Journal of Advances in Science and Technology

Vol. IV, No. VII, November-2012, ISSN 2230-9659

PROXY FRAMEWORKS FOR MOBILE COMPUTING AND WIRELESS COMMUNICATION

www.ignited.in

Journal of Advances in Science and Technology
Vol. IV, No. VII, November-2012, ISSN 2230-9659

Proxy Frameworks for Mobile Computing and Wireless Communication

Jaspal Singh

Research Scholar, Singhania University, Rajasthan India

ABSTRACT:-The use of proxies is commonplace in today's networks, where they are used for a huge variety of network services. A proxy is an intermediary placed in the path between a server and its clients. Proxies are used for saving network bandwidth, reducing access latency and coping with network and device heterogeneity.

----------♦----------

OVERVIEW

In the specific case of mobile computing and wireless communication, proxies are mainly used to overcome the three major problems of these networks: throughput and latency differences between the wired and the wireless links, host mobility, and limited resources of the mobile hosts (MH). Although proxies may be used also for implementing specific services in ad hoc mobile networks, usually they are used in infra-structured mobile networks, since their functions commonly place high demands on both processing and memory. Thus, in tins chapter we will mainly discuss proxy-based architectures for infra-structured mobile networks.

In most cases, proxies act as protocol translators, caches and content adapters for clients with network or device constraints and are placed on, or close to, the border between the wired and the wireless networks, such as at the wireless Access Points (AP) (also called Base Stations or Mobility Support Stations). Besides these canonical functions, however, proxies can perform a wide range of other complex tasks on behalf of the mobile clients, such as handover, session or consistency management, personalization, authentication, check pointing, service/resource discovery, and others.

The major advantages of using a proxy-based architecture for serving mobile clients, when compared to an end-to-end approach, cue the following: (a) all mobility- and wireless-dependent transformations (translation, transcoding) can be assigned to the proxy and need not be handled by the servers, allowing legacy services to be directly used for mobile access; (b) all processing required for protocol and content transformations is distributed to other nodes where they are required, avoiding an overload at the servers; (c) placing proxies at (or close to) a node with the wireless interface enables more agile and accurate monitoring of the wireless link quality, detection of MH disconnections, as well as better selection of the required adaptation; and finally (d) transformations at any communication layer can be implemented, and are more easily adapted/customized according to the specific capabilities of the wireless links.

As expected, there is a huge amount of work on proxy-based middleware for mobile and wireless computing, each solving the problems specific to some sort of service or application, such as Web access, multimedia streaming, database access, etc. Many authors use the teems gateway, intermediary or agent instead of proxy, and although there might be some subtle differences in their meanings, we will use these terms interchangeably and use the general definition of a proxy as being an entity that intercepts communication or performs some service on behalf of some mobile client.

INTRODUCTION

Attaining the goals of ubiquitous and pervasive computing [6, 2] is becoming more and more feasible as the number of computing devices in the world increases rapidly. However, there are still significant hurdles to overcome when integrating wearable and embedded devices into a ubiquitous computing environment. These hurdles include designing devices smart enough to collaborate with each other, increasing ease-of-use, and enabling enhanced connectivity between the different devices.

When connectivity is high, the security of the system is a key factor. Devices must only allow access to authorized users and must also keep the communication secure when transmitting or receiving personal or private information.

Implementing typical forms of secure, private communication using a public-key infrastructure on all devices is difficult because the necessary cryptographic algorithms are CPU-intensive. A common public-key cryptographic algorithm such as RSA using 1024-bit keys takes 43ms to sign and 0.6ms to verify on a 200MHz Intel Pentium Pro (a 32-bit processor) [30]. Some devices may have 8-bit

microcontrollers running at 1-4 MHz, so public-key cryptography on the device itself may not be an option. Nevertheless, public-key based communication between devices over a network is still desirable.

Since proxies are primarily used to bridge and smooth the differences between networks and devices, and to perform application-specific adaptations, then functions are designed according to:

- The different characteristics of the wired and wireless networks which are to be bridged, such as throughput latency, reliability, probability of disconnection, etc.

- The specific characteristics of the mobile host, such as: display size, user input/output mechanisms, processing capacity, size of RAM and persistent memory, limited energy supply, etc.

- The application type and its specific requirements/ such as fast response time, low network latency, reliable communication, mobility or disconnection transparency, cache coherence, etc.

Tirese aspects give an idea of the wide range of adaptation and management functions that can possibly be assigned to proxies. They may handle communication protocol issues, data transmission and encoding, device-specific customizations, handover and mobility management, security and authentication, recovery from disconnection, etc.

hi spite of the huge diversity of proxy-centered architectures and proposals we have identified two orthogonal forms of classifying and comparing all proxy-based approaches. Tire first dimension takes into account some general characteristics of the proxy-based architecture, while the second dimension focuses on the tasks, i.e. functionalities/ assigned to the proxies. These two classifications will be further detailed in sections 3 and 4, respectively.

Obviously, there are also other possible criteria for classifying proxy-based approaches, hi particular, Dikaiakos [IS] has written a very interesting survey about proxy-based infrastructures specifically for the Web. He proposes a classification of proxy approaches in three dimensions: system architecture, functionality and interactions. Regarding system architecture, he distinguishes between centralized and distributed architectures/ options for proxy placement, and proxy configurability/programniability. Concerning functionality, he proposes six broad categories, which are consistent with our task categorization. Finally, with interactions the author considers whether the proxy supports synchronous or asynchronous communication. In addition, the article also compares eight proxy-based architectures and frameworks for the Web in deep detail. Hence, we recommend it as complementary reading to the interested reader.

To allow the architecture to use a public-key security model on the network while keeping the devices themselves simple, we create a software proxy for each device. All objects in the system, e.g., appliances, wearable gadgets, software agents, and users have associated trusted software proxies that either run on an embedded processor on the appliance, or on a trusted computer. In the case of the proxy running 011 an embedded processor 011 the appliance, we assume that device to proxy communication is inherently secure.[1] If the device has minimal computational power,[2] and communicates to its proxy through a wired or wireless network, we force the communication to adhere to a device- to-proxy protocol. Proxies communicate with each other using a secure proxy-to-proxy protocol based on SPKI/SDSI (Simple Public Key Infrastructure / Simple Distributed Security Infrastructure). Having two different protocols allows us to run a computationally-inexpensive security protocol on impoverished devices, and a sophisticated protocol for resource authentication and communication 011 more powerful devices. We describe both protocols in this section.

Using the ideas described above, we have constructed a prototype automation system which allows for secure, yet efficient, access to networked, mobile devices. In this system, each user wears a badge called a I<21 which identifies the user and is location-aware: it "knows" the wearer's location within a building. User identity and location information is securely transmitted to the user's software proxy using the device-to-proxy protocol.

Devices themselves may be mobile and may change locations. Attribute search over all controllable devices can be performed to find the nearest device, or the most appropriate device under some metric.[3]

By exploiting SPKI/SDSI, security is not compromised as new users and devices enter the system, or when users and devices leave the system. We believe that the use of two different protocols, and the use of the SPKI/SDSI framework in the proxy-to-proxy protocol has resulted in a secure, scalable, efficient, and easy-to-maintain automation system.

To allow our architecture to use a public-key security model on the network while keeping the resources themselves simple, we create a software proxy for each resource. All objects in the system, e.g., appliances, wearable gadgets, software agents, and users have associated trusted software proxies that either run on an embedded processor oil the appliance, or on a trusted computer, hi the case of the proxy running oil an embedded processor on the appliance, we assume that resource-proxy communication is inherently secure. [2] If the resource has minimal computational power.' and communicates to its proxy through a wired or

wireless network, we force the communication to adhere to a resource-proxy protocol. Proxies communicate with each other using a secure proxy-proxy protocol based on SPKI/SDSI (Simple Public Key Infrastructure / Simple Distributed Security Infrastructure). With two different protocols, we are allowed to run a computationally-inexpensive security protocol on impoverished resources, and a sophisticated protocol for authorization and communication oil more powerful resources.

The resource-proxy protocol varies for different types of resources. In particular, we consider lightweight resources with low-bandwidth wireless network connections and slow CPUs, and heavyweight resources with higlier-bandwiclth connections and faster CPUs. We assume that heavyweight resources are capable of running proxy software locally (i.e., the proxy for a printer coulcl run on the printer's CPU). With a local proxy, a sophisticated protocol for secure resource-proxy communication is unnecessary, assuming critical parts of the resource are tamper resistant. For lightweight resources, the proxy must run elsewhere. An example of a resource-proxy protocol for a lightweight resource is one in which the resource and its proxy share symmetric keys with which they encrypt and authenticate their communication.

For the proxy-proxy protocol, we have adopted a client-server architecture. When a particular principal, acting on behalf of a resource or user, makes a request via one proxy to a resource represented by another proxy, the first proxy acts like a client, and the second as a server. Services on the server are either public or protected by SPKI/SDSI access control lists (ACLs). To gain access to a service protected by an ACL, a client must send a signed copy of its request, and a chain of SPKI/SDSI certificates demonstrating that it is a member of a group in an entry on the ACL.

Hie proxy-proxy protocol layers SPKI/SDSI access control over an application protocol, which in turn is layered over a key-exchange protocol. This allows us to deal with a variety of application protocols which may be implemented across wired or wireless links in a heterogeneous network.

Using the SPKI/SDSI framework, ACLs associated with resources can be created once and rarely need to be modified. User access rights are modified by issuing certificates based on group membership; rights can be revoked through a variety of mechanisms such as online checks, hi addition, SPKI/SDSI features an elegant model for delegation of authority, allowing for the partitioning of responsibilities. The user maintaining an ACL on a resource could, but need not be, the same user that authorizes others to access the resource. This significantly eases the burden of system administration.

PROXY FRAMEWORKS

As proxies have been used as a general approach for handling dynamic adaptation, several efforts have been made to develop generic proxy architectures, or proxy frameworks, that can be customized or extended to solve a particular problem. An example of such an effort is IETF's Open Pluggable Edge Services , winch proposes a reference architecture for web proxies, addressing issues as security, distribution and dynamic configuration.

hi this section we describe common mechanisms used in proxy frameworks and compare well-known systems, such as TACC, RAPID ware, Mobi ware, MARCH, Web Intermediaries, and MOCA Proxy Framework. The RAPID ware project has proposed adaptive proxy services for multimedia streams. Mobiware is a QoS-aware middleware platform for multimedia applications which also provides support for handoff control. Web Intermediaries (WEI) have been developed at IBM, for HTTP-based adaptations, such as personalizing contents, transcoding, or caching. MARCH, TACC [S] and MOCA's Proxy Framework are general-purpose content adaptation frameworks.

Most proxy frameworks provide general-purpose solutions for the following four main issues: (a) implementation and composition of adaptation modules, called adapters; (b) description of the conditions in which the adapters should be applied; (c) monitoring of the context, such as the mobile device's profile, the application's state and the communication bandwidth; and (d) the loading of adapters. In the remainder of this section we will discuss these features in more detail. A complementary discussion about proxy frameworks can be found in [IS].

Adapter Development **:** The main customization point of a proxy framework is the adapter, a module responsible for implementing a transcoding function of a message or its content. A proxy (i.e. an instance derived from the framework) may use several adapter instances for implementing specific adaptations required for different clients or contexts. Taking into account the client's current context, a proxy determines at runtime which adapter should be used for a message or data content. In some situations (e.g. contexts), more than one adapter can be selected for transcoding a message. Therefore, some frameworks support the definition of priorities, ordering, and/or composition of adapters.

Most proxy frameworks are designed using extensibility mechanisms and component-based approaches to support the development and composition of adapters, as well as their loading into a proxy, hi some frameworks, such as WEI, RAPID ware and MARCH, adapters can be developed as independent and composable components that are stored in adapter repositories or libraries and

deployed in proxies. Some frameworks provide classes of special-purpose adapters. For example, Mobi ware supports two kinds of adapters: Active Media Filters, for media content adaptation, and Adaptive FEC Filters, for error correction. RAPIDware also provides some FEC filters, in order to improve the ability of the audio/video stream to tolerate errors in a wireless environment. The TACC model supports adapters for transformation (content adaptation), aggregation (information collecting), caching and customization.

Adapter Selection : The decision of winch adapters to use and when to use them is an extensible characteristic of proxy frameworks, which can be defined in two ways: via programmable interfaces or via rule-based configuration. An example of the first way can be found in Mobiware, where the application requirements (utility function) and the adaptations to be applied (adaptation policy) must be programmed using a framework-provided API. When a rule-based configuration is supported, the developer must define rules winch contain the trigger conditions, described in terms of the client and network states (i.e. context); the adaptations to be executed; and sometimes also a priority of the rule. Usually, the rules are described manually via an XML (or RuleML) file, hi MARCH the selection process evaluates the rule set during session setup, and produces as the result a set of adapters to use (chain of adapters), hi MoCA's Proxy Framework and WEI, rules are evaluated just before each message is sent to the client.

Rule-based systems are easily configured and less error prone (defining a model) than the ones based on programmable interfaces; besides there is no need to deal with intrinsic details of the framework. Furthermore, only the content provider can decide which adaptation is acceptable under different contexts, and thus, by using rules, may define the sequence of adaptations to apply to data, better controlling their composition, which is a very complex task to automate.

Context Monitoring: The monitoring and gathering of context information (i.e. the client's profile, and conditions of the execution environment, such as available resources, load and energy at the mobile host and the network) are part of the desired functionality of proxy frameworks. The collection of the network state, such as available bandwidth or connectivity, is generally done via a monitoring function or service, as in TACC, MARCH and MoCA Proxy Framework. Information related to the client may be obtained at their startup connection request [1], via a customization database containing profiles [S], or through monitormg of the device's resources [52]. hi most frameworks, context changes are notified through asynchronous events, which must be interpreted and processed by the proxy in order to execute the appropriate action.

Adapter Loading and Execution : According to how adapters are loaded and activated, proxy frameworks can be classified as configurable or dynamic proxies. In a configurable proxy, adapters are defined statically at proxy deployment time. The developer can change the proxy's behavior by using trigger rules that define the order and the context in winch an adapter should be executed. A dynamic proxy supports dynamic and on-demand loading of adapters from an adapter repository, according to the current context.

Two examples of dynamic proxies are RAPIDware and MARCH. RAPIDware provides a composable proxy framework to support the dynamic composition of services by fetching adapters (called filters) from a repository, and instantiating and reconfiguring them dynamically on the proxy in response to the changing needs of mobile clients. MARCH provides a dynamic execution environment for adapters, which facilitates the uploading of proxies on a per-session basis, which may be placed on the server or on mobile devices, hi MARCH, the MAS (Mobile Aware Server) component is in charge of making the decision of which adapters, chosen from the proxy repositories, are to be used and where to execute them.

Ail example or framework for configurable deployment of proxies is Web intermediaries (WBI). At proxy startup, the registered adapters (or plug-ins) are instantiated with the corresponding firing conditions in rules with an associated priority. WEI supports die aggregation of adapters, and the proxy can be placed either on the server or on the client side. Another example is MoCA ProxyFramework, where the adapters are instantiated during proxy initialization, according to the trigger rules (described ni an XML configuration file) specifying the context 111 which the adaptation (or set of adaptations) should be applied. This framework also supports chaining of adapters, use of priorities, and mechanisms for specifying caching policies.

Comparing the two approaches, the dynamic loading of adapters provides more flexibility to the system. However, configurable proxies support verification of a consistent combination/configuration of adapters, hi addition, dynamic (down)loadnig of adapters can be time consuming. Therefore, it is more suited for systems where context changes are not very frequent.

Table presents the cited frameworks, summarizing their main characteristics according to the aspects discussed in tins section and in section.

	RAPIDware	Mobiware	MARCH	TACC	MoCA Framework	WBI
Purpose	Multimedia	Multimedia, QoS	General	General	General	Web Apps.
Level	Middleware	Middleware	Application	Middleware	Middleware	Application
Proxy Placement	server-side	client-side and server-side	server-side and proxy-pair	server-side	server-side	client-side and server-side.
Dynamic Adapter Loading	Yes	Yes	Yes	No	No	No
Adaptation Selection	Programmable	Programmable	Trigger-Rules Configuration	Programmable	Trigger-Rules Configuration	Trigger-Rules Configuration
Functionality	Content Adaptation	Content Adaptation, Handover Mngt	Content Adaptation	Caching, Content Adaptation	Caching, Content Adaptation	Caching, Content Adaptation
Communication	Synchronous	Synchronous	Synchronous	Synchronous	Synchronous, Asynchronous	Synchronous, Asynchronous
Context Awareness	wireless link	wireless link	device & wireless link	wireless link	device & wireless link	-

Table: Comparison table of extensible proxy approaches

Comparing the presented systems, one should notice that all of them support content adaptation, while caching management appears as the second most frequent functionality, and handover management is provided only by Mobiware. Furthermore, there are equal numbers of systems concerning the level (middleware versus application), the capability of dynamic adapter loading, and the form of adaptation selection (programmable versus trigger-rule configuration). Concerning communication capabilities, only MoCA Framework and WEI support asynchronous (publish/ subscribe) communication, which has been recognized as best suited for mobile computing. Context awareness is also supported by most of the frameworks (i.e. except WEI), but only MARCH and MoCA Framework consider also the state of the client's devices. Although it is quite difficult to compare the frameworks, Mobiware seems to be one of the most complete systems in terms of supported functionality, extensibility and architecture.

APPLICATIONS

The system can be used as the framework to build many different types of applications. In this section we will describe an example application that highlights the functionality and privacy that is provided by the wearable communicator.

Mobile Audio : We developed a mobile audio application using the above described system. That is, as a user with a wearable communicator moves from room to room, a single audio stream will follow him or her. always playing from the nearest speakers. The wearable communicator is constantly being polled by its proxy, asking for its location. This information is reported to an automation script that runs on top of the proxy.

When an audio stream is sent to the proxy, the automation script uses a directory server to obtain a list of speakers that are reporting their location in the same area as the wearable communicator. The automation script chooses the closest one and redirects the audio. If. at any time, the location of the wearable communicator changes, and hence, the nearest speakers change, the audio output is again redirected. Figure is an overview of the application.

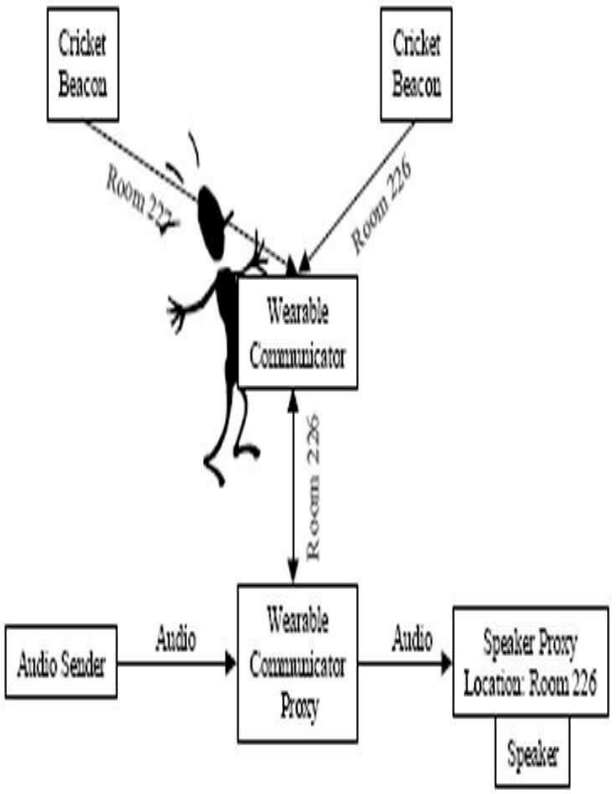

Figure. Audio Example Application

Since the audio is redirected by the proxy (and only the proxy knows the user's location), the user's location is kept private. The proxy could also route other types of information to the user's location such as text messages, or video, while keeping the users actual location private. For other applications, the user could set the wearable communicator's proxy to only give out the location to select people but keep it private from others.

Other Applications : The system can also easily support other applications. For example many customization can be made upon entering a new room, such as turning on the lights, setting the thermostat, and opening the blinds. Another

application would be to customize the desktop of the computer where the user is logged in. The system could also forward phone calls to the phone nearest the user. Or. for privacy, the system could be set up to only print your documents when you are located next to the printer.

DEVICE ARCHITECTURE

The primary- design goal of the architecture is security. That is. the authentication, authorization, and privacy of all communication. An architecture that fulfils this requirement needs an end-to-end security layer, from the user controlling the device to the device itself. In addition, the architecture must be appropriate for the devices being controlled. Enhancing the security of, for example, a wearable camera should not require the addition of expensive processing power. The system must be secure with the addition of, at most, a cheap, simple micro-controller.

Public-key cryptography is ideal for authentication and authorization. Unfortunately, public-key cryptography requires signif cant computational power. A common public- key cryptographic algorithm such as RSA using 1024-bit keys takes 43ms to sign and 0.6ms to verify on a 200MHz. Intel Pentium Pro [13]. This is using a 32-bit processor; some of the devices in this system may have 8-bit microcontrollers running at 1-4 MHz, so public-key cryptography on the device itself is simply not an option.

However, public-key based communication between devices over a network is still desirable. To allow the architecture to use a public-key security model on the network while keeping the devices themselves simple, we create a software proxy for each device which we run on a separate, trusted computer. Between the proxy and the device, we implement a symmetnc-key-based security protocol. The proxy can implement sophisticated access control and authentication algorithms, while the device remains simple. Additionally, it is possible to run many proxies on the same computer, allowing the amortization of their cost, since they may require a signifcant amount of processing power and memory to control access to the device.

Devices : By focusing on impoverished devices, we handle the base case; more complex devices can be built by incorporating more of the proxy software onto the device itself. The devices are most likely controlled by simple 8- or 16-bit micro-controllers running at 1-4 MHz. The devices typically take control commands as input and output simple state values. For example, a radio has simple input variables such as on/off, tuning the station, and adjusting the volume. It outputs state such as the current station and volume level.

Devices also need a method for communicating with their proxies. A device and proxy can communicate using wireless methods such as radio frequency (RF) or mfrared. or they could use a wired solution like Ethernet. Regardless of the medium, a reliable communication protocol is required.

Proxies : The proxy is software that runs on a network-visible computer. The proxy 's primary function is to make access- control decisions on behalf of the device it represents. It may also perform secondary functions such as running scripted actions on behalf of the device and interfacing with a directory service.

The proxy can implement computationally expensive security algorithms since it runs on a computer that has sig- nifcantly more processing capabilities than the device. The proxy can also store large access control lists that would not £t in the device's memory. It uses these mechanisms to act as a guardian; the proxy authenticates users and only allows those with valid permissions to control the device.

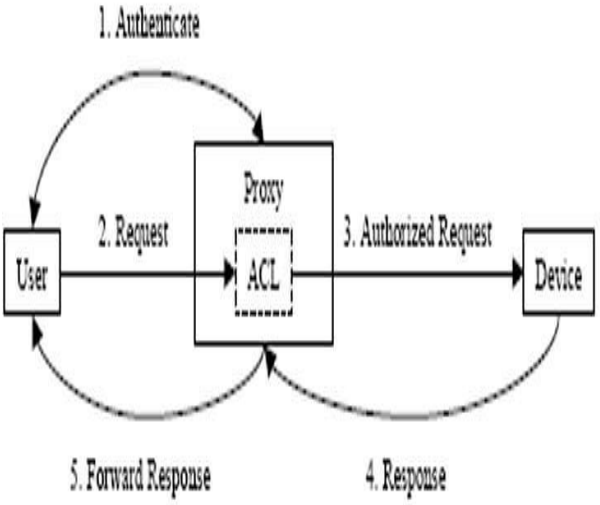

Figure. Security model

Security Model : The proxy and device share a secret key. Tins secret key allows them to communicate using symmetric-key authentication and encryption. Symmetric-key operations take much less processmg power than public-key. so the device can do this computation with a small micro-controller.

All communication passes through the proxy, so it authenticates and then routes communication from the user to the device. The sow of communication is shown in Figure with each step described below.

- The proxy and user authenticate each other. They also set up a secure communication channel.

- The user sends his or her request to the proxy.
- The proxy checks its access control list (ACL) to verify the user is allowed to perform the specif ed request. If this check succeeds, the proxy forwards the request on to the device. Otherwise, the proxy responds with an error message.
- The device performs the requested action and sends a response back to the proxy.
- The proxy forwards the response back to the user.

Device Initialization: When a device is initialized it must be assigned a proxy and it must obtain a secret key that is shared with the proxy. This is done by physically touching the device to the computer that will run the proxy. When the device is touched to the computer, a proxy is created and the proxy then generates a random secret key that it shares with the device. This initialization is straightforward and easy for the user who is initializing the device. The user does not need to perform any manual con£guration.

REFERENCES

[1] 16/32-bit lpc2000 family. http: //www.nxp.com /products /microcontrollers/32bit/index.html.

[2] ARM extended trace macrocell (etm) technical reference guide. http://www.arm.com/documentation/TraceDebug.

[3] ARM's coresight on-chip debug and trace technology. http://www.arm.com/products/solutions/CoreSight.html.

[4] Armulator, ARM. http://www.arm.com /support/ARMulator.html.

[5] Device file system guide. http://www.gentoo.org/doc/en/devfs guide.xml.

[6] exportfs, srvfs - network file server from plan9 man pages. http://plan9.belllabs.com/magic/man2html/4/exportfs.

[7] Features of the msp430 bootstrap loader (rev. d). http://focus.ti.com/lit/an/slaa089d/slaa089d.pdf.

[8] Freescale, MPC565 user's manual, 2002.

[9] Introduction to on-board programming with intel flash memory. http://www.intel.com/design/flcomp/applnots/29217902 .pdf.

[10] Iso 13239 : High-level data link control protocol.

[11] Msp430 : Ultra low power mcu from texas intruments. http://www.ti.com/msp430.

[12] National ecological observatory network. http://www.neoninc.org.

[13] OCP-IP: Open Chip Protocol International Partnership. http://www.ocpip.org.

[14] Providing asynchronous file i/o for the plan 9 operating system. http://pdos.csail.mit.edu/papers/plan9:jmhickey-meng.pdf.

[15] Simple Object Access Protocol (SOAP). http://www.w3.org/TR/soap.

[16] Simulavr: an AVR simulator. http://savannah.nongnu.org .

[17] The two percent solution. http://www.embedded.com/story/OEG20021217S0039.

[18] What processor is in your product? http://www.embedded.com/columns/showArticle.jhtml ?articleID=193101174.

[19] Emstar: A software environment for developing and deploying wireless sensor networks. In Proceedings of the USENIX 2004 Annual Technical Conference, 2004.

[20] The Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface, 2004. http://www.nexus5001.org.

[21] Guest editorial: Concurrent hardware and software design for multiprocessor SoC. Trans. On Embedded Computing Sys., 5(2):259–262, 2006.

[22] D. E. L. G. M. H. A. Cerpa, J. Elson and J. Zhao. Habitat monitoring: Application driver for wireless communications technology. In Proceedings of the 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, April 2001, 2001.

[23] K. M.-M. A. Mayer, H. Siebert. Debug support, calibration and emulation for multiple processor and powertrain control socs. IEEE Trans. Comput., 55(2):174–184, 2006.

[24] C. G. A. S. Tanenbaum and B. Crispo. Taking sensor networks from the lab to the jungle. IEEE Computer Magazine, 39(8):98–100, 2006.

[25] D. F. Bacon. Realtime garbage collection. Queue, 5(1):40–49, 2007.

[26] T. W. Bart Vermeulen and S. Bakker. Ieee 1149.1-compliant access architecture for multiple core debug on digital system chips. In Proceedings of the International Test Conference, 2002.

[27] K. A. Bartlett, R. A. Scantlebury, and P. T.Wilkinson. A note on reliable full-duplex transmission over half-duplex links. Commun. ACM, 12(5):260–261, 1969.

[28] S. Bhattacharya, J. Darringer, D. Ostapko, and Y. Shin. A mask reuse methodology for reducing system-on-a-chip cost. In ISQED '05: Proceedings of the 6th International Symposium on Quality of Electronic Design, pages 482–487, Washington, DC, USA, 2005. IEEE Computer Society.

[29] G. Biegel and V. Cahill. A framework for developing mobile, context-aware applications. In 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom 2004, March 2004.

[30] Bluetooth.com : The official Bluetooth Technology Website.http://www.bluetooth.com/bluetooth/.