. (It is assumed each service provider manages the local infrastructure according to the specifications of the virtual enterprise customer, in order to scale to very large enterprises.) We therefore use one ''global'' context for cross-data-center relationships managed by the virtual enterprise and one ''local'' context for each participating data center that manages local resources and services for the virtual enterprise according to local policies. Each infrastructure service with cross-data-center interactions acts in both the global context for the virtual enterprise and the appropriate local context for the {virtual enterprise, data center} pair. Each infrastructure service with purely local interactions acts solely in the local context. For example, the tom.com virtual enterprise owns a Web switch that is programmed to direct requests to the data centers that are currently contracted to host the tom.com e-service. The Web switch and the external contact points of the various data centers (Web servers or load balancers or Web caches and so forth) are configured in the global context for tom.com. A participating data center's infrastructure that supports tom.com (such as back-end Web servers hidden behind a Web cache) is configured in the local context for tom.com within that data center.

An Infrastructure Composition Framework

This paper describes a protocol and framework by which the deployment of resources, services, and contexts is specified by management tools and discovered by the affected components. There are two sets of deployment information to manage: the persistent instructions on how services are to be deployed, as specified by an administrator or system management software, and the (possible subset of these) deployments currently running. We call the former ''designated deployments'' and the latter ''current deployments.''

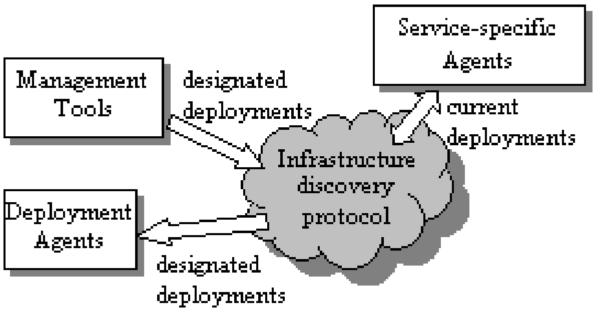

Figure 1: The infrastructure composition framework.

Management tools use the protocol to initiate changes in designated deployments. System management tools can provide high-level interfaces to model and modify the administrator's (or automatic management software's) designated deployments. The infrastructure composition framework automates the process by which the enterprise is configured to match those intentions. The eventual goal is that the administrator may execute a high-level command that reads something like ''add a new Web server to the tom.com context''; the management tools and the composition framework then execute a number of configuration steps required to make that happen. Each resource runs one deployment agent that uses the designated deployment information to discover that resource's ''personality,'' that is, what infrastructure services it is to provide and in which contexts. The deployment agent initially queries this designated deployment information at boot time and then subscribes to notifications of updates. The deployment agent contacts the appropriate service- specific agents on that resource to start, stop, and reconfigure contexts for the services as instructed by changes in designated deployments.

Each service-specific agent thus executed then takes the actions needed to start, stop, or reconfigure contexts for the associated service. These changes in current deployments are then registered using the protocol, such that other service-specific agents can configure or deconfigure interactions with the redeployed service.

A newly started service-specific agent uses the protocol to discover other current service deployments and configures any interactions required based on that information. The agent also subscribes to notifications of changes in current deployments for the relevant service types, such that interactions can be reconfigured on an ongoing basis.

Figure 2 depicts a general-purpose server that runs two infrastructure services that participate in the protocol.

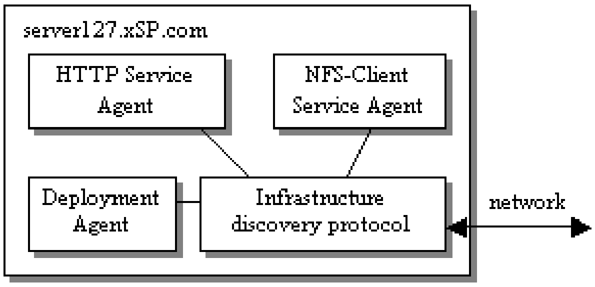

Figure 2: A server resource running two services.

The deployment agent discovers that it is to start the two services at boot time using the protocol and starts the services; the two service agents advertise their presence to other services and also discover and configure interactions with remote services using the protocol.

Figure 3 depicts two service-specific agents, one an HTTP load balancer and the other a firewall, reacting to notification of a new HTTP server deployed on resource server127 in context "tom.com" (in which context all participate).

The HTTP load balancer was originally told only to operate in the "tom.com" context, whereupon it discovered the appropriate set of target Web servers to balance among. Any change in the tom.com Web server membership is communicated to the load balancer agent so that it may update its set of targets. If an array of load balancers is employed then each instance may discover the other instances using our protocol, affecting the algorithms used to divvy up content; this is an example of discovering clustered instances of a specific product and configuring proprietary interactions.

A {resource, service type, context} 3-tuple defines the granularity at which deployments may be specified and discovered using the protocol, together with more detailed information made available through attributes associated with the tuple. The level of abstraction represented in the protocol is quite high-level, identifying the infrastructure services and resources participating in an Internet service together with basic information on how to establish communications. Further service-specific configuration information and actions may be addressed by more specific protocols or as extensions to this protocol through the service attributes or context information.

Further Details on the Framework

The deployment and service-specific agents for an appliance that do not support the infrastructure composition protocol natively may instead run on a nearby general-purpose server. The service-specific agent converts between the protocol and the proprietary management interface for the device. For specific-purpose appliances, the services to provide may be implicitly defined by the type of device and therefore the device need not query the designated set of services to execute. If so, at boot time a query is still generated for the resource's designated deployment, but only the information on designated context identifiers, not the service types, is used from the query result.

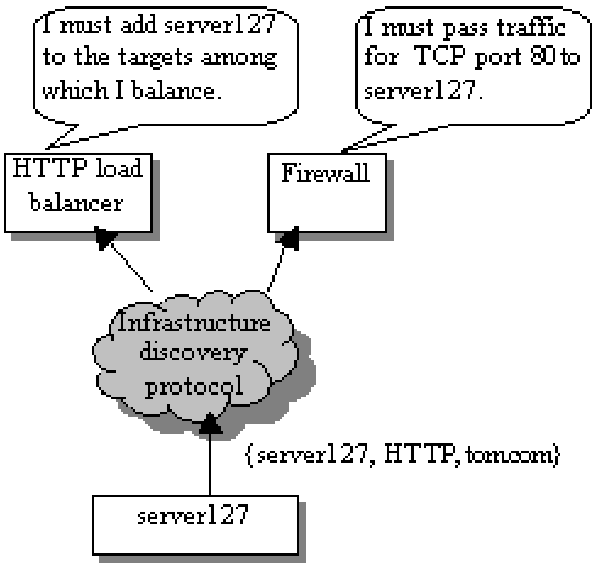

Figure 3: Two services configuring new relationships.

For general-purpose servers, the deployment agent and the service-specific agents may extend the existing means of starting and stopping services at system startup and shutdown time, such as the UNIX "init.d" scripts or the Windows "Services" applet. These are manually configured methods of specifying whether a certain service is to be started at boot time (or during normal system operation if manually executed). As extended by our framework, services may instead be started or stopped at any time based on remote instructions. The existing service scripts or applets may also add to their existing actions (which include "startup" and "shutdown") the action of reconfiguring due to changes in the rest of the enterprise. We have thus far avoided any mention of specific communication protocols or encodings for the infrastructure composition protocol. This is because we wish to leverage existing technology for this, for which there are a number of choices, as discussed in the following section. The formats for specifying the names of resources, services, and context identifiers can vary depending on the underlying technology, such as strings within a URL or XML-based representations. Likely network transport protocols include UDP datagrams for local traffic and TCP for cross-data-center or other longer-haul traffic.

Leveraging Existing Discovery Protocols

Service discovery protocols serve the purpose of advertising information about available services, their attributes, and their access methods to prospective clients. Although our purposes are geared more toward enterprise management and automated discovery of relationships between component services, we share the concerns of representing, finding, and accessing services using standard names and attributes. Service discovery protocols are currently a fertile area of innovation and competition in the industry; the field does not need yet another protocol that encompasses this topic. Accordingly, our designs incorporate existing discovery protocols and APIs for the new purpose of infrastructure discovery, also resisting the temptation to invent a new API that is independent of the underlying discovery protocol.

Several existing protocols primarily target ad-hoc, non-centrally-managed networks, such as access to currently nearby services by mobile clients and networks of consumer electronics devices or office equipment. Among these are the Jini Lookup Service [3], the Simple Service Discovery Protocol (SSDP) [2], and Salutation [5]. Protocols oriented toward discovery of services in centrally managed networks include the DNS service location resource record [7] and the Service Location Protocol (SLP) [1]. Service advertisement and discovery protocols are also a part of the business-to-business frameworks E-Speak [4] and Universal Description, Discovery, and Integration (UDDI) [6]; many of our concerns regarding dynamic reconfiguration of intra-enterprise relationships can apply in a cross-enterprise context as well.

Our focus on enterprise management influences our designs in certain new directions. Services form relationships based on management policy (as expressed in designated deployments and context identifiers), rather than automatically choosing remote service peers from an available pool based on service-specific criteria (such as a print server that advertises the proper printer capabilities) or user selection from a menu of choices. Resources are explicitly identified in the protocol, rather than being implicitly defined in service location identifiers, and are represented as entities that may perform multiple services. Services are tied together by common context identifiers. Both current and designated service deployments are represented, such that the protocol may be used to determine the "personality" of the local resource. We also strive for stronger consistency guarantees than certain protocols that can deliver stale information and that do not enforce consistency across replicated repositories, since we should represent a consistent view of the enterprise for agreement among the various interacting components.

Leveraging the Service Location Protocol

This section presents an example design that leverages the Service Location Protocol (SLP) version 2 [1], a service discovery protocol on the IETF standards track, for infrastructure service discovery. SLP defines URL encodings for service deployments, called *Service URLs*. For example, an LPR protocol network printer named "queuename" on system "printserver" might be described by Service URL:

```
service:printer:lpr://printserver/queuename;\
        (media-size=na-letter)
```

This information is usually advertised and queried using UDP datagrams to *Directory Agents (DAs)*, which function as caches for service advertisements[1] Queries use LDAPv3 search predicates [25]. SLP messages may optionally be authenticated using digital signatures such as DSA [26].

Local Infrastructure Discovery

For infrastructure discovery, we keep the information on "designated deployments" separate from the information on "current deployments" using a new namespace (technically a "naming authority" string) for designated deployments named "conf". Deployment agents use the "conf" namespace to discover the services to be provided by the associated resource, then register deployed services and discover remotely deployed services using the usual namespaces.

Designated and current deployments are represented as Service URLs. Information on the associated resource and context for a service deployment is represented using attributes of the same names. Example Service URLs for designated and current deployments, respectively, of a service named tomapp on resource server127 as part of the tom.com context are:

```
service:tomapp.conf://futureuse;\
      (resource=server127.xSP.com),\
      (contexts=tom.com)
```

```
service:tomapp://server127.xSP.com;\
      (resource=server127.xSP.com),\
      (contexts=tom.com)
```

[1] We disregard the SLP small network model without Directory Agents for the enterprise-class purposes of this paper.

The "futureuse" identifier is a placeholder for the URL of further configuration information, the format of which we plan to specify in a future design phase. Resources are explicitly identified as an actual deployment attribute, rather than relying on the *addr-spec* portion of the Service URL, to facilitate searches on resource keys. Contexts are represented as attributes in part so that common mechanisms for queries on resource and context identifiers may be employed, although mapping contexts to the SLP feature of administrative "scopes" may suffice instead.

Both designated and current deployments are registered and unregistered using the usual SLP mechanisms, specifying Service URLs for the appropriate namespaces and that include the "resource" and "context" attributes.

To query for deployments, we use the SLP query by service-type name and attributes, extended to also allow queries for service-types that include the wildcard "*". For example, to discover the designated service and context deployments for a resource, a query is sent with service-type "*.conf" and a resource qualifier but no context qualifier. Queries for wildcard service-types also support arbitrary service types, the names of which are not well known but that register well known attributes, permitting configuration dependencies to be discovered. For example, a firewall can query for all services within the contexts in which the firewall operates, discovering the appropriate protocols and ports for non-well-known service types through the registered attributes.

Means by which software may be notified of changes in SLP service registrations are proposed in an IETF draft extension [12]. As above, we extend the design to allow subscribing to notifications of updates to arbitrary services, without knowing the appropriate service-type names in advance. Changes in designated deployments need be communicated to only the affected resources, which may need to start or stop services accordingly. We therefore extend the design such that subscriptions can be qualified by predicates such as "(resource=myhostname)". Once the services are actually started or stopped, the affected components are notified of the change in actual deployment as usual.

Distributed Infrastructure Discovery

SLP is designed to function within a single cooperatively managed network. We wish to adapt the protocol to represent the required service relationships across a "virtual enterprise" of distributed data centers. The two-level hierarchy of information to be managed, one level for the local data center (for which SLP is already designed) and one for the infrastructure services visible across the virtual enterprise, is expected to scale appropriately using SLP.

Preliminary designs call for at least one Virtual Enterprise Directory Agent (VEDA) that is a repository for information on all SLP Directory Agents (DAs) that participate at the global level. We call DAs that operate in the global context "global DAs" to distinguish these from DAs that operate solely in the local contexts of a data center, if the data center is arranged such that there are such "local" DAs. A VEDA is simply a global DA whose presence is guaranteed to exist for a sufficient period of time to "bootstrap" a new data center into the virtual enterprise – the distinction is necessary because we allow for the set of participating global DAs to be highly dynamic due to data centers joining and leaving the virtual enterprise as conditions warrant.

Initial discovery between a VEDA and the global DAs for a new outsourced data center joining the virtual enterprise is expected to occur out-of-band, perhaps as part of the process by which a contract is formed between the virtual enterprise and the data center. The new DAs and existing global DAs subsequently discover each other and exchange information on the relevant service deployments using SLP DA interaction protocols proposed in an IETF draft extension [16], further extended to encrypt communications to avoid giving too many clues to the "black hats." All cross-data-center SLP traffic occurs between global DAs; infrastructure services operating in a global context register information with a global DA that relays the information to remote global DAs. Figure 4 depicts Data Center Provider 2 joining a virtual enterprise. The new data center's global Directory Agent announces its presence to the Virtual Enterprise Directory Agent, which registers the new global DA and responds with a list of global DAs for the virtual enterprise. Only one peer DA is shown, hosted by the same provider as the VEDA. The new and existing DAs then exchange information on service deployments acting in the global context at the respective data centers and configure any cross-data- center interactions accordingly.

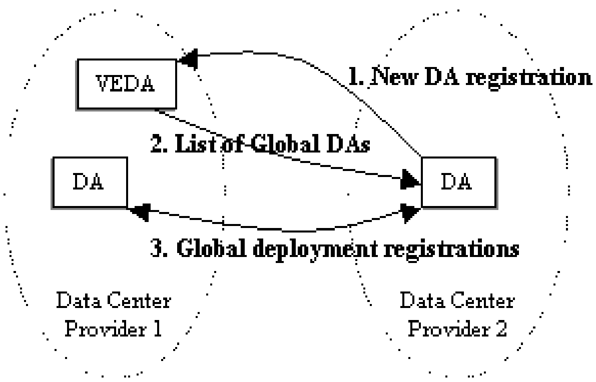

Figure 4: New data center joins the virtual enterprise.

Prototype Results

We prototyped a single-data-center subset of the framework, without support for cross-data-center interactions. The prototype infrastructure composition protocol is based on OpenSLP 0.9.1 [23], an open source implementation of SLPv2, modified to support wildcard service queries as described previously. We also added a temporary measure for proposed deployment update notification extensions [11] not available at this time. The temporary mechanism simply executes the local deployment agent upon any service registration or deregistration received by a Directory Agent. Each resource therefore runs a DA (this is not normally the case in an SLP network); all updates are broadcast to each DA through DA interaction protocols built into OpenSLP. The deployment agent in turn informs each local service- specific agent of each update, which, of course, does not scale to realistically sized networks.

The resource and service components participating in the protocol are:

- An *ipfw/ipchains* firewall (and Network Address Translation in the future) on an intelligent network interface card in an HP J5000 PA-RISC server running HP-UX 11.0. All traffic to the following three systems is routed through this firewall.
- An NFS file server and boot server for the following two systems on an IA32 server running Linux 2.4.3.
- Two Apache 1.3.19 Web servers on two IA32 servers running Red Hat Linux 6.2. These systems boot over the network using Etherboot 4.6.3 and mount their file systems over NFS from the above boot/file server.

The above four server systems each run an SLP Directory Agent and have installed the infrastructure composition framework deployment agent and the appropriate service-specific agents. The deployment agent and the service-specific agents are implemented as bash/POSIX shell scripts to match the language of the existing service startup/shutdown scripts shipped with the operating systems (this is a somewhat clumsy language for this purpose; use of languages specifically targeted at configuration such as GNU cfengine [28] could prove beneficial). SLP registrations and queries are accomplished from a shell script by running a utility named slptool from the OpenSLP package that allows SLP protocol actions to be expressed as run string parameters. The service-specific agents leverage the existing system startup/shutdown scripts (such as */etc/rc.d/init.d/nfs* on Linux) to start and stop the associated services.

The server systems execute the local deployment agent at boot time to configure and start any services that participate in our framework. The deployment agent then queries the SLP ".conf" namespace to determine the designated services to start on the local resource, executing the service-specific script for each such service. The deployment agent is also called upon each change in designated or current deployments among the four systems, passing this information to the service-specific scripts.

In the absence of any fancier tools that abstract away the details of SLP syntax, we use the slptool utility to configure designated service deployments for any of the four systems. For example, an Apache Web server is started on system App4139 and configured for customer paladin.com using the following command on any of the systems:

```
slptool register http-server.conf:\
 apache//App4139 ''(resource=App4139),\
 (contexts=paladin.com)''
```

Upon receipt of the above registration, the SLP Directory Agent on App4139 runs our deployment agent, passing it the above information. The deployment agent in turn executes the http- server/apache service-specific script. That script configures an Apache "virtual host" for paladin.com (a mapping of the paladin.com domain name to a set of Web content and other Web site configuration information; there may be many virtual hosts served by the same Apache server) by modifying the httpd.conf file. The service-specific script also starts or restarts Apache using */usr/local/apache/bin/apachectl* and registers the new current Apache deployment using an slptool command similar to the above but without the ".conf" namespace.

The firewall service-specific agent is notified of the new http-server deployment on App4139, and opens access to TCP port 80 on App4139 via an *ipchains* command if not already. Conversely, if the http-server on that system is deregistered, access to the port is blocked.

The Web content for each customer is mounted over NFS from the file server. The NFS clients on the two Web server systems and the NFS server on the file server are services that participate in our framework. An NFS client operating in the paladin.com context locates the NFS server for paladin.com using an SLP query for the filesys-server:nfs service and mounts from the server hostname and file path as returned in the query results. The NFS server discovers the configured NFS clients through a query for filesys-client.conf:nfs registrations and allows access to the appropriate file system for the resources thus identified. Both NFS clients and servers are notified of updates in deployments and respond accordingly, and so a client will unmount a file system from a deregistered server and re-mount from a newly registered server; a server will drop access for a deregistered client and add access for a new client.

To move the paladin.com Web site from one Web server to the other, the old filesys-client:nfs and http-server:apache entries are deregistered (or modified to drop the paladin.com context) and new entries are registered for the new resource (or existing entries are modified to add the paladin.com context). The reconfigurations that ensue are as follows:

- The NFS server revokes access to paladin.com files for the old Web server system and grants access to the new system.

- The paladin.com files are unmounted from the old Web server system and are mounted on the new system.
- The Apache virtual host configuration for paladin.com on the old Web server is removed and a new virtual host is created on the new server.
- If there are no more contexts for the old Apache to serve then the Apache server is shut down and the firewall blocks port 80. If the new Apache server is not yet running it is started and the firewall opens port 80.

The above actions can be triggered by a command of form ''move web site paladin.com from server1 to server2.'' The full set of actions has been observed to occur in less than two seconds. Add such components as network address translation, HTTP load balancing, Web caches, application servers and database servers, VLANs, and so forth to the picture and we can assert that the infrastructure composition framework results in a real benefit even for infrastructures that change only infrequently.

The prototype uses diskless servers such that in the future the protocol may also trigger software deployment activities. A notification to act in a new context causes the server to assemble and boot software needed to serve the associated context (such as the appropriate operating system and Web server software), in addition to mounting the appropriate files over NFS as currently prototyped. This is to be accomplished through a software deployment service that associates the context identifiers with NFS mount points for the OS, applications, and data, together with other configuration information required to deploy a new instance of a software stack previously configured for that context.

Another future direction is to represent dependencies between service reconfigurations on the local resource, such as to ensure each service using the file systems for a context have completed reconfiguration out of that context before the file systems are unmounted by the file system client. We are also looking at publishing additional service/context-specific configuration information over HTTP from URLs associated with the context identifiers. For example, when the service-specific agent for the Apache Web server is instructed to act in a new context it looks up such information as the administrator's e-mail address and authorization rules at URL http://context/config/http-server/apache.xml .

Related Work

We have previously discussed comparisons between our enterprise infrastructure composition protocol and a number of industry service discovery protocols, which function mainly for users to discover available end user services or for connecting office equipment or personal technology devices. In addition to these, the Secure Service Discovery Service (SSDS) of the UCB Ninja research project [8] targets users locating end user services over a wide area. Future directions for the Ninja project are to support purchasing services and compute power as commodities, at which time we can expect there will be an even stronger parallel to our work.

The IBM Océano research project [21] tackles a problem shared with our work: reconfiguring resources of a single data center that are dynamically allocated among multiple customers according to demand. For example, their paper mentions automatically reprogramming VLAN-based networks using SNMP when a server is reallocated among customers. The Muse project at Duke University [27] reprograms switches to serve requests from dynamic server sets allocated among customers. Their paper also discusses use of diskless servers for the same purposes of efficient reallocation across software environments as in our work. Further information on the protocols and mechanisms by which notification of state changes is performed and reconfigurations are triggered in these projects is not available to us at this time, but it seems a standards-based solution as we propose would fit perfectly.

Distributed infrastructure composition applies to various technologies termed ''metacomputing'', ''grid computing'', and ''peer-to-peer''. Although these technologies are primarily aimed at assembling large-scale computing power for supercomputing applications or at wide-area resource sharing or collaboration, the focus is beginning to also include enterprise applications and service providers. The emphasis is on such resource-oriented topics as discovery, description, allocation, and quality of service. Generally, applications employ new distributed resource access middleware or adaptations of existing distributed computing APIs such as MPI and CORBA. Darwin [18] concentrates on network resource management for distributed service providers; resources are tied together in a ''virtual mesh'' through brokers. Globus [19] and Legion [20] implement distributed computing middleware, including resource brokering and mechanisms for ongoing discovery of information on various aspects of resources in the distributed system, based on such protocols as LDAP and NIS. Our work differs from these in that it is focused on automated composition of service components in an enterprise infrastructure rather than on automated composition of resource components of a distributed application (over a partially pre-configured infrastructure). But it doesn't seem a far reach to adapt these technologies for our purposes. An automated infrastructure composition framework could complement these technologies for applications that interact with a dynamic set of peer applications or underlying infrastructure services. This could be achieved by extending the middleware associated with the technology or by leveraging emerging service discovery standards as we propose, with a possible advantage for lighter-weight standards-based protocols that may lend themselves to use in a wider variety of devices.

Certain of these projects, and others such as the IETF Middlebox Communication Working Group [13], deal with protocols for communication of policy and/or quality of service information regarding a particular networking packet flow among the devices that participate in the conversation. For example, the appropriate firewall ports may be automatically opened for the duration of an IP telephony session. Automated configuration of end-to-end infrastructure for a particular service conversation is analogous to our goals of automated configuration of an enterprise infrastructure according to management policy, and may be a point of further future investigation. A number of projects, including FLASH [17] and previous work at NASA Ames [24], distribute service configuration files from a central repository, using facilities such as ONC NIS or UNIX r-commands (rcp, et al). The main goal is to keep multiple systems in sync rather than to dynamically form associations between systems, although such concerns as locating appropriate NFS servers do appear (the NASA Ames setup uses DNS resource records for this purpose, assuming the information does not change very often). These systems deal primarily with higher-level services configured using files and running on general- purpose servers that support file systems and file transfer protocols over an enterprise network that has been previously composed. The central site publishes detailed service-specific configuration information, rather than generalized information that is interpreted by service-specific agents on the remote nodes.

Conclusions

We have described a framework for automated discovery and configuration of the designated deployment of services on a resource based on centralized management instructions, and for automated configuration of interdependencies between infrastructure services. The management instructions describe designated service deployments in high-level terms (“deploy an Apache Web server on system server127 for customer tom.com”); the framework automates the process whereby the various enterprise components are reconfigured to match these instructions. Information on the membership of resources and services, as discovered through common context identifiers, is often sufficient to automate configuration of interactions between services, and can serve as a starting point for discovering more specific service configuration information via the published attributes and via descriptions associated with the published context identifiers.

We suggest that a standardized infrastructure subscribing to such a framework instead of proprietary administration interfaces would play a key role in more quickly reconfigurable, and more reliably reconfigurable, enterprise networks and services that scale to very large networks by distributing configuration intelligence among the various participant services. This technology can also serve as a foundation for virtual enterprise networks comprised of resources from a possibly dynamic set of disparate administrative domains by automatically configuring interactions needed across domains. We further propose extending existing statically configured mechanisms for starting and stopping services in general-purpose operating systems to handle such dynamism.

Existing service discovery protocols may be readily adapted for these purposes. It is our position that use of lightweight and standards-aligned protocols encourages adoption in the widest variety of devices and applications. In the case of the Service Location Protocol, proposed extensions for deployment notification subscriptions and for Directory Agent interactions, together with further recommendations in this paper for wildcard service queries and subscription predicates, would greatly enhance its usefulness for disseminating the information needed for automated infrastructure composition.

Acknowledgements

The author gratefully acknowledges Tom Wylegala, Lance Russell, Dejan Milojicic, Sven Graupner, Baila Ndiaye, and Dan Conway of HP Labs, and Jon Stearley and Paul Anderson of USENIX LISA, for their comments and suggestions on this paper. Thanks also to Lance for the iNIC and software interface and to Steve Hoyle and David A. Barrett of HP Labs for use of their diskless boot setup.

Availability

Bug fixes to OpenSLP developed during this effort have been contributed back to OpenSLP. OpenSLP enhancements and source code and documentation for the software used in our prototyping may be downloaded at http://www.hpl.hp.com/personal/Todd_Poynor/ . There may also be a future effort to provide these features in Kempf & Associates SLP for Win32, see http://www.moeller-antik.com/~guttman/kaslp.htm .

Author Information

Todd Poynor has spent his entire professional career of 16 years at Hewlett-Packard in research and development of the HP 1000 Real Time Executive, the HP-UX kernel and system utilities, and most recently a number of technologies for computing platforms and Internet infrastructures. Reach him at postal address 1501 Page Mill Rd, Palo Alto, CA, 94304-1126 or at e-mail address todd_poynor@hp.com.

References

[1] Guttman, E., et al., “Service Location Protocol, Version 2,” *IETF RFC 2608,* June 1999.

[2] Goldand, Yaron, et al., "Simple Service Discovery Protocol /1.0, IETF Internet-Draft draft-cai-ssdp-v1-03.txt, October 1999.

[3] Jini home at http://www.sun.com/jini/ .

[4] E-Speak home at http://www.e-speak.hp.com .

[5] Salutation home at http://www.salutation.org/ .

[6] UDDI home at http://www.uddi.org/ .

[7] Gulbrandsen, A. and Vixie, P., "A DNS RR for specifying the location of services," *IETF RFC 2052,* October, 1996.

[8] Gribble, Steven, et al., "The Ninja Architecture for Robust Internet-Scale Systems and Services," *Computer Networks Special Issue on Pervasive Computing*, Vol. 35, No. 4, March, 2001.

[9] Scott, Donna, "Commentary: More Hardware Won't Solve Web Site Outages," *Gartner Viewpoint*, January, 2001.

[10] Buyya, Rajkumar and Vazhkudai, Sudharshan, "Compute Power Market: Towards a Market-Oriented Grid," *Proceedings First IEEE International Conference on Cluster Computing and the Grid*, May, 2001.

[11] Kempf, James and Goldschmidt, Jason, "Notification and Subscription for SLP," *IETF Internet-Draft draft-kempf-srvloc-notify-05.txt*, January, 2001.

[12] IETF Middlebox Communication Working Group home at http://www.ietf.org/html.charters/midcom-charter.html .

[13] Yaffe, Joel, "Why Established Hosting Providers Should Fear Loudcloud." *Giga Information Group IdeaByte*, August, 2000.

[14] Schatt, Stan, "Enterprise Network Building Blocks: Crafting an E-Business Infrastructure," *Giga Information Group Planning Assumption*, March, 2000.

[15] Carpenter, Brian E., "Architectural Principles of the Internet," *IETF RFC 1958*, June 1996.

[16] Zhao, Weibin, et al., "mSLP – Mesh-enhanced Service Location Protocol," *IETF Internet-Draft draft-zhao-slp-da-interaction-09.txt*, October, 2000.

[17] de Silveira, Gledson Elias, and Fabio Q. B. da Silva, "A Configuration Distribution System for Heterogeneous Networks," *Proceedings Twelfth Systems Administration Conference (LISA 1998)*, December, 1998.

[18] Chandra, Prashant, et al., *Darwin: Customizable Resource Management for Value-Added Network Services*, November, 2000.

[19] Foster, Ian, Carl Kesselman, and Steven Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, 2001.

[20] Grimshaw, Andrew, Adam Ferrari, et al., "Legion: An Operating System for Wide-Area Computing," *IEEE Computer*, Vol. 32, Num. 5, May, 1999.

[21] Appleby, K., L. Fakhouri, et al., "Océano – SLA Based Management of a Computing Utility," *Proceedings IFIP/IEEE International Symposium on Integrated Network Management*, 2001.

[22] Wilkes, John, Patrick Goldsack, et al., "Eos – The Dawn of the Resource Economy," *Proceedings HotOS VIII*, May, 2001.

[23] OpenSLP home at http://www.openslp.org .

[24] Traugott, Steve and Joel Huddleston, "Bootstrapping an Infrastructure," *Proceedings Twelfth Systems Administration Conference (LISA 1998)*, December, 1998.

[25] Howes, Tim, "The String Representation of LDAP Search Filters," *RFC 2254*, December, 1997.

[26] National Institute of Standards and Technology, "Digital Signature Standard," *Technical Report NIST FIPS PUB 186*, U. S. Department of Commerce, May, 1994.

[27] Chase, Jeffrey B., Darrell C. Anderson, et al., "Managing Energy and Server Resources for a Hosting Center," *Proceedings Eighteenth Symposium on Operating System Principles (SOSP)*, October, 2001.

[28] Burgess, Mark and R. Ralston, "Distributed Resource Administration using cfengine," *Software – Practice and Experience*, Vol. 27, p. 1083, 1997.