IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922 © 2004 Published by the IEEE Computer Society
Vol. 5, No. 9; September 2004

A Middleware Architecture for Securing Ubiquitous Computing Cyber Infrastructures

Raquel Hill, *University of Illinois at Urbana-Champaign*
Jalal Al-Muhtadi, *University of Illinois at Urbana-Champaign*
Roy Campbell, *University of Illinois at Urbana-Champaign*
Apu Kapadia, *University of Illinois at Urbana-Champaign*
Prasad Naldurg, *University of Illinois at Urbana-Champaign*
Anand Ranganathan, *University of Illinois at Urbana-Champaign*

HESTIA , a programmable middleware solution implemented as a network of middleboxes, secures critical information services in large-scale ubiquitous computing environments. This programmable, distributed, object-oriented framework enables the integration of security, privacy, and reliability mechanisms in service-access interfaces and implementations.

Ubiquitous environments organize networked computer devices into a distributed system that cooperates and coordinates its activities with its users. Soon ubiquitous computing will extend beyond the boundaries of prototype experiments and encompass larger areas, enabling smart buildings, campuses, and fleets.[1] However, security, privacy, and fault tolerance are major hurdles for wide-scale deployment of the technology.

Today's users expect computing and information systems to be available even under attack, to perform their tasks in a timely manner, and to consistently provide accurate results. The problem of securing *critical cyber infrastructure* is more difficult in smart buildings. In such cases, the CCI must bind networks, processors, and devices with policies, mechanisms, protocols, and services to offer survivable and secure operations while providing better management and finer

integration between heterogeneous components. Yet, a secure CCI for smart buildings is essential because many of the building's services—surveillance systems; heating, ventilation, and air conditioning; lights; door locks; and so on—are critical to supporting its inhabitants.

We propose the Heterogeneous Survivable Trusted Information-Assurance Architecture, a middleware solution that provides a secure layer for CCIs such as smart buildings and other large-scale ubiquitous computing environments. HESTIA offers distributed deployment of mechanisms, such as access control, anonymity, replication, load balancing, auditing, and intrusion detection. (The "Related Work" sidebar discusses other middleware platforms.) We've begun implementing HESTIA in a smart building at the University of Illinois at Urbana-Champaign called the Seibel Center. The Siebel Center showcases state-of-the-art computing and communications infrastructure in its offices, meeting rooms, and classrooms. Digital locks; heating, cooling, and lighting controllers; video cameras; and other digital sensors and actuators pervade the building and are accessible through a private network. We're developing several services for controlling different functionality in this smart building, including locking and unlocking doors; controlling lights; and configuring heating, ventilation, and air conditioning systems. We're also developing applications that use the smart building environment as a whole —for example, services that unlock doors and turn on lights automatically for disabled persons moving in the environment. Once this implementation is complete, we hope to extend our work to allow the implementation of HESTIA in other smart buildings as well as environments that include multiple buildings or nontraditional structures (for example, aircraft carriers).

Example: Door-lock access at the Seibel Center

To illustrate HESTIA 's role in the Seibel Center smart building, we present the following example. Faculty, students, and visitors need reliable access to the computing, communication, and information services they're authorized to use. The building must also be secure against accidental software or hardware failures and against malicious attacks. Because the Siebel Center is an open academic environment, draconian access controls are not feasible; however, the building CCI's security and survivability is a key consideration. Moreover, the building's heterogeneous components and subsystems must be integrated to enhance interoperability.

All the doors in our building have e-locks, which open to authenticated, authorized users with swipe cards. These swipe cards rely on a networked lock server. The swipe-card detector sends a message over the network that causes a lookup in a door-lock access database associated with the lock server. If the system authorizes the user, the door opens in response to a reply message

from this server. We're extending this mechanism to provide additional services. For example, to assist people with disabilities, we'll offer UbiSense location technology (www.ubisense.net). UbiSense employs special tags that transmit ultrawide radio bands. The tags can be associated with particular users. Tag detectors can detect the presence and location of a user within six inches of the user's actual location. Some of the building's services can use the UbiSense system to identify and authorize disabled users waiting to enter a door and to open the door through the door-lock mechanisms.

The door-lock server is thus an important CCI service. It's also a single point of failure, so it can be the target of various attacks. Therefore, because all messages transmitted between the swipe-card detector and the door-lock server are in plain text, the HESTIA layer must provide the required encryption support to protect the confidentiality of the information exchanged and to preserve user privacy. Additionally, access to the door-lock service itself must be fault tolerant. Moreover, HESTIA must balance the load dynamically when requests become a bottleneck to server access. Finally, to prevent denial-of-service (DoS) attacks, no client in this system should know where the server is located.

Architecture

Figure 1 gives an overview of HESTIA integrated in our smart building. HESTIA partitions the network into a *service domain* and an *application domain*. Network-level firewalls and network address translation (NAT) boxes enforce this partitioning at the HESTIA layer's perimeter.

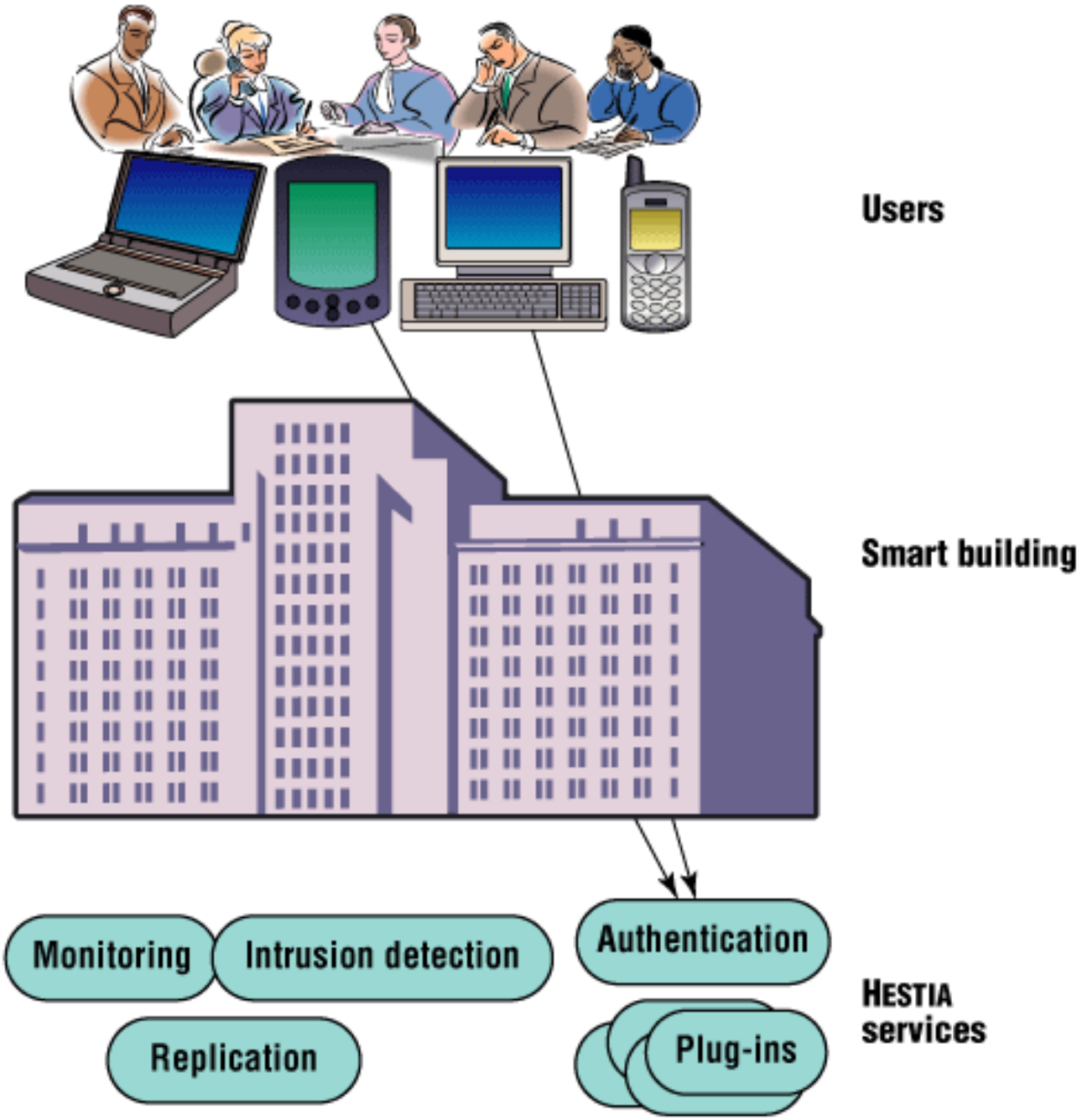

Figure 1. Overview of HESTIA integrated in our smart building.

Service domain

The service domain provides users and services with a set of mechanisms for fault tolerance, quality of service, and privacy through a network of *middleboxes*. A middlebox is a node containing an instantiation of security, privacy, and load-balancing mechanisms. Middleboxes provide a programmable, distributed-object interface so that service owners and administrators can exercise fine-grained control over network service use, deployment, management, and control. Middleboxes also provide appropriate filtering mechanisms to enforce customizable anonymity, confidentiality, and privacy concerns. The middleboxes network acts as a cluster of reconfigurable computing and communication nodes to integrate security, privacy, and reliability in the service domain.

Because a smart-building environment is dynamic and context aware, it must accommodate many factors, such as context information, role hierarchies, security policies, building plans, risk factors, and service dependencies. Managing all these factors individually is difficult. Therefore, the service domain contains a knowledge base and an inference engine. The knowledge base is a repository for context information, role hierarchies, security policies, and so on. The inference engine uses information in the knowledge base to compose requirements. It also composes mechanisms to satisfy those requirements, generates service graphs, and applies the necessary policy decisions (access control, privacy, and so on) to the service graphs. Additionally, the inference engine must generate auditing mechanisms, and intrusion detection monitoring for detecting malicious activity. Figure 2 illustrates how HESTIA employs user and service policies to derive the appropriate security mechanisms.

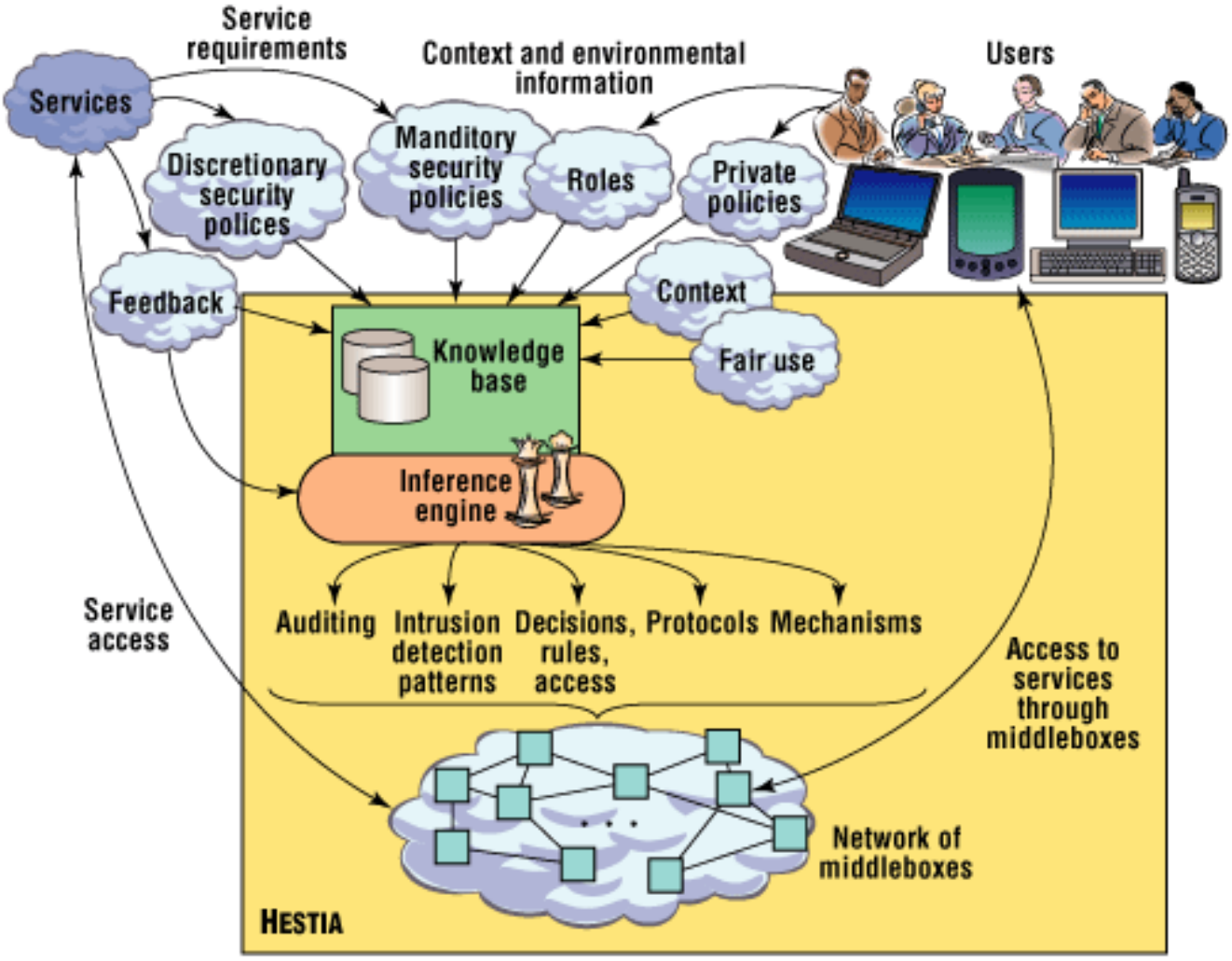

Figure 2. Deriving security mechanisms from requirements and policies in HESTIA .

HESTIA extensively uses role-based access control, typically tying resource access to user roles and contextual information. The inference engine also contains a secure feedback component. When HESTIA denies a user access to a resource, the system must provide that user with useful feedback on how to gain access—for example, whether to simply return later or to obtain additional credentials. However, unconstrained feedback might reveal too much information about the system's policies. Therefore, we use Know,[2] a mechanism for providing useful feedback while honoring the privacy of the system's policies. Such a component is important in a ubiquitous environment, where many users can interact with a plethora of devices.

Application domain

In the application domain, the HESTIA discovery protocol exposes building services as application service interfaces of middlebox proxies mapped across routers and firewalls (see Figure 2). The application domain gives users—on the basis of their roles, for example—an abstraction of what services are available and a set of interfaces to interact with the service domain. The application domain also gives users discovery and lookup services.

Application domain services (discovery, lookup, and so on) are also subject to security constraints, so users can view and access them according to their roles. For example, access to the door-lock service in our smart building is available through the service domain via a proxy that runs on a middlebox. This service then appears in discovery and lookup services in the application domain. After discovering the door-lock service proxy's location, users can interact with the proxy in the service domain. Other services in the Siebel Center will include sensor-controlled heating and cooling, sprinkler systems, a location service that tracks user movements while respecting their privacy, and so on.

Middlebox service protection layers

All the services in our smart building need suitable protection mechanisms. The network of middleboxes provides such mechanisms, including load balancing, fault tolerance, anonymity, quality, and secure services. A specific layer in our middleware provides each of these services.

Load balancing

This layer distributes service processes among participating middleboxes according to some probability distribution function. The load-balancing service's main goal is to ensure that no middlebox becomes overloaded. The load balancer gets a resource requirement profile for each service that the system is to host. This information should include CPU, disk space, and bandwidth requirements. At this layer, process migration occurs when it's necessary to distribute the middleboxes' load. Service migration must occur securely. For example, a service such as Kerberos that stores private keys might require a secure transfer to another middlebox. Furthermore, a mobile Internet Protocol approach can maintain access to migrated proxies for connected users. The middlebox that the proxy migrates from forwards messages to the proxy's new home.

Addressing. When a middlebox hosts a service, the load balancer receives a pseudonym for

that service. It then creates an object reference for it. Next, the load balancer registers the object reference and the pseudonym with the name space. To resolve a service, a user presents the pseudonym to the name server, which returns the corresponding object reference, or handle, to the user.

Denial-of-service protection. Through load distribution, middleboxes can handle larger aggregate loads, thus increasing the network's DoS resilience. Additionally, services can specify usage constraints as part of their requirements. This layer maintains the aggregate load for a service's proxies and keeps it within the maximum allowable load, thereby preventing DoS attacks on the actual service. For example, because users heavily use the door-lock service in our building, its proxies could attract a considerable amount of load. In such cases, the load-balancing layer could migrate proxies to ensure that no middlebox is overloaded. Furthermore, if someone tries to overload the service by making repeated requests for access, the load-balancing layer ensures that the aggregate request bandwidth to the door-lock database remains within acceptable thresholds.

Fault tolerance

Although HESTIA could provide basic fault tolerance for services (through replication, for example), this layer maintains service availability by replicating the proxies for a service. If proxies are attacked (for example, a DoS attack), this layer can instantiate new proxies for the service on the middleboxes. HESTIA deploys services along with replication requirements for their proxies. The system can replicate proxies for quick recovery using active or passive replication mechanisms. Active replication maintains the replica in the same state as the original proxy. Passive replication uses periodic checkpointing of consistent states. For example, the door-lock service can deploy several proxies through the load-balancing layer. This makes this service accessible through several proxies, increasing its availability and fault tolerance. If a proxy fails, its active (or passive) replica can act as a substitute. The fault tolerance service could require more sophisticated fault-tolerant middlebox access schemes.

Anonymity

This layer obfuscates the identity and location of the client, the middleboxes, or both. The Onion Routing,[3] Crowds,[4] and Mist[5] routing concepts at this layer provide anonymity. In our solution, which we based on Mist, users or services establish routes through middleboxes while keeping their locations private. Handles maintain forward and reverse paths for packets and establish a route though middleboxes. Each router knows only the previous and next hops. These routes

eventually end at some middlebox, which serves as a point of communication for the entity. Mist calls this middlebox the *lighthouse* for that entity. Mist directs traffic for an entity to its lighthouse, which then forwards the data down the established path. Because communication occurs through lighthouses, an entity's actual location remains hidden. Hence, the anonymity layer services decouple the users' identity from their location, giving them location privacy or location anonymity. Users can optionally choose to not reveal their identities, thereby achieving location and identity anonymity.

The anonymity layer provides anonymity services for both users and services.

Anonymity for users. Because users must constantly interact with services in ubiquitous computing environments, a system could feasibly track a user's movements. For example, a service administrator could mine service logs and infer user locations. Providing anonymity to users ensures that their locations remain secret while still letting them interact with services through their lighthouses.

Anonymity for services. Although it's possible to deploy critical services through middleboxes, we prefer to keep these service locations private. This prevents malicious parties and even insiders from identifying the specific machines on which to mount attacks. Such attacks could cripple the targeted service. Hence the middleboxes' anonymity layer provides this functionality. For example, the ubiquitous environment might store extensive information pertaining to users. This information might include personal data and usage statistics of users. Storing such data threatens user privacy. Another way to make data storage anonymous is to keep it at undisclosed locations. Although data is accessible through the service's lighthouse, that data's location remains unknown. This prevents malicious parties from denying access to data, because once again they don't know on which machines to mount their attacks. Furthermore, the fault tolerance layer can replicate data to increase its availability and make it more resilient to DoS attacks.

Thus, continuing with our example, the anonymity layer could keep the location of the door-lock service secret. This is important because the door-lock service is crucial to a building's operation, and exposing this service's location would make the building vulnerable to attackers.

Quality

This layer gives users an interface for specifying the level of quality they wish to receive. Users can specify quality based on data transmission metrics or quality of service (QoS)—for example, delay, bandwidth, or packet loss rates. Users can also specify quality based on a client's

required level of security or quality of protection (QoP) in terms of confidentiality, integrity, anonymity, and so on. For example, services can request higher grade encryption and authentication for users connecting with the service. Users can request routes from the QoS layer with QoP requirements (such as bandwidth distribution), which are immune to traffic analysis. The door-lock service, for example, can demand a certain level of authentication (QoP) for access to the locks. It can also request QoS parameters such as low latency for quick response times.

Secure services

This layer uses middleboxes to provide access control, confidentiality, and integrity for services. Services can upload access policies into this layer. These access policies, along with system policies, could then control access to the services. The door-lock service could request the use of encryption and signatures to ensure the privacy and integrity of messages to the service.

Functionally, the middlebox network in Figure 2 acts as a cluster of reconfigurable computing and communication nodes. Clients can access services in our system only through the HESTIA layer. Servers create and install proxies on the middlebox network either proactively or reactively in response to client requests. These proxies provide a restricted view of service interfaces, corresponding to the requesting clients' authorizations. They can also implement the server's logic and dynamically offload the server's computations. The proxies also act as filters to enforce confidentiality and privacy concerns. In addition, the system can replicate these proxies for load balancing and fault tolerance, or it can form a customized network of them to provide QoS or virtual-private-network-style routing. The policy specifications drive the instantiation of proxy objects on middleboxes, and the inference engine can compose different functional and nonfunctional requirements to create customized service objects on demand. By defining the interfaces and rules for composition correctly, we can let the inference engine compose different protocols and functions using standard object composition techniques to provide differentiated services.

Our architecture's proxy objects are not persistent, and they maintain little global state information. Ideally, we'd like to migrate and restart proxies on any middlebox in our network with little overhead, but proxies can't be migrated to heavily loaded middleboxes. Therefore, we carefully designed the proxies to implement soft-state protocols. Even when a proxy is attacked and compromised, the damage is contained, and a new proxy can easily restart on a different middlebox. Proxies have little knowledge of other services and systems. This design aspect lets us build truly survivable network services—which can continue to provide service guarantees and degrade gracefully under attack—atop existing best-effort network service models.

Implementation

We've implemented the network of middleboxes as an overlay network over the Transmission Control Protocol and Internet Protocol. We choose Prolog to act as an inference engine. We use CORBA as the major backbone for communication in our distributed system (www.corba.org). CORBA offers several services that are instrumental in implementing some of the needed functionality, including support for atomic transaction, persistent objects, and platform independence. Many CORBA implementations are heavyweight and might not be appropriate for implementing an overlay network. So, we're experimenting with the Universally Interoperable Core (UIC), which provides a lightweight, high-performance CORBA implementation.[6] Every layer in our architecture has a broker that provides a CORBA Interface Definition Language interface so that services and users can access that layer's functionality.

Conclusion

HESTIA will open new frontiers in the design, development, and deployment of trusted CCI for buildings. The unique middlebox architecture's programmable, distributed, object-oriented framework is inherently survivable. HESTIA will significantly affect the adoption of CCI in buildings as diverse as hospitals, airports, offices, laboratories, and power plants. It will demonstrate that it's possible to deploy smart-building services without compromising privacy or critical safety, security, or survivability properties. It will also encourage new industries to build applications that exploit smart buildings. These might include ubiquitous services for users with disabilities; safety and rescue operations; sophisticated antitheft approaches; personal safety applications; communication, collaboration, and education services; and passenger information subsystems.

References

1. J. Al-Muhtadi , S. Chetan, and A. Ranganathan, , "Super Spaces: A Middleware for Large-Scale Pervasive Computing Environments,"*Proc. 2nd IEEE Ann. Conf. Pervasive Computing*

and Comm. Workshops (PerCom 04), IEEE CS Press, 2004,pp. 198–202; http://csdl.computer.org/comp/proceedings/percomw/2004/2106/00/21060198abs.htm.

2. A. Kapadia , G. Sampemane, and R.H. Campbell, , "Know Why Your Access Was Denied: Regulating Feedback for Usable Security," to be published in *Proc. 11th ACM Conf. Computer and Communications Security,* ACM Press, 2004.

3. M.G. Reed , P.F. Syverson, and D.M. Goldschlag, , "Anonymous Connections and Onion Routing,"*IEEE J. Selected Areas in Communication,* vol. 16, no. 4, 1998,pp. 482–494; http://csdl.computer.org/comp/proceedings/sp/1997/7828/00/78280044abs.htm.

4. M. Reiter and A.D. Rubin, , "Crowds: Anonymity for Web Transactions,"*ACM Trans. Information and System Security* (TISSEC), vol. 1, no. 1, 1998,pp. 66–92.

5. J. Al-Muhtadi , et al., "Routing through the Mist: Privacy Preserving Communication in Ubiquitous Computing Environments,"*Proc. 22nd Int'l Conf. Distributed Computing Systems* (ICDCS 02), IEEE CS Press, 2002,pp. 74–83; http://csdl.computer.org/comp/proceedings/icdcs/2002/1585/00/15850074abs.htm.

6. M. Román , R.H. Campbell, and F. Kon, , "Reflective Middleware: From Your Desk to Your Hand,"*IEEE Distributed Systems Online,* vol. 2, no. 5, 2001; http://dsonline.computer.org/0105/features/rom0105_print.htm.

Raquel Hill is a postdoctoral research associate with a joint appointment in the Department of Computer Science and the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign. Her research interests include security for wired and wireless infrastructures, resource allocation protocols, and security requirements and policies. She received her PhD in computer science from Harvard University. She is a member of the IEEE. Contact her at the Siebel Center for Computer Science, 201 N. Goodwin, Urbana, IL 61801; rlhill@uiuc.edu.

Jalal Al-Muhtadi is a PhD candidate and graduate research assistant working on the Gaia project at the University of Illinois at Urbana-Champaign. His research interests include middleware and infrastructure security and privacy issues. He received his MS in computer science from the University of Illinois at Urbana-Champaign. Contact him at the Siebel Center for Computer Science, 201 N. Goodwin, Urbana, IL 61801; almuhtad@cs.uiuc.edu.

Roy Campbell is a professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His research interests include active spaces for ubiquitous computing, authorization for sensor networks, simulations of network security, and the design of peer-to-peer distributed operating systems. He received his PhD in computer science from the University of Newcastle upon Tyne. He is a member of the IEEE and the ACM. Contact him at the Siebel Center for Computer Science, 201 N. Goodwin, Urbana, IL 61801; rhc@uiuc.edu.

Apu Kapadia is a PhD candidate in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His research interests include high-performance networking and security, and he is currently investigating secure routing protocols, active queue management schemes, and privacy in ubiquitous environments. He received his MS in computer science from the University of Illinois at Urbana-Champaign. He is a member of the IEEE and the ACM. Contact him at the Siebel Center for Computer Science, 201 N. Goodwin, Urbana, IL 61801; akapadia@uiuc.edu.

Prasad Naldurg is a postdoctoral research scholar and visiting lecturer in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His research interests include systems and network security, and applications of formal methods and cryptography to related problems. He received his PhD in computer science from the University of Illinois at Urbana-Champaign. Contact him at the Siebel Center for Computer Science, 201 N. Goodwin, Urbana, IL 61801; naldurg@cs.uiuc.edu.

Anand Ranganathan is a doctoral candidate and research assistant working on the Gaia project in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His research interests include pervasive computing, middleware, mobile computing, the Semantic Web, and automated reasoning and learning. He received his BTech in computer science from the Indian Institute of Technology in Madras. Contact him at the Siebel Center for Computer Science, 201 N. Goodwin, Urbana, IL 61801; ranganat@uiuc.edu.

Related Work

Several middleware platforms for distributed programming introduce metaprogramming extensions that provide functionality similar to middleboxes. CORBA provides portable *interceptors*,[1] which let developers extend and control the object request broker's behavior. However, interceptors have limited capabilities and can reside only on the client and server sides, whereas we envision middleboxes to run on intermediate machines.

Java RMI introduced remote-method-invocation *stubs* for distributed objects.[2] A stub is a remote reference to a distributed object that merely forwards all method calls to the target object.

Jini,[3] and some CORBA implementations such as TAO (*T*he *A*daptive Communication Environment *O*bject Request Broker),[4] support smart proxies. A smart proxy resembles a stub, but it can provide additional features such as results caching, failover, and custom protocols to communicate back to the target object.

Microsoft .NET Remoting supports constructs (RealProxy and TransparentProxy) similar to a smart proxy.[5] Middleboxes also provide functionality similar to smart proxies. However, unlike typical smart-proxy frameworks, which create one proxy per client, HESTIA supports many-to-many relationships between clients and middleboxes on the one hand, and between services and middleboxes on the other. For example, a middlebox can forward a client's request to one of many active services to achieve load balancing. For fault tolerance, HESTIA lets users dynamically create middleboxes on the fly, and it supports dynamic bindings between middleboxes and services as they become available. Furthermore, several mediators might be necessary to provide sufficient services. For example, anonymity might require communication channels to traverse several middleboxes in sequence to provide a better level of concealment. HESTIA 's network of middleboxes, along with its ability to dynamically replicate proxies, fulfills this need for multiple mediators.

References

1. "CORBA 3.03 Specification," OMG, 2004; www.omg.org/technology/documents/formal/corba_2.htm.
2. "Core Java: Java Remote Method Invocation (Java RMI)," Sun Microsystems; http://java.sun.com/products/jdk/rmi.
3. J. Waldo, , "The Jini Architecture for Network-Centric Computing,"*Comm. ACM,* vol. 42, no. 7, 1999,pp. 76–82.
4. D.C. Schmidt , D.L. Levine, and S. Mungee, , "The Design and Performance of Real-Time Object Request Brokers,"*Computer Comm.,* vol. 21, no. 4, 1998,pp. 294–324.
5. P. Obermeyer and J. Hawkins, , "Microsoft .NET Remoting: A Technical Overview," Microsoft, July 2001; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/hawkremoting.asp.