Faunus: a flexible middleware for specifying and managing multimodal, multiparty collaborations

Surendar Chandra[*] and Maribeth Back
FX Palo Alto Laboratory Inc., 3174 Porter Drive, Palo Alto, CA 94304
surendar@acm.org, back@fxpal.com

ABSTRACT

Faunus addresses the challenge of specifying and managing complex collaboration sessions. Many entities from various administrative domains orchestrate such sessions. Faunus decouples the entities that specify the session from entities that activate and manage them. It restricts the operations to specific agents using capabilities. It unifies the specification and management operations through its naming system. Each Faunus name is persistent and globally unique. A collection of attributes are attached to each name. Together, they represent a collection of services that form a collaboration session. Anyone can create a name; the creator has full read and write privileges that can be delegated to others. With the proper capability, anyone can modify session attributes between an active and inactive state. Though the system is designed for Internet scale deployments, the security model for providing and revoking capabilities currently assumes an Intranet style deployment. We have incorporated Faunus into a DisplayCast system that originally used Zeroconf. We are incorporating Faunus into another project that will fully exercise the power of Faunus.

Categories and Subject Descriptors

H.4.3 [**Information Systems Applications**]: Communications Applications

General Terms

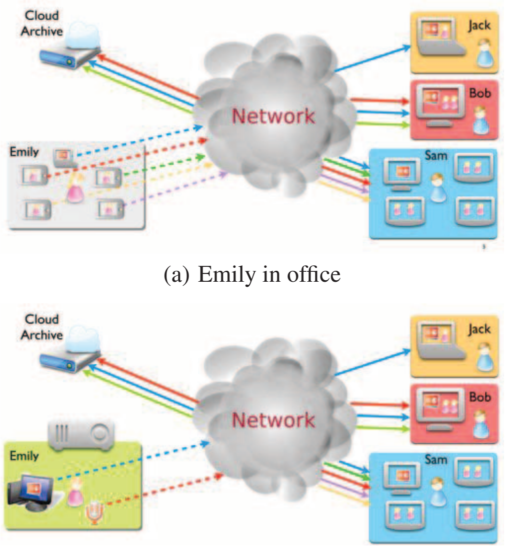
PERFORMANCE, EXPERIMENTATION

Keywords

session management, session specification

1. INTRODUCTION

Faunus is designed to manage collaboration amongst group members from different administrative domains. Members use different modalities (e.g., audio, video) from various infrastructure elements that may only be assigned for the collaboration duration (e.g., projectors). We illustrate a typical usage scenario from the perspective of Emily in Figure 1. Emily is collaborating with Bob, Jack and Sam. At first, she uses DisplayCast [4] to share her presentation in real-time as well as archive them to a MPEG4 movie (Figure 1(a)). She uses four cameras that capture her from all directions and shares them with Bob and Sam. On the client end, these videos are composed to provide a rich tele-immersive experience [17]. Bob's network restricts him to watch two video streams in a stereo setup. Bob, Emily, Jack and Sam do not belong to the same organization. Because of a policy restricting video conferencing with outsiders, Emily does not share videos of herself with Jack. Partway, she moves to the conference room (Figure 1(b)). Suppose an indoor localization system exists that detects she has moved to the conference room, her persona and sessions are transitioned to a pre-planned service configuration for the conference room. While she retains her communication endpoint to continue the collaboration, the services available to her change. She gains a projector that allows Bob to present his results, a desktop to continue her presentation and a full-room audio capture system while losing her cameras. While she walks from her office to the conference room, the localization system dynamically assigns her hallway cameras so that she can continue her collaboration.



(a) Emily in office

(b) Emily in conference room

Figure 1: Target usage scenario

This scenario illustrates several research challenges. Judicious

[*] Current affiliation: Backup Recovery Systems Division, EMC Corp.

Middleware 2012 Industry Track, December 3–7, 2012, Montreal, Quebec, Canada.
Copyright 2012 ACM 978-1-4503-1613-2/12/12 ...$15.00.

use of networking and processing resources for synchronized multiparty collaboration is crucial. Faunus targets the session specification challenge. Emily needs a means to name and describe the components that will form her communication end point. During active collaboration, system agents monitor her movement from her office to the conference room and then deactivate her office session and activate her conference room setup. Since Emily collaborates across organizational boundaries; each agent requires appropriate privileges to complete their action. For example, Emily requires four cameras (e.g., named *cam-front-left*, *cam-front-right*, *cam-back-left*, *cam-back-right*) that each support resolutions of 1920 x 1080 along with a computer that can show her PowerPoint presentation *Faunus.ppt*. She also needs to describe the sessions composed of these components; she wants her presentation to be shown on *Bob*'s, *Jack*'s and *Sam*'s Player [4] and archived into a movie file `/archives/EmilyDescribingFaunus.mp4`. Even though she refers to the components with her own choice for a name, they are part of the infrastructure fabric. The network camera referred to by Emily as *cam-front-left* might actually be referred to as *cm3214* by the support staff. Similarly, *Bob*'s player is mapped to the nearby shared player at the local cafe. Once mapped, each component exposes its operating parameters to other participants. For example, the camera specifies that it can create an 8 Mbps stream forcing *Bob* to not watch all the videos using the cafe WiFi hotspot.

Existing solutions do not allow entities that span different administrative domains from operating on a shared session configuration. Faunus unifies session specification and management operations. It achieves its goal by disentangling mechanism from policy. Faunus provides a way to describe the policy: the components that stream from one to another and the configuration of each component that participates in the collaborative session. Agents operate on the Faunus name to activate the collaboration session. They reflect the current state of a session by modifying the name attributes. The security capability restricts such actions to authorized components. New mechanisms are incorporated with minimal changes; making the systems extensible. Faunus name configurations persist even when the sessions are inactive; allowing for seamless transition between planning and managing sessions. We show that Faunus specifications are powerful enough for several usage scenarios.

Next, §2 lists related work. §3 describes the Faunus name space while §4 describes its architecture and the API. §5 shows how DisplayCast uses Faunus with conclusions in §6.

2. RELATED WORK

Faunus provides a means to name services. It also expresses the session policy between services using the naming system. Naming [21] is a fundamental distributed system service. Recently deployed systems either managed services (e.g., Dynamic DNS (DDNS) [25], Lightweight Directory Protocol (LDAP) [30]) or used an ad hoc scheme (e.g., Apple Multicast DNS (mDNS) [9], Microsoft Simple Service Discovery Protocol (SSDP) [10]).

Managed services provide a definitive way to name services using trusted servers. They maintain their integrity by exercising strict control over update policy. LDAP users are typically restricted to updating names that are bound to their organizational privilege. For example, Emily might be allowed to modify names in */organization=fxpal/group=mir/project=displaycast/* while Bob manages names in */organization=berkeley/group=eecs/*. DDNS defines mechanisms for secure record updating as well [26]. Note that DDNS operates on a global scale while LDAP is restricted to organization level naming. Their update policies are too rigid to name sessions that include more than one service. For example, a session between Emily and Bob could be named by either of them but restricted from updating session attributes by the other participant. A solution of removing all restrictions and allow complete read/write access is unsatisfactory even in an Intranet. Ad hoc services manage names without servers. Each service chooses its name in an uncoordinated, first come basis. Each active service directly responds to name queries from other participants. For example, mDNS uses link-local multicast while SSDP uses HTTP over link-local multicast and unicast UDP to advertise and locate services. Inactive services as well as naming sessions that involve two or more services are unsupported and require a proxy agent. Since responses originate from a single agent, two actors cannot modify naming attributes; all updates need to be sent to the name maintainer using a separate RPC protocol. Faunus incorporates features from both these approaches. Similar to ad hoc services, Faunus allows anyone to create a name. Similar to a managed scheme, inactive services do not require a proxy. The creator can selectively delegate their read/write credentials. Like Faunus, most modern naming schemes can attach a list of attributes and values with each name. (e.g., TXT records for DDNS and Zeroconf).

DisplayCast [4] currently uses both paradigms. It uses Zeroconf to manage names. Each entity uses a globally unique ID (GUID) as its name to avoid name collisions; user friendly names are stored as a name attribute in the DNS TXT record. Thus, finding a specific component requires a multicast query to locate all GUIDs followed by unicast queries of each individual service to collect their user friendly name so that it can be presented to the user. DisplayCast also uses a centralized controller that collects the soft-state attribute updates and services them via HTTP/REST. The session between a Streamer and the Player/Archiver is initiated by the Player/Archiver and is transient. Faunus extends this scheme to provide persistence to services and to allow others to manage the attributes.

Data-Oriented Network Architecture [14] replaces DNS names with a flat, self-certifying name. It replaces DNS name resolution with a name-based anycast primitive that lives above the IP layer. It focuses on the persistence, authenticity, and availability of each name. It uses self-certifying names to maintain integrity. Faunus names services and sessions. It manages their integrity using capabilities. Ford et al. [12] provide a globally persistent personal name and a means for entities to continue communicating with each other. Faunus currently does not manage the reachability between services. Kalofonos et al. [13] introduce *Passlets* to allow users to share access to their personal devices; capabilities play a similar role in Faunus. They focus on the usability of personal sharing; Faunus application developers can benefit from their experiences.

Prior research investigated ways of discovering active services and the means of interacting with them. Service oriented architectures such as Java Jini (renamed Apache River - `http://river.apache.org`) provide mechanisms for services to register themselves to a locator service. Clients first lookup this locator service and then use Java Remote Method Invocation (RMI) to migrate the remote service for local interaction. The Salutation effort [2] investigated similar goals in a portable fashion. When active, services registered themselves with a management service. The management service built a catalog of online services along with their capabilities and location of device drivers required to communicate with each service. Clients searched this catalog to locate compatible services and then communicated with them through the APIs provided by the management service. Prior systems were not designed to operate with inactive sessions. Faunus describes the participants in a session and manages the capabilities that allows entities to configure them. Using appropriate capabilities, independent software agents transition between active and inactive states.

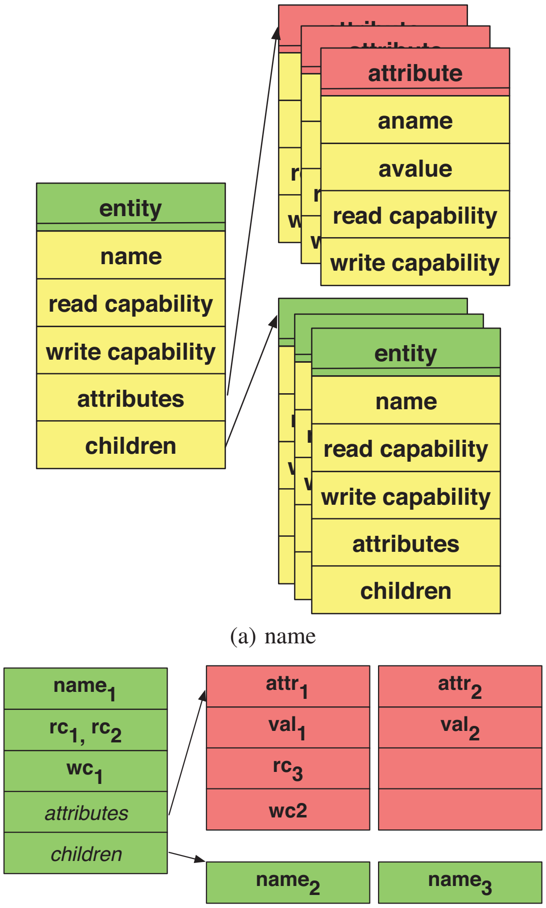
Earlier systems developed ways of describing and controlling synchronous interaction between services. SIP and H.323 [24] defines signaling and control of Internet telephony and conferencing applications. The Maestro system [3] achieves tight synchrony between collaborating services with consistency guarantees on message delivery, membership changes and failures. Machnicki et al. [18] describes the Virtual Director that automates the tasks required to capture and stream lecture webcasts [28]. Their system can control the recording equipment, configure stream broadcasting parameters and effect camera control. Through a Tcl/Tk based automation control script, it automates camera view selection as well as content selection based on monitoring audience questions. Zhang et al. [31] develop a scripting language that is capable of expressing the cinematography rules necessary to automate lecture capture and distribution. Faunus only describes the participants of a session; another agent is expected to activate and manage the desired session. The session configuration parameters will be stored within the Faunus attribute store. We argue that the key-value attribute store is powerful enough to provide this functionality.

Broadcast Management System (BMS) [27] and Indiva [22] share similar design goals as Faunus. Faunus can be used to reimplement BMS and Indiva. Faunus can also help provide additional functionality. BMS manages Internet MBone broadcasts. It stores a description of software components and their configurations in a Postgres database. BMS uses data base schemas that are optimized for managing MBone sessions. Faunus name attributes can replace the BMS postgres database. Since Faunus attributes are globally accessible, the broadcast configurations can be shared across multiple BMS instances. Since the Faunus attribute names are generic, BMS can be extended to support other broadcast mechanisms besides MBone. Faunus capabilities can also allow fine grain control by different operators. For example, some operators can initiate a BMS session while others can change the multicast addresses used by the same session. Indiva uses a file system metaphor to name and access resources in a hierarchical name space. A resource can be aliased and appear in multiple places in the hierarchy using file links. Each Indiva instance uses a different tree with no means of sharing name spaces across instances. Names for static services are predefined by the Indiva node administrator. Directory names for software processes, active sessions and streams are created on-the-fly when processes are launched or when a soft-state announcement is received. Attributes of active sessions are available in a hidden file under each dynamic directory. Indiva provides a Tcl based command shell (as well as a GUI interface) to operate on services using their names. It also provides common tasks such as *encode* and *record* that can be configured via the command shell and associated with services and devices through the name space. Indiva command shell can use Faunus names to encapsulate Indiva directories. Indiva directory attributes are available through the Faunus attribute collection. Since Faunus names are global, they can be shared among multiple Indiva instances. Faunus names are persistent and so can be used to define active and inactive sessions.

3. FAUNUS

First, we describe the Faunus data model. A prototype implementation is presented in the next section.

Faunus names (Figure 2(a)) do not encode any information about the location or state of the service being named and hence *pure* [20]. Thus, Faunus names can be used to describe as well as migrate services and sessions across organizational boundaries. We use a 128 bit GUID [15] to name objects. GUIDs are guaranteed unique across space and time. Since the names themselves do not offer guidance on their location, our prototype uses a centralized approach. We used Redis (`http://redis.io`) for storage. Redis has acceptable performance [19], at least for an organization level storage. Recent developments in scalable key-value stores [7] makes centralized approaches practical for Internet scale deployment. Distributed mechanisms [23] can also be used.



(a) name

(b) sample

Figure 2: Faunus name

A collection of attribute-value tuples store information about each name. A Faunus name without any attribute is functionally equivalent to a non-existent name. Faunus itself does not reserve any particular attribute name. For comparison, using a combination of DNS SRV, A, and AAAA records, Zeroconf [1] defines the service name, protocol name, port, IPv4, and IPv6 addresses for all names. As an Intranet focussed service, Faunus itself does not place any limitation on the size of the attributes.

Each name can store an unordered collection of children. The specific semantics on how these children relate to this name is left to the application developer. Services that require an ordered collection of children can encode this order into the name of the children (e.g., "2-GUID$_a$" and "1-GUID$_p$" to order GUID$_p$, GUID$_a$ in ascending order). It is also possible to store the children as attributes to the name. However, as we will see in the next section, read and write privileges of attributes can be protected with capabilities while children are public. Since attributes of each name itself can be protected with capabilities, no private information is lost in knowing whether a GUID name is a child of another.

3.1 Capabilities for delegation

The ability to read and modify Faunus name attributes are protected using a collection of capabilities [16]. Capability tokens anonymously grant specific privileges to its holder. When a Faunus name is created, we create two 64 bit integers by storing 32 bit random numbers created by *arc4random()* in the high and low order bits and designate them as read and write capabilities for attributes of that name. These tokens are delegated by sharing them with other agents. Agents with write capability can create additional capabilities as well as revoke existing capabilities. For example, a limited duration read access can be granted by creating an additional read capability and then sharing them. When the access duration is over, this new capability is revoked. Faunus also allows fine-grain control over individual attributes; each attribute can have their own read and write capability. The individual capabilities take precedence over the name capability. Revoking all read or write capabilities makes the attributes public.

We illustrate a sample service named $name_1$ in Figure 2(b). $name_1$ is the parent of $name_2$ and $name_3$. The application that uses this name defines the semantics of the relationship between $name_1$, $name_2$ and $name_3$. $name_1$ has two attributes: $attr_1$ and $attr_2$, with values val_1 and val_2, respectively. Capability rc_1 or rc_2 is required to read the values for attribute $attr_2$ and rc_3 is required to read the value of $attr_1$. Write capability wc_1 is required to add a new attribute to $name_1$ while wc_2 is required to modify $attr_1$. Revoking rc_1 and rc_2 would make $attr_2$ publicly readable though $attr_1$ still requires rc_3.

Applications that use the Faunus middleware are expected to manage their collection of capability tuples $< name$, $attribute$, $read|write$, $capability >$ and present them to Faunus for access to the name attributes. In §4.3, we describe per user credential wallets that makes this transparent for most operations.

3.2 Faunus API

Faunus exposes the following API to applications:

- CREATENAME(*type*, *public*): creates a new Faunus name. If *public*, publishes it on a whiteboard server in a user defined group, *type*. The API automatically stores the *name* in the postit service and the read/write credentials in the wallet. Note that Faunus is designed for large scale deployments; it does not support listNames or deleteName functionality.
- ADDCHILD(*name*, *child*) and DELCHILD(*name*, *child*) adds and deletes the specified Faunus *child* from the *name*, respectively. LISTCHILDREN(*name*) returns the collection of children of *name*. Adding and deleting children requires write privileges on the *name* while listing requires read privileges. Faunus middleware automatically presents the appropriate credentials from the wallet. When the user does not possess appropriate credentials, the request is denied.
- ADDATTR(*name*, *attr*, *value*), GETATTR(*name*, *attr* and DELATTR(*name*, *attr*) adds a new attribute, gets the value of a current attribute and deletes an existing attribute, respectively. Appropriate credentials are presented from the credentials wallet. We also provide a LISTATTRS(*name*) to list all the attributes for the *name*.

4. FAUNUS PROTOTYPE ARCHITECTURE

We implemented Faunus using a system illustrated in Figure 3. Faunus interacts with a number of services: a *Faunusd* server that uses a key-value store to archive the name records, a *Whiteboard* server to partially publish names and a per user capability *Wallet* and *Postit* service. We expect at least one instance of the *Whiteboard* and *Faunusd* servers per Intranet domain though users will not always interact with servers that belong to the same domain. Next, we describe the design rationale behind each component.

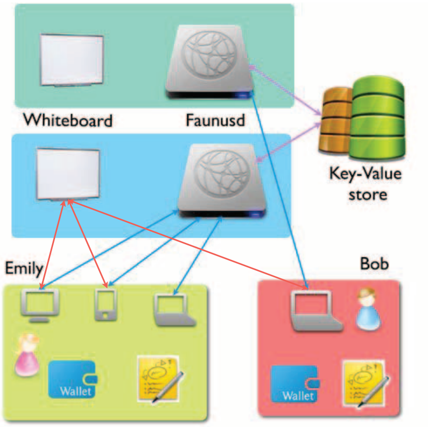


Figure 3: Faunus architecture

4.1 Faunusd server

The Faunus middleware interacts with the *Faunusd* server for its operation. Our prototype uses HTTP/REST GET method for interacting with *Faunusd* and receives JSON strings for responses. The *Faunusd* server uses the GUID name as the key and stores its state as a JSON string in a Redis key-value store. Thus, any number of *Faunusd* servers can be deployed against the key-value store to achieve good scalability. All transactions are serialized through the key-value store. Consistency analysis while using a number of *Faunusd* servers operating on the same name, especially when using a distributed key-value store is a topic for future research. The *Faunusd* server and the key-value store form the trusted computing base; the key-values stored are not publicly accessible.

Portable URLs should be under 8000 characters long [11]. Thus, our choice of HTTP/REST over GET places a practical limit on the attribute size. HTTP POST based approach or a different remote procedure call (RPC) mechanism will alleviate some of this limitation. Though storage is plentiful, the ability of Faunus to create an unlimited number of new names will place some practical limits on the size of the per name attribute collection. For comparison, Apple iCloud limits its free key-value store to 1 MB per app with a total key limit of 1024 for each account. Amazon DynamoDB provides storage of 100 MB, 5 writes/second and 10 reads/second for free with tiered pricing beyond those limits.

4.2 Per user capability wallet

Collections of capabilities protect reading and modifications of the name attributes as well as for specific attributes. For example, reading the value of $attr_2$ in $name_1$ requires either rc_1 or rc_2 (Figure 2(b)). Each Faunus user might have a subset of these capabilities; some of those might have already been revoked. Faunus middleware fully manages the user's capabilities in a wallet and makes dealing with capabilities transparent (except for functions that delegate and share capabilities). The wallet provides an API to add, list and delete capabilities. Faunus uses these APIs to get all capabilities available to the user (potentially rc_1 and rc_2) and then sequentially uses each of them until access is granted. The APIs also support capability delegation by exporting a collection of capabilities in an opaque (internally sqlite3) database.

Modern users frequently use a variety of laptops, desktops and mobile devices. For example, Emily regularly uses a tablet, laptop and a desktop while Bob only uses his laptop (Figure 3). A shared capability wallet is useful. For example, 1Password (`https://agilebits.com/onepassword`) supports secure sharing amongst a variety of desktop and mobile devices using various commodity cloud storage services (e.g. DropBox) while Google Wallet uses its own secure storage. Prior systems are proprietary without any public API for our use. Hence, we developed a service that periodically reconciles local changes to the wallet with a global per user wallet. Since our wallet does not support modification primitives, update conflicts are not a concern [29]. When the global wallet is not synchronized with all the user's devices, they may be unable to access an attribute that the user received access in another (potentially disconnected) device [5]. The security of Faunus is never compromised by reconciliation delays. The wallet itself needs to be secure lest the users access rights become public. Our prototype does not incorporate a more rigorous security mechanism to protect the wallet while operating across multiple trust domains. The current mechanism is however adequate for Intranet scenarios.

4.3 Utilities: Whiteboard and Postit service

Faunus names can be shared through ad hoc mechanisms (e.g., email, NFC etc.). We also provide a Whiteboard service to publicize a Faunus name. The names are grouped by an application specified type (e.g., IANA types [6]). Others can browse these published names. The whiteboard servers are not synchronized amongst each other. Since Faunus operates on a global scale, the listing on any particular whiteboard server gives a partial snapshot of all the publicly available names. In DisplayCast, we use this service to replace Zeroconf browse functionality.

We also provide a per-user postit service that automatically keeps track of all the names that were created by the user as well as names that were discovered or shared to it. Postit service reconciles its state amongst all the user's devices. Faunus name attributes are protected by capabilities; the whiteboard and postit services are not part of the trusted base.

4.4 Whiteboard and Postit API

We expose the following API to the user applications:

- Postit service provides REMEMBERNAME(*name*, *type*), LISTNAMES(*type*) and FORGETNAME(*name*, *type*). Updates are propagated to all the user's devices. Deletions create death certificates [8] that are propagated for a fixed duration.
- Whiteboard is accessed using BROWSELOCAL(*type*), REGISTERNAME(*name*, *type*) and UNREGISTERNAME(*name*, *type*). Whiteboard servers are not federated: BROWSELOCAL returns a partial list of all *names* that were classified as *type*.
- We also provide a means to export and merge select capabilities for delegation purposes. These functions are described in further detail in the source code.

5. USING FAUNUS

We have implemented Faunus and its support services using about 4,000 lines of Objective C code on Apple OS X/iOS. We are porting the client side middleware to Windows 7 using C#/.NET. These code artifacts are publicly released as part of DisplayCast [4]. Next, we show how we integrated Faunus into an existing DisplayCast system as well as how Faunus is used to develop distributed location management software agents.

5.1 Faunus in DisplayCast

This section describes two implementations of DisplayCast. The first uses Zeroconf to manage resources, and the second uses Faunus.

5.1.1 DisplayCast using Zeroconf

DisplayCast [4] originally used Zeroconf to advertise sources (i.e., Streamer) and destinations (i.e., Player and Archiver) of screen sharing sessions. Each component independently kept track of all others. Synchronous access to Zeroconf records were provided using a ControlProxy that collected the Zeroconf soft-state announcements. DisplayCast solved the name collision problem of Zeroconf by naming each component with a GUID. The user specified names were advertised as a DNS TXT attribute. Persistent parameters such as the GUID name and user specified attributes of each component were stored in the operating system specific application preferences database. Application preferences of users who used a mobile home folder were shared among all of their devices. However, DisplayCast parameters are unique to each device. Hence, we augment each parameter by a value unique to each host: the MAC address of the network interface. For example, a *Streamer* using a laptop with a Ethernet MAC address of *eth-mac* would store *Streamer-eth-mac:id=*$guid_1$*, name="Surendar Laptop"*. The *Players* advertised parameters of each active session using DNS TXT records (and captured by the ControlProxy).

The *Streamer* uses a push metaphor to *Project* themselves to *Players* and *Archive* to *Archivers*. The *Player*s and *Archiver*s use a pull metaphor to *Watch* as well as *Archive Streamer* contents. We also provided a HTML5 controller that interacted with the ControllerProxy. It used a *world-in-miniature* metaphor to control the various components. For example, users resized the *Streamer* session window on the *Player* using the HTML5 interface.

The current DisplayCast maintains its persistent state in the preferences database which are advertised while the components were online. It does not support persistent sessions; each session between a source and destination creates a new session id that is advertised through Zeroconf. Once users had configured a DisplayCast session using the HTML5 service, any failure of the *Player* requires a manual restart and reconfiguration of the sessions.

5.1.2 DisplayCast using Faunus

We incorporated Faunus into DisplayCast. Instead of using local user preferences database, all persistent attributes are stored in Faunus. Sessions are represented as Faunus names and are persistent. Sessions are only destroyed after an explicit action. Restarting the *Player* attempts to restart all the active sessions. As long as the components are online and available, a fully configured setup automatically restarts itself. The HTML5 controller only needs to ensure that the session is active and hence is always in sync with the design of a particular session. Currently, we locate the Faunus name associated with a particular device to keep the functionality similar to the legacy DisplayCast. We are investigating the UI control elements necessary for treating all user sessions on any of the user's devices as shared and that could be instantiated in any of their currently active devices. The changes required were minimal; under 100 lines of Objective C calls to the Faunus framework.

5.2 Software agents

Faunus was motivated by the need to use software agents to offload some of the functionality required by DisplayCast. For example, indoor localization can benefit users by locating nearby projectors. For this functionality, we require the DisplayCast components as well as users to be located. We have deployed a Cisco WiFi localization and a custom Bluetooth localization system. DisplayCast Player can then identify its locatable components (MAC ID of WiFi and Bluetooth) and register them with a agent which will monitor where such device is currently located. Prior to Faunus, there was no way for the location agent to record its findings other than to implement a custom RPC interface between the agent and the Player. With Faunus, the agent only requires the $< name$, $attribute$, $capability >$ to read and modify the location attribute.

We are currently incorporating this functionality using Faunus.

5.3 Empirical performance analysis

Our architectural choices (e.g., key-value store to save state as well as serialize updates) makes Faunus scalable. We have deployed the redis, Faunusd and Whiteboard servers in a Mac Mini (OS X 10.7.4, 2.7GHz Intel i7, 8GB). CREATENAME() took 13.9 msec while also publishing the name to a Whiteboard (with the same redis server) took 15.5 ms. 25% of this time was spent on local string processing and creating the GUID with the remaining attributable to HTTP REST interaction with Faunusd as well as with the redis server. This performance is adequate; name and attribute manipulation is not expected to be in the application critical code path. Further experiments will investigate the scalability claims.

6. DISCUSSION

Faunus allows agents to name and manage attributes of services and sessions in a uniform fashion. Capabilities restrict the set of actors that are allowed to operate on each name. The current prototype offers good performance and is flexible. Much work is needed in porting the client libraries to more platforms as well as incorporate Faunus into a variety of applications. The prototype is open sourced through the DisplayCast project.

Acknowledgements

We thank Larry Rowe for his comments and suggestions.

7. REFERENCES

[1] Zero configuration networking (zeroconf). `http://www.zeroconf.org/`.

[2] Salutation architecture specification (part-1) version 2.0c. Technical report, Salutation Consortium, June 1999.

[3] K. P. Birman, R. Friedman, M. Hayden, and I. Rhee. Middleware support for distributed multimedia and collaborative computing. *Softw. Pract. Exper.*, 29(14):1285–1312, Dec. 1999.

[4] S. Chandra and L. A. Rowe. DisplayCast: a high performance screen sharing system for intranets. In *ACM Multimedia 2012*, Nara, Japan, Oct. 2012. `https://github.com/DisplayCast`.

[5] S. Chandra and X. Yu. A trace-driven analysis of wireless group communication mechanisms. *ICST Mobile Communications and Applications Journal*, 2012.

[6] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire. Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry. RFC 6335, 2011.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *SOSP '07*, pages 205–220. ACM, 2007.

[8] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC*, pages 1–12, Aug. 1987.

[9] Dns service discovery. `http://www.dns-sd.org/`.

[10] A. P. et al. UPnP device architecture 1.1. Technical report, UPnP Forum, Oct. 2008.

[11] R. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, T. Berners-Lee, Y. Lafon, and J. F. Reschke. Http/1.1, part 1: Uris, connections, and message parsing. (RFC 2817,

[12] B. Ford, J. Strauss, C. Lesniewski-Laas, S. Rhea, F. Kaashoek, and R. Morris. Persistent personal names for globally connected mobile devices. In *Symposium on Operating systems design and implementation*, OSDI '06, pages 233–248, Seattle, Washington, 2006.

[13] D. N. Kalofonos, Z. Antoniou, F. D. Reynolds, M. Van-Kleek, J. Strauss, and P. Wisner. Mynet: A platform for secure p2p personal and social networking services. In *PERCOM '08*, pages 135–146, Hong Kong, Mar. 2008.

[14] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *SIGCOMM '07*, pages 181–192, Kyoto, Japan, 2007. ACM.

[15] P. J. Leach, M. Mealling, and R. Salz. A universally unique identifier (uuid) urn namespace. (RFC 4122), July 2005.

[16] H. M. Levy. *Capability-Based Computer Systems*. Butterworth-Heinemann, Newton, MA, USA, 1984.

[17] J. Liang, Z. Yang, B. Yu, Y. Cui, K. Nahrstedt, S.-H. Jung, A. Yeap, and R. Bajcsy. Experience with multi-camera tele-immersive environment. In *NSF Experience Workshops on Pervasive Computing and Cluster Computing*, 2005.

[18] E. Machnicki and L. A. Rowe. Virtual director: Automating a webcast. In *Multimedia Computing and Networking (MMCN '02)*, volume 4673, pages 208–225, Jan. 2002.

[19] Y. Mao, E. Kohler, and R. T. Morris. Cache craftiness for fast multicore key-value storage. In *EuroSys '12*, pages 183–196, Bern, Switzerland, 2012. ACM.

[20] R. M. Needham. Names. In S. Mullender, editor, *Distributed systems*, pages 89–101. ACM, New York, NY, USA, 1989.

[21] R. M. Needham and A. J. Herbert. *The Cambridge distributed computing system*. Addison-Wesley, 1982.

[22] W. T. Ooi, P. Pletcher, and L. A. Rowe. Indiva: A middleware for managing distributed media environment. In *SPIE/ACM Multimedia Computing and Networking (MMCN)*, Santa Clara, CA, Jan. 2004.

[23] A. Rowstron and P. Drushel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350, Nov. 2001.

[24] H. Schulzrinne and J. Rosenberg. A comparison of SIP and H.323 for Internet Telephony. In *NOSSDAV '98*, Cambridge, England, July 1998.

[25] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. Dynamic updates in the domain name system. RFC 2136, Apr. 1997.

[26] B. Wellington. Secure domain name system (DNS) dynamic update. RFC 2137, Nov. 2000.

[27] D. Wu, A. Swan, and L. A. Rowe. Internet mbone broadcast management system. In *Multimedia Computing and Networking 1999*, volume 3654, pages 41–51, Jan. 1999.

[28] T.-P. Yu, D. Wu, K. Mayer-Patel, and L. A. Rowe. dc: A live webcast control system. In *Multimedia Computing and Networking (MMCN 2001)*, Jan. 2001.

[29] X. Yu and S. Chandra. Designing an asynchronous group communication middleware for wireless users. In *MSWiM '09*, pages 274–279, Tenerife, Canary Islands, 2009.

[30] K. D. Zeilenga. Lightweight directory access protocol (LDAP). RFC 4510, June 2006.

[31] C. Zhang, Y. Rui, J. Crawford, and L.-W. He. An automated end-to-end lecture capture and broadcasting system. *ACM Trans. Multimedia Comput. Commun. Appl.*, 4(1):6:1–6:23, Feb. 2008.