A Design and Implementation of Active Network Socket Programming

K.L. Eddie Law, Roy Leung
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
Toronto, Canada
eddie@comm.toronto.edu, roy.leung@utoronto.ca

***Abstract*—The concept of programmable nodes and active networks introduces programmability into communication networks. Code and data can be sent and modified on their ways to destinations. Recently, various research groups have designed and implemented their own design platforms. Each design has its own benefits and drawbacks. Moreover, there exists an interoperability problem among platforms. As a result, we introduce a concept that is similar to the network socket programming. We intentionally establish a set of simple interfaces for programming active applications. This set of interfaces, known as Active Network Socket Programming (ANSP), will be working on top of all other execution environments in future. Therefore, the ANSP offers a concept that is similar to "write once, run everywhere." It is an open programming model that active applications can work on all execution environments. It solves the heterogeneity within active networks. This is especially useful when active applications need to access all regions within a heterogeneous network to deploy special service at critical points or to monitor the performance of the entire networks. Instead of introducing a new platform, our approach provides a thin, transparent layer on top of existing environments that can be easily installed for all active applications.**

Keywords-active networks; application programming interface; active network socket programming;

I. Introduction

In 1990, Clark and Tennenhouse [1] proposed a design framework for introducing new network protocols for the Internet. Since the publication of that position paper, active network design framework [2, 3, 10] has slowly taken shape in the late 1990s. The active network paradigm allows program code and data to be delivered simultaneously on the Internet. Moreover, they may get executed and modified on their ways to their destinations. At the moment, there is a global active network backbone, the ABone, for experiments on active networks. Apart from the immaturity of the executing platform, the primary hindrance on the deployment of active networks on the Internet is more on the commercially related issues. For example, a vendor may hesitate to allow network routers to run some unknown programs that may affect their expected routing performance. As a result, alternatives were proposed to allow active network concept to operate on the Internet, such as the application layer active networking (ALAN) project [4] from the European research community. In the ALAN project, there are active server systems located at different places in the networks and active applications are allowed to run in these servers at the application layer. Another potential approach from the network service provider is to offer active network service as the premium service class in the networks. This service class should provide the best Quality of Service (QoS), and allow the access of computing facility in routers. With this approach, the network service providers can create a new source of income.

The research in active networks has been progressing steadily. Since active networks introduce programmability on the Internet, appropriate executing platforms for the active applications to execute should be established. These operating platforms are known as execution environments (EEs) and a few of them have been created, e.g., the Active Signaling Protocol (ASP) [12] and the Active Network Transport System (ANTS) [11]. Hence, different active applications can be implemented to test the active networking concept.

With these EEs, some experiments have been carried out to examine the active network concept, for example, the mobile networks [5], web proxies [6], and multicast routers [7]. Active networks introduce a lot of program flexibility and extensibility in networks. Several research groups have proposed various designs of execution environments to offer network computation within routers. Their performance and potential benefits to existing infrastructure are being evaluated [8, 9]. Unfortunately, they seldom concern the interoperability problems when the active networks consist of multiple execution environments. For example, there are three EEs in ABone. Active applications written for one particular EE cannot be operated on other platforms. This introduces another problem of resources partitioning for different EEs to operate. Moreover, there are always some critical network applications that need to run under all network routers, such as collecting information and deploying service at critical points to monitor the networks.

In this paper, a framework known as Active Network Socket Programming (ANSP) model is proposed to work with all EEs. It offers the following primary objectives.

- One single programming interface is introduced for writing active applications.
- Since ANSP offers the programming interface, the design of EE can be made independent of the ANSP.

This enables a transparency in developing and enhancing future execution environments.

- ANSP addresses the interoperability issues among different execution environments.
- Through the design of ANSP, the pros and cons of different EEs will be gained. This may help design a better EE with improved performance in future.

The primary objective of the ANSP is to enable all active applications that are written in ANSP can operate in the ABone testbed . While the proposed ANSP framework is essential in unifying the network environments, we believe that the availability of different environments is beneficial in the development of a better execution environment in future. ANSP is not intended to replace all existing environments, but to enable the studies of new network services which are orthogonal to the designs of execution environments. Therefore, ANSP is designed to be a thin and transparent layer on top of all execution environments. Currently, its deployment relies on automatic code loading with the underlying environments. As a result, the deployment of ANSP at a router is optional and does not require any change to the execution environments.

II. Design Issues on ANSP

The ANSP unifies existing programming interfaces among all EEs. Conceptually, the design of ANSP is similar to the middleware design that offers proper translation mechanisms to different EEs. The provisioning of a unified interface is only one part of the whole ANSP platform. There are many other issues that need to be considered. Apart from translating a set of programming interfaces to other executable calls in different EEs, there are other design issues that should be covered, e.g.,

- a unified thread library handles thread operations regardless of the thread libraries used in the EEs;
- a global soft-store allows information sharing among capsules that may execute over different environments at a given router;
- a unified addressing scheme used across different environments; more importantly, a routing information exchange mechanism should be designed across EEs to obtain a global view of the unified networks;
- a programming model that should be independent to any programming languages in active networks;
- and finally, a translation mechanism to hide the heterogeneity of capsule header structures.

A. *Heterogeneity in programming model*

Each execution environment provides various abstractions for its services and resources in the form of program calls. The model consists of a set of well-defined components, each of them has its own programming interfaces. For the abstractions, capsule-based programming model [10] is the most popular design in active networks. It is used in ANTS [11] and ASP [12], and they are being supported in ABone. Although they are developed based on the same capsule model, their respective components and interfaces are different. Therefore, programs written in one EE cannot run in anther EE. The conceptual views of the programming models in ANTS and ASP are shown in Figure 1.

The authors appreciate the Nortel Institute for Telecommunications (NIT) at the University of Toronto to allow them to access the computing facilities for this research project.

There are three distinct components in ANTS: application, capsule, and execution environment. There exist user interfaces for the active applications at only the source and destination routers. Then the users can specify their customized actions to the networks. According to the program function, the applications send one or more capsules to carry out the operations. Both applications and capsules operate on top of an execution environment that exports an interface to its internal programming resources. Capsule executes its program at each router it has visited. When it arrives at its destination, the application at destination may either reply it with another capsule or presents this arrival event to the user. One drawback with ANTS is that it only allows "bootstrap" application.

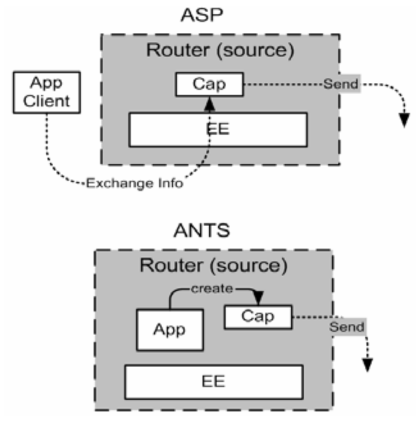


Figure 1. Programming Models in ASP and ANTS.

In contrast, ASP does not limit its users to run "bootstrap" applications. Its program interfaces are different from ANTS, but there are also has three components in ASP: application client, environment, and AAContext. The application client can run on active or non-active host. It can start an active application by simply sending a request message to the EE. The client presents information to users and allows its users to trigger actions at a nearby active router. AAContext is the core of the network service and its specification is divided into two parts. One part specifies its actions at its source and destination routers. Its role is similar to that of the application in ANTS, except that it does not provide a direct interface with the user. The other part defines its actions when it runs inside the active networks and it is similar to the functional behaviors of a capsule in ANTS.

In order to deal with the heterogeneity of these two models, ANSP needs to introduce a new set of programming interfaces and map its interfaces and execution model to those within the routers' EEs.

B. *Unified Thread Library*

Each execution environment must ensure the isolation of instance executions, so they do not affect each other or access others' information. There are various ways to enforce the access control. One simple way is to have one virtual machine for one instance of active applications. This relies on the security design in the virtual machines to isolate services. ANTS is one example that is using this method. Nevertheless, the use of multiple virtual machines requires relatively large amount of resources and may be inefficient in some cases. Therefore, certain environments, such as ASP, allow network services to run within a virtual machine but restrict the use of their services to a limited set of libraries in their packages. For instance, ASP provides its thread library to enforce access control. Because of the differences in these types of thread mechanism, ANSP devises a new thread library to allow uniform accesses to different thread mechanisms.

C. Soft-Store

Soft-store allows capsule to insert and retrieve information at a router, thus allowing more than one capsules to exchange information within a network. However, problem arises when a network service can execute under different environments within a router. The problem occurs especially when a network service inserts its soft-store information in one environment and retrieves its data at a later time in another environment at the same router. Due to the fact that execution environments are not allowed to exchange information, the network service cannot retrieve its previous data. Therefore, our ANSP framework needs to take into account of this problem and provides soft-store mechanism that allows universal access of its data at each router.

D. Global View of a Unified Network

When an active application is written with ANSP, it can execute on different environment seamlessly. The previously smaller and partitioned networks based on different EEs can now be merging into one large active network. It is then necessary to advise the network topology across the networks. However, different execution environments have different addressing schemes and proprietary routing protocols. In order to merge these partitions together, ANSP must provide a new unified addressing scheme. This new scheme should be interpretable by any environments through appropriate translations with the ANSP. Upon defining the new addressing scheme, a new routing protocol should be designed to operate among environments to exchange topology information. This allows each environment in a network to have a complete view of its network topology.

E. Language-Independent Model

Execution environment can be programmed in any programming language. One of the most commonly used languages is Java [13] due to its dynamic code loading capability. In fact, both ANTS and ASP are developed in Java. Nevertheless, the active network architecture shown in Figure 2 does not restrict the use of additional environments that are developed in other languages. For instance, the active network daemon, anted, in Abone provides a workspace to execute multiple execution environments within a router. PLAN, for example, is implemented in Ocaml that will be deployable on ABone in future. Although the current active network is designed to deploy multiple environments that can be in any programming languages, there lacks the tool to allow active applications to run seamlessly upon these environments. Hence, one of the issues that ANSP needs to address is to design a programming model that can work with different programming languages. Although our current prototype only considers ANTS and ASP in its design, PLAN will be the next target to address the programming language issue and to improve the design of ANSP.

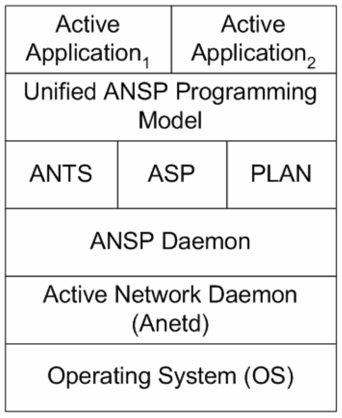


Figure 2. ANSP Framework Model.

F. Heterogeneity of Capsule Header Structure

The structures of the capsule headers are different in different EEs. They carries capsule-related information, for example, the capsule types, sources and destinations. This information is important when certain decision needs to be made within its target environment. A unified model should allow its program code to be executed on different environments. However, the capsule header prevents different environments to interpret its information successfully. Therefore, ANSP should carry out appropriate translation to the header information before the target environment receives this capsule.

III. ANSP PROGRAMMING MODEL

We have outlined the design issues encountered with the ANSP. In the following, the design of the programming model in ANSP will be discussed. This proposed framework provides a set of unified programming interfaces that allows active applications to work on all execution environments. The framework is shown in Figure 2. It is composed of two layers integrated within the active network architecture. These two layers can operate independently without the other layer. The upper layer provides a unified programming model to active applications. The lower layer provides appropriate translation procedure to the ANSP applications when it is processed by different environments. This service is necessary because each environment has its own header definition.

The ANSP framework provides a set of programming calls which are abstractions of ANSP services and resources. A capsule-based model is used for ANSP, and it is currently extended to map to other capsule-based models used in ANTS and ASP. The mapping possibility to other models remains as our future works. Hence, the mapping technique in ANSP allows any ANSP applications to access the same programming resources in different environments through a single set of interfaces. The mapping has to be done in a consistent and transparent manner. Therefore, the ANSP appears as an execution environment that provides a complete set of functionalities to active applications. While in fact, it is an overlay structure that makes use of the services provided from the underlying environments. In the following, the high-level functional descriptions of the ANSP model are described. Then, the implementations will be discussed. The ANSP programming model is based upon the interactions between four components: *application client*, *application stub*, *capsule*, and *active service base*.

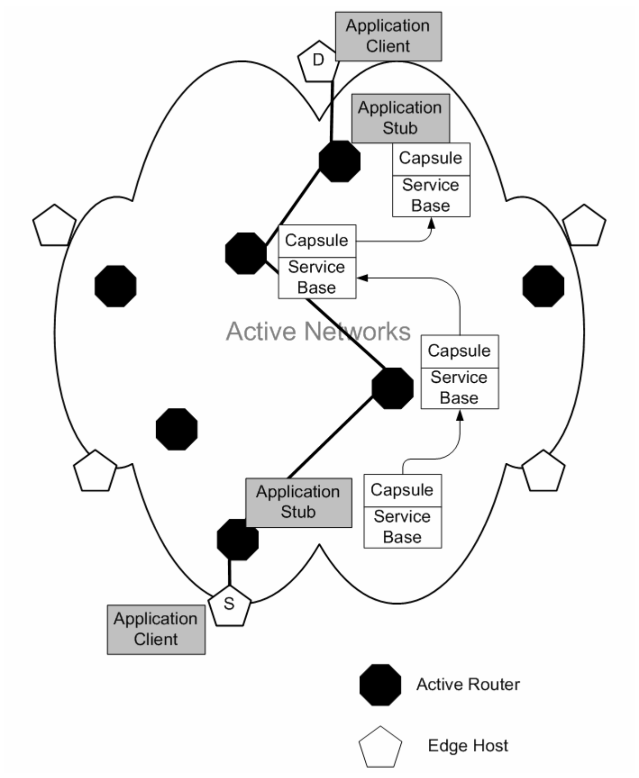


Figure 3. Information Flow with the ANSP.

- *Application Client*: In a typical scenario, an active application requires some means to present information to its users, e.g., the state of the networks. A graphical user interface (GUI) is designed to operate with the application client if the ANSP runs on a non-active host.
- *Application Stub*: When an application starts, it activates the application client to create a new instance of application stub at its near-by active node. There are two responsibilities for the application stub. One of them is to receive users' instructions from the application client. Another one is to receive incoming capsules from networks and to perform appropriate actions. Typically, there are two types of actions, that are, to reply or relay in capsules through the networks, or to notify the users regarding the incoming capsule.
- *Capsule*: An active application may contain several capsule types. Each of them carries program code (also referred to as forwarding routine). Since the application defines a protocol to specify the interactions among capsules as well as the application stubs. Every capsule executes its forwarding routine at each router it visits along the path between the source and destination.
- *Active Service Base*: An active service base is designed to export routers' environments' services and execute program calls from application stubs and capsules from different EEs. The base is loaded automatically at each router whenever a capsule arrives.

The interactions among components within ANSP are shown in Figure 3. The designs of some key components in the ANSP will be discussed in the following subsections.

A. Capsule (ANSPCapsule)

```
ANSPXdr decode()
ANSPXdr encode()
int length()
Boolean execute()
```

New types of capsule are created by extending the abstract class `ANSPCapsule`. New extensions are required to define their own forwarding routines as well as their serialization procedures. These methods are indicated below:

The execution of a capsule in ANSP is listed below. It is similar to the process in ANTS.

1. A capsule is in serial binary representation before it is sent to the network. When an active router receives a byte sequence, it invokes `decode()` to convert the sequence into a capsule.
2. The router invokes the forwarding routine of the capsule, `execute()`.
3. When the capsule has finished its job and forwards itself to its next hop by calling `send()`, this call implicitly invokes `encode()` to convert the capsule into a new serial byte representation. `length()` is used inside the call of `encode()` to determine the length of the resulting byte sequence.

ANSP provides a XDR library called ANSPXdr to ease the jobs of encoding and decoding.

B. Active Service Base (ANSPBase)

In an active node, the Active Service Base provides a unified interface to export the available resources in EEs for the rest of the ANSP components. The services may include thread management, node query, and soft-store operation, as shown in Table 1.

TABLE I. ACTIVE SERVICE BASE FUNCTION CALLS

Function Definition	Description
`boolean send(Capsule, Address)`	Transmit a capsule towards its destination using the routing table of the underlying environment.
`ANSPAddress getLocalHost()`	Return address of the local host as an `ANSPAddress` structure. This is useful when a capsule wants to check its current location.
`boolean isLocal(ANSPAddress)`	Return true if its input argument matches the local host's address and return false otherwise.
`createThread()`	Create a new thread that is a class of `ANSPThreadInterface` (discussed later in Section VIA "Unified Thread Abstraction").
`putSStore(key, Object)` `Object getSStore(key)` `removeSStore(key)`	The soft-store operations are provided by `putSStore()`, `getSSTore()`, and `removeSStore()`, and they put, retrieve, and remove data respectively.
`forName(PathName)`	Supported in ANSP to retrieve a class object corresponding to the given path name in its argument. This code retrieval may rely on the code loading mechanism in the environment when necessary.

C. *Application Client (ANSPClient)*

```
boolean start(args[])
boolean start(args[],runningEEs)
boolean start(args[],startClient)
boolean start(args[],startClient, runningEE)
```

Application Client is an interface between users and the nearby active source router. It does the following responsibilities.

1. Code registration: It may be necessary to specify the location and name of the application code in some execution environments, e.g., ANTS.
2. Application initialization: It includes selecting an execution environment to execute the application among those are available at the source router.

Each active application can create an application client instance by extending the abstract class, `ANSPClient`. The extension inherits a method, `start()`, to automatically handle both the registration and initialization processes. All overloaded versions of `start()` accept a list of arguments, `args`, that are passed to the application stub during its initialization. An optional argument called `runningEEs` allows an application client to select a particular set of environment variables, specified by a list of standardized numerical environment ID, the ANEP ID, to perform code registration. If this argument is not specified, the default setting can only include ANTS and ASP.

D. *Application Stub (ANSPApplication)*

```
receive(ANSPCapsule)
```

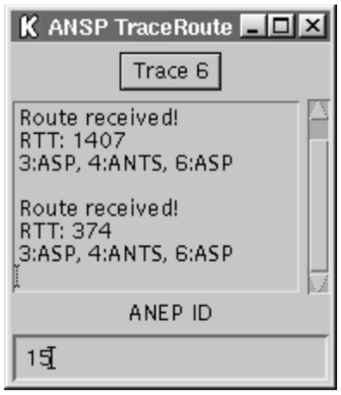
Application stubs reside at the source and destination routers to initialize the ANSP application after the application clients complete the initialization and registration processes. It is responsible for receiving and serving capsules from the networks as well as actions requested from the clients. A new instance is created by extending the application client abstract class, `ANSPApplication`. This extension includes the definition of a handling routine called `receive()`, which is invoked when a stub receives a new capsule.

IV. ANSP EXAMPLE: TRACE-ROUTE

A testbed has been created to verify the design correctness of ANSP in heterogeneous environments. There are three types of router setting on this testbed:

1. Router that contains ANTS and a ANSP daemon running on behalf of ASP;
2. Router that contains ASP and a ANSP daemon that runs on behalf of ANTS;
3. Router that contains both ASP and ANTS.

The prototype is written in Java [11] with a `traceroute` testing program. The program records the execution environments of all intermediate routers that it has visited between the source and destination. It also measures the RTT between them. Figure 4 shows the GUI from the application client, and it finds three execution environments along the path: ASP, ANTS, and ASP. The execution sequence of the `traceroute` program is shown in Figure 5.



```
Route received!
RTT: 1407
3:ASP, 4:ANTS, 6:ASP

Route received!
RTT: 374
3:ASP, 4:ANTS, 6:ASP
```

15

Figure 4. The GUI for the TRACEROUTE Program.

The `TraceCapsule` program code is created by extending the `ANSPCapsule` abstract class. When `execute()` starts, it checks the Boolean value of `returning` to determine if it is returning from the destination. It is set to true if `TraceCapsule` is traveling back to the source router; otherwise it is `false`. When traveling towards the destination, `TraceCapsule` keeps track of the environments and addresses of the routers it has visited in two arrays, `path` and `trace`, respectively. When it arrives at a new router, it calls `addHop()` to append the router address and its environment to these two arrays. When it finally arrives at the destination, it sets `returning` to `false` and forwards itself back to the source by calling `send()`.

When it returns to source, it invokes `deliverToApp()` to deliver itself to the application stub that has been running at the source. `TraceCapsule` carries information in its data field through the networks by executing `encode()` and `decode()`, which encapsulates and de-capsulates its data using External Data Representation (XDR) respectively. The syntax of ANSP XDR follows the syntax of XDR library from ANTS. `length()` in TraceCapsule returns the data length, or it can be calculated by using the primitive types in the XDR library.

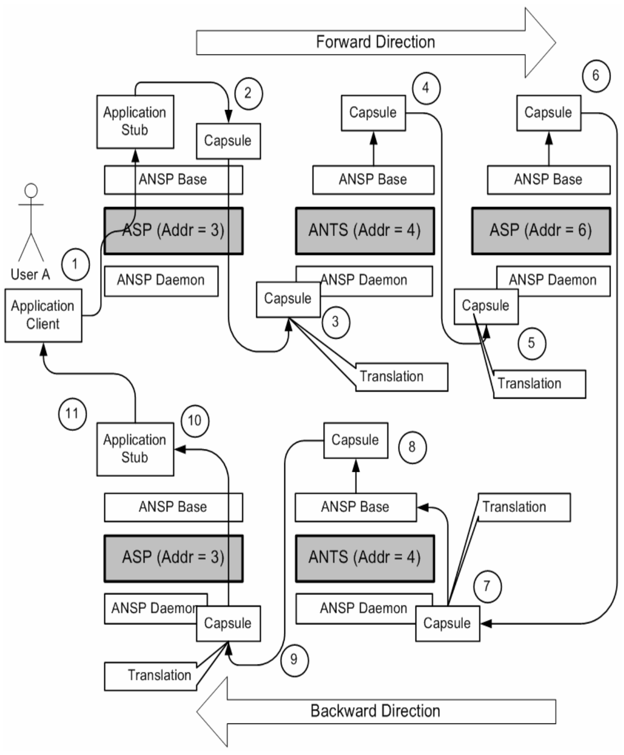


Figure 5. Flow of the TRACEROUTE Capsules.

V. CONCLUSIONS

In this paper, we present a new unified layered architecture for active networks. The new model is known as Active Network Socket Programming (ANSP). It allows each active application to be written once and run on multiple environments in active networks. Our experiments successfully verify the design of ANSP architecture, and it has been successfully deployed to work harmoniously with ANTS and ASP without making any changes to their architectures. In fact, the unified programming interface layer is light-weighted and can be dynamically deployable upon request.

REFERENCES

[1] D.D. Clark, D.L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," in *Proc. ACM Sigcomm'90*, pp.200-208, 1990.

[2] D. Tennenhouse, J. M. Smith, W. D. Sicoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, pp. 80-86, Jan 1997.

[3] D. Wetherall, U. Legedza, and J. Guttag, "Introducing new internet services: Why and how," *IEEE Network Magazine*, July/August 1998.

[4] M. Fry, A. Ghosh, "Application Layer Active Networking," in *Computer Networks*, Vol.31, No.7, pp.655-667, 1999.

[5] K. W. Chin, "An Investigation into The Application of Active Networks to Mobile Computing Environments", Curtin University of Technology, March 2000.

[6] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura, "Self Organizing Wide-Area Network Caches", *Proc. IEEE INFOCOM '98*, San Francisco, CA, 29 March-2 April 1998.

[7] L. H. Leman, S. J. Garland, and D. L. Tennenhouse, "Active Reliable Multicast", *Proc. IEEE INFOCOM '98*, San Francisco, CA, 29 March-2 April 1998.

[8] D. Descasper, G. Parulkar, B. Plattner, "A Scalable, High Performance Active Network Node", In IEEE Network, January/February 1999.

[9] E. L. Nygren, S. J. Garland, and M. F. Kaashoek, "PAN: a high-performance active network node supporting multiple mobile code system", In the Proceedings of the 2nd IEEE Conference on Open Architectures and Network Programming (OpenArch '99), March 1999.

[10] D. L. Tennenhouse, and D. J. Wetherall. "Towards an Active Network Architecture", In *Proceeding of Multimedia Computing and Networking*, January 1996.

[11] D. J. Wetherall, J. V. Guttag, D. L. Tennenhouse, "ANTS: A toolkit for Building and Dynamically Deploying Network Protocols", *Open Architectures and Network Programming, 1998 IEEE* , 1998 , Page(s): 117 –129.

[12] B. Braden, A. Cerpa, T. Faber, B. Lindell, G. Phillips, and J. Kann. "Introduction to the ASP Execution Environment": www.isi.edu/active-signal/ARP/index.html.

[13] "The java language: A white paper," Tech. Rep., Sun Microsystems, 1998.