Composable Services for Active Networks

AN Composable Services Working Group

May 1998

Abstract

A composition method provides a means to specify a composite service constructed from components. This document focuses on composition of services to be executed at a single node and within a single execution environment. The purpose is to identify features common to all composition methods, present examples of composition methods, and outline larger issues that may be amenable to common solutions.

1 Overview of Document

A composition method provides a means to specify a composite service constructed from components. As such, the realization of a composition method is a programming language whose primary purpose is to operate on components, but may have additional language capabilities (e.g., arithmetic, variable declarations). The method and language used to create the components are considered orthogonal to the composition method.

The emphasis of this document is composition of services to be executed at a *single node* and *within* a single execution environment (EE). Composition within an execution environment is largely an autonomous decision. This document, therefore, does not give a prescription about how composition should be accomplished. Instead the purpose is threefold: (1) to enumerate features that will present, in some form, in any composition method, (2) to provide examples of composition methods discussed in terms of the common features and (3) to outline common issues that go beyond basic functionality and may be amenable to shared solutions. The reader is assumed to be familiar with active networking in general [5] and the architectural framework being developed by the architecture working group [1].

This document includes:

- Terminology to discuss composition (Section 2).
- A list of reference examples of composite behavior (Section 3). A developer of a composition method might use this as a checklist to determine whether the method satisfies some of the range of desired composition capabilities. Naturally, this list of reference examples is not a complete specification of desired functionality.
- A list of attributes of a good composition method (Section 4). This list emphasizes networking-specific concerns that are likely to influence the success of a composition method.
- A list of features that will be present, in some form, in any composition method (Section 5). This list is intended to support a common method of describing the capabilities of a composition method, to ease comparisons. For example, a user who is selecting an EE for application

development might compare the composition support in different EEs using this list of properties.

- A set of examples of composition methods (Section 6). This captures part of the current state-of-the-art in composition methods.

- A discussion of issues that each composition method will need to address, beyond the basic issue of providing mechanisms to create composite services (Section 7). For example, this includes a discussion of the relationship between security and composition. Solutions to problems in these areas may be shared across different composition methods.

- A list of challenge problems in the area of composable services (Section 8).

1.1 Status of Memo

The starting point for this document was the rough consensus of the participants in the composable services working group at the Active Networking Workshop in Tucson, AZ, March 9-10, 1998. Comments are solicited and should be directed to the appropriate mailing lists (Section 10).

This development of this document was significantly influenced by the book "Programming Languages: Design and Implementation," Pratt and Zelkowitz, Third Edition, 1996.

2 Terminology

The following terms will be used to discuss composition.

- components - the entities that can be combined to form a composite

- composite - the entity that results from applying a composition method to a set of components; used synonymously with "composite service"

- composition method - the method for combining components to form a composite; the realization of a composition method will be a programming language for creating composites

3 Reference Examples

The examples given in this section are intended to serve two purposes. First, they define (by example) what is meant by composable services and therefore give the reader something concrete to ground the rest of the (more abstract) discussion in this document. Second, they provide the developer of a composition method with an incomplete list of desired functionality.

3.1 Traceroute

A traceroute service provides some information about each router in the path from a source to a specified destination. The Internet traceroute service nominally provides a list of the routers (identified by name and IP address) and three measurements of round trip time between the source and each router. An active traceroute service nominally provides the identify of the routers in the path from a source to a specified destination; the service may provide additional information (e.g., the list of EEs in operation at each router).

Active traceroute is likely to be a fairly simple service, that does not significantly exercise the composition method.

3.2 TCP Reno over IPv6

Composition of protocols according to standard layering should be supported by the composition method. For example, it should be possible to request the Reno version of the TCP protocol to be run over version 6 of the IP protocol.

3.3 Reliable Multicast

A multicast application must specify the type of routing to be used (e.g., shared versus source-specific trees), the type of reliability required (e.g., unreliable, exactly-once) and the mechanism to recover from lost packets (e.g., source recovery, local recovery). Thus, an example of a composite service is shared-tree routing with exactly-once delivery and local recovery from lost packets. Obviously any other legal combination of the pieces listed above forms a different composite service.

4 Attributes of a Good Composition Method

The success of a composition method depends on its acceptance for use by service developers. These developers will be influenced by both basic language features and attributes that are particular to networking. Basic language features of importance include clarity and simplicity, support for abstraction, naturalness for the application and ease of debugging and verification [3]. Attributes that are of particular interest for developing network services include:

- Performance

  Critical to the success of active networking is the ability to offer services that have good performance. The composition method plays an important role in the performance of composed services. For example, the composition method may dictate which parts of a service can be compiled versus interpreted during execution.

  A second component of performance concerns the global impact of using composite services. A composite service should offer good performance to the individual user and place a reasonable burden on the shared resources of the network. This requirement might favor, for example, a composition method that offers a concise packet format for specifying or requesting a composite service [4].

- Ease of security

  Some control over access to composite services will be necessary for security purposes. For example, a particular user may need to be authorized to access a particular composite service. A composition method should support the inclusion of the necessary security mechanisms, for example by preventing side affects that would allow a service to hide behavior. The security implications of composite services are discussed further in Section 7.

- Ease of management

  Once developed and in use, composite services will need to be managed. Management tasks will include tracking the use of services (for accounting and/or billing purposes), identifying the source of a problems that arise, upgrading services, and terminating service execution as needed. A composition method should support the development of network management for composite services either as an integral part of the composition process or as a straightforward post-process.

5 Features of Composition Methods

- Sequence control

  A composition method will have structures for controlling the order of execution of components. These structures are termed *sequence control*. Simple examples of ordering include sequential execution (e.g., execute component A in its entirety followed by component B) and concurrent execution (e.g., execute components A and B concurrently).

  More complex ordering results when components and composites can interact with one another to create hierarchy in the execution sequence. Various forms of hierarchy include: calling one component from another, calling a component from itelf (i.e., recursion) and calling a composite from another composite. The last form — encapsulating a composite to be used as a component in a new composite — requires a naming mechanism.

- Shared data control

  A composition method will have mechanisms that determine how data is shared between components. Examples of methods for sharing data include: *explicit parameters* that allow the shared data to be passed directly to the component, *shared objects* that allow the definition of data that is accessible by all components involved in sharing, and *inheritance* that allows implicit sharing based on pre-defined relationships between components.

  If components can execute in parallel and need to share data, then *message passing* between components provides another data sharing alternative. Note that if shared objects are used for data sharing amongst components executing in parallel, then access control mechanisms will be necessary to ensure the integrity of the data.

- Binding time

  Binding refers to the instantiation (or selection) of an entity from a set of options, as necessary for execution. In the context of composition, the entities that might be selected include the particular components in the composite. The possibility of different versions of components over time raises interesting binding issues. The options for binding time include at composite creation time (generally occurring at an end-system) and at composite execution time (generally occurring at an active node).

- Invocation methods

  The invocation methods are the events that can cause a composite to be executed. The most common invocation method will be the arrival of a packet at an active node. However, other events may also cause invocation of a composite. For example, an active node might be configured to execute a neighbor discovery composite as part of the system startup process.

- Division of functionality

The specification of a composite will occur through a combination of content carried in packets and content resident (or dynamically loaded on-demand) at nodes. For example, the packet might carry a program written in a scripting language that includes calls to node-resident service routines [2].

6 Examples of Composition Methods

This section contains examples of composition methods under development within the DARPA active networking program. The emphasis is on the way that each method addresses the features outlined in Section 5. The intention is to illustrate a set of options for composition and facilitate comparison between options.

6.1 Programming Language for Active Networks (PLAN)

[Nettles - 1 page]

6.2 Active Network Toolkit (ANTS)

[Wetherall - 1 page]

6.3 Netscript

[Florissi - 1 page]

6.4 Language-Independent Active Network Environment (LIANE)

Composition in LIANE has two parts. First, the user selects an *underlying program* from amongst those offered by an active node. The underlying program provides a basic service (e.g., forwarding) and includes a set of *processing slots* to be used for customization (e.g., to replace the default forwarding table with a customized forwarding table). Each processing slot is associated with a specific execution point in the underlying program. In the second part of composition, the user selects or provides a set of *injected programs* used to customize the underlying program. Each injected program is "bound" to one or more processing slots. The injected program is eligible for execution when the appropriate slot is reached ("raised"). More than one injected program may be bound to the same slot; the order of execution of instructions belonging to different injected programs bound to the same slot is non-deterministic. This style of composition has advantages with respect to proving properties about the composite service based on properties of the underlying program and preservation of properties by the injection process.

In the terminology of this document, the injected programs correspond to components and the underlying program specifies the basic sequence control and part of the execution of the composite. Using the terminology of Section 5, the features of LIANE are the following:

Sequence control The choice of underlying program provides the basic sequence control for a composite. An underlying program can impose linear sequencing by exporting mutually-exclusive slots that are raised in a particular sequence. To enforce strict sequential execution of components, only one injected program can be bound to each slot. Parallel execution can be achieved in two ways: multiple injected programs can be bound to a given slot or multiple slots can be raised at the same time. LIANE does not specify the order or interleaving of bound statements within a slot, thus the statements in different injected programs may be executed concurrently. Further, slots are raised if and only if some slot specific condition is true. Unless such conditions are mutually exclusive, multiple slots can be raised concurrently.

LIANE allows an injected program to raise events that other injected programs can bind to. Thus, a component can "call" another component. An underlying program with slot bindings (i.e., a composite) can be bound to a slot in another underlying program.

Shared data control Data can be shared between the underlying program and the injected programs, or between multiple injected programs. Both types of data sharing take place through a shared tuple space and through updates to shared variables. Thus, shared data control is achieved through shared objects. There is no notion of inheritance or of explicit parameters passed to components.

There is no explicit message passing or access control imposed by LIANE for shared access during concurrent execution. The injected programs are responsible for maintaining the integrity of their data during concurrent accesses.

Binding time The choice of underlying program is static and done at composite creation time (i.e., when the packet requesting the service is created). An initial choice of injected programs is also done at composite creation time. However, the event binding mechanism can support the dynamic binding of new injected programs at run-time.

Invocation methods Invocation of a composite typically occurs due to the arrival of a packet that requests a particular underlying program and selects or provides a set of injected programs. However, the event paradigm is sufficiently general that a composite could be bound to an event other than a packet arrival, for example a timer interrupt or a variable value exceeding a threshold.

Division of functionality The packet carries the selection of underlying program (chosen from a fixed set resident at the node) and the specification of the injected programs (chosen from those resident at the node or supplied explicitly via code in the packet). Resident at the node are a fixed set of underlying programs and a set of possible injected programs.

7 Other Issues

Composition of services raises a number of issues that are not specific to the particular composition method. Instead, these issues are likely to arise for any composition method and generic solutions may be applicable across composition methods. It is desirable to identify such orthogonal issues so that solutions can be developed independent of composition methods.

- Node-resident content deployment

  As discussed earlier, the specification of a composite will occur through a combination of content carried in packets and content resident at nodes. This raises the issue of how the node-resident content is deployed. Possibilities include off-line loading by the node provider as part of node management and on-demand loading. A variety of techniques are possible for on-demand loading, for example code pulling [6].

- Resource discovery

  Creating or using a composite service requires knowledge of available components or composite services. This raises a number of issues in the area of resource discovery. How are components and composites named, and how are those names known? How are different versions of the same component or composite managed? What scoping rules apply for the availability of resources? For example, is the availability of a composite service affected by the part of the network topology where invocation will occur?

- Errors and error handling

  Errors associated with composition may occur during creation or execution of a composite. Error checking and handling during creation of a composite is likely to be specific to the composition method and thus is not discussed here. We restrict this discussion to errors that can occur during execution, with the understanding that the frequency of these errors may be affected by error checking at composite creation time.

  A variety of errors may occur during composite execution. For example, some part of the expected node-resident content may not be present, even after any on-demand loading mechanisms have been employed. Content expected in the packet may not be present. The execution of a component or composite may exceed resource limits.

  Related to the issue of errors are the mechanisms and policies for handling errors. For example, when an error occurs, who is notified? If an error occurs in a component, how does that affect the rest of the composite?

- Security

  Composition raises several security issues. Some composites will require that the user be authorized to execute the composite. Some components will require that the composite service developer be authorized to use the components in service creation, or that the composite carry security associations based on the components involved. The authorization to use components may depend on the details of the resulting composite service. (For example, a service developer or user may be authorized to use components A and B, but not in arbitrary compositions.) The discovery of composition resources may also be subject to security-based access constraints. For example, some components and composites may require authorization to ask about their availability.

- Accounting

  The use of some components and composites will need to be monitored for accounting purposes.

8 Challenge Problems

[Needs to be done.]

9 What is Not Covered

This document does not (currently) address composition of services to be executed across groups of nodes or composition of services involving multiple execution environments.

10 Addresses

Comments should be sent to the mailing list:
    ActiveNets_Services@ittc.ukans.edu
also archived at:
    http://www.ittc.ukans.edu/ãnserve/
or to the editor:
Ellen Zegura
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
Phone: 404-894-1403
Fax: 404-894-0272
Email: ewz@cc.gatech.edu

References

[1] AN Architecture Working Group. Architectural framework for active networks. Ken Calvert, editor, 1998.

[2] M. Hicks, P. Kakkar, T. Moore, C. Gunter, and S.Nettles. PLAN: A Programming Language for Active Networks. Submitted to International Conference on Functional Programming, 1998.

[3] T. Pratt and M. Zelkowitz. *Programming Languages: Design and Implementation.* Prentice Hall, third edition, 1996.

[4] B. Schwartz, W. Zhou, Z. Jackson, W. Strayer, D. Rockwell, and C. Partridge. Smart packets for active networks. http://www.net-tech.bbn.com/smtpkts/smtpkts.ps.gz, Jan 1998.

[5] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1), 1997.

[6] D. Wetherall, J. Guttag, and D. L. Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. In *IEEE OPENARCH'98*, San Francisco, CA, April 1998.