

Alfa's Characterization of Network-Based Styles

Nikunj R. Mehta

October 2003.

Table 1: Pipe-and-filter style characteristics

| Characteristic | Pipe-and-filter style (PF) |
|-----------------------|---|
| Data | <i>Records</i> are exchanged between pipes and filters. A special <i>end-of-pipe</i> signal is used to indicate that the pipe does not have any further records. |
| Structure | <i>A filter</i> has interfaces for <i>writing</i> to, <i>reading</i> from, <i>closing</i> pipes, and being <i>notified</i> of the closing of pipes. Every filter should have interfaces for either reading from or writing to pipes or both. It should also have an interface for closing every pipe to which it has an interface for writing, and being notified about its closing by any filter. <i>A pipe</i> has interfaces for <i>source</i> and <i>sink</i> filters, to allow a source to <i>close</i> the pipe, and to <i>notify</i> source filters about its closing. Every pipe should have interfaces for both source and sink filters. It should also have interfaces for being closed by every source filter, and notifying all sources when it is closed. |
| Interaction | A filter blocks when it writes to a pipe, when it closes a pipe, and when it reads from a pipe but no records are available for reading. A pipe does not block when it notifies sources of its closing. |
| Behavior | A filter cannot write to a pipe, once it is notified that the pipe has been closed. A filter cannot close a pipe that has already been closed. A pipe should notify all its sources once it is closed by any of its sources. A pipe should relay all data received from its source filters to every one of its sink filter after combining the data uniformly. |
| Topology | A filter's write interface is connectable to a pipe's source interface, its read interface is connectable to a pipe's sink interface, its close interface is connectable to a pipe's close interface, and a pipe's notify interface is connectable to a filter's notified interface. A filter cannot be connected to a pipe through more than one read/write interface. |

Table 2: Uniform pipe-and-filter style characteristics

| Characteristic | Uniform pipe-and-filter style (UPF) |
|-----------------------|---|
| Data | <i>Records</i> are exchanged between pipes and filters. A special <i>end-of-pipe</i> signal is used to indicate that the pipe does not have any further records. All records have the same data type. |
| Structure | <p>A <i>filter</i> has interfaces for <i>writing</i> to, <i>reading</i> from, <i>closing</i> pipes, and being <i>notified</i> of the closing of pipes. Every filter should have interfaces for either reading from or writing to pipes. It should also have an interface for closing every pipe to which it has an interface for writing, and being notified about its closing by any filter. Every <i>transforming</i> filter should have exactly two interfaces to write to a pipe and one interface to read from a pipe. Every <i>source</i> filter should have one interface to write to a pipe, but no interfaces for reading from a pipe, and no <i>sink</i> filter can have an interface to write to a pipe.</p> <p>A <i>pipe</i> has interfaces for <i>source</i> and <i>sink</i> filters, to allow a source to <i>close</i> the pipe, and to <i>notify</i> source filters about its closing. Every pipe should have interfaces for both source and sink filters. It should also have interfaces for being closed by every source filter, and notifying all sources when it is closed. All pipes have exactly one sink and one source interfaces.</p> |
| Interaction | A filter blocks when it writes to a pipe, when it closes a pipe, and when it reads from a pipe but no records are available for reading. A pipe does not block when it notifies sources of its closing. |
| Behavior | A filter cannot write to a pipe, once it is notified that the pipe has been closed. A filter cannot close a pipe that has already been closed. A pipe should notify all its sources once it is closed by any of its sources. A pipe should relay all data received from its source filters to every one of its sink filter after combining the data uniformly. |
| Topology | A filter's write interface is connectable to a pipe's source interface, its read interface is connectable to a pipe's sink interface, its close interface is connectable to a pipe's close interface, and a pipe's notify interface is connectable to a filter's notified interface. A filter cannot be connected to a pipe through more than one read/write interface. Connected pipes and filters cannot form cycles. |

Table 3: Replicated repository style characteristics

| Characteristic | Replicated repository style (RR) |
|-----------------------|--|
| Data | <i>Resources</i> , and <i>requests</i> for reading, closing, updating and creating resources are exchanged between clients and repositories. <i>Resources</i> , <i>token requests</i> and <i>resource tokens</i> are exchanged among repositories. |
| Structure | A <i>client</i> has interfaces for <i>reading</i> resources from, <i>closing</i> , <i>updating</i> or <i>creating</i> resources upon a repository. Each client must have at least one of the these interfaces. It should also have an interface to close resources upon a repository iff it has an interface to read from that repository. A <i>repository</i> has interfaces to <i>accept requests</i> from clients to read, close, update, and create resources. It should have an interface to accept resources being closed for every client that reads resources from it. It has interfaces to <i>provide resources</i> for being read and updated. It also has interfaces to <i>make token requests</i> , <i>accept token requests</i> , <i>provide resource tokens</i> , and <i>accept resource tokens</i> to ensure correct patterns of shared resource access. Each repository should have an interface to accept resource tokens iff makes token requests to that repository. |
| Interaction | Clients block for resources requested to be read, closed, updated or created as well as for closing, updating, or creating them. Repositories block neither for requests or resources from clients, nor for requests for tokens to other repositories, nor for resource tokens from other repositories. |
| Behavior | Clients can only close a resource after reading it. Clients can only update (create) a resource after requesting to update (create) it. Repositories respond with a resource only after being requested for it by a client. Repositories can only generate resource tokens after receiving requests for them. Repositories can only make token requests after receiving resource requests from clients to read, update, or create them . Repositories cannot respond to resource requests until they have the necessary resource tokens to satisfy the request. |
| Topology | A client's read interface is connectable to a repository's read request interface, its close interface is connectable to a repository's close request interface, and its read resource interface is connectable to a repository's resource-for-being-read interface. A client's update interface is connectable to a repository's update request interface, its update resource interface is connectable to a repository's resource-for-being-updated interface, and its updated resource interface is connectable to a repository's updated resource interface. A client's create interface is connectable to a repository's create request interface, and its created resource interface is connectable to a repository's created resource interface. A repository's token request interface is connectable to another repository's accept-token-request interface, its resource token interface is connectable to another repository's accept-resource-token interface, its resource interface is connectable to another repository's accept-resource interface. Every repository should be connected to all other repositories. A client is connected to only one repository. |

Table 4: Cache style characteristics

| Characteristic | Cache style (X) |
|-----------------------|---|
| Data | <i>Resources and requests</i> for reading resources are exchanged between clients, caches, and repositories. |
| Structure | <p>A <i>client</i> has an interface each for <i>requesting</i> and <i>receiving</i> resources.</p> <p>A <i>cache</i> has an interface to <i>accept resource requests</i> from and <i>provide resources</i> to clients, and interfaces to <i>make resource requests</i> to and <i>accept resources</i> from repositories. Each cache has an interface to provide resources to a client iff the client makes resource requests to it. Each cache has an interface to accept resources from repositories iff it requests resources from it.</p> <p>A <i>repository</i> has interfaces to <i>accept resource requests</i> and <i>provide resources</i>.</p> |
| Interaction | Clients block for resources they have requested. Caches block for resources they have requested. Repositories do not block for resource requests. |
| Behavior | <p>Caches (repositories) do not provide resources unless requested by a client (cache). A cache may respond to a resource request from a client without requesting a repository for that resource.</p> <p>Clients and caches immediately switch to receiving a resource after requesting them</p> |
| Topology | <p>A client's resource request interface is connectable to a cache's accept resource request interface, and its resource interface is connectable to a cache's provide resource interface.</p> <p>A cache's resource request interface is connectable to a repository's accept resource request interface and its accept resource interface is connectable to a repository's provide resource interface.</p> <p>Each client should be connected to one cache, and vice versa.</p> |

Table 5: Client-server style (remote session, remote data access, and client-stateless server styles) characteristics

| Characteristic | Client-server style (CS) |
|-----------------------|--|
| Data | <i>Requests and responses are exchanged between clients and servers.</i> |
| Structure | <p><i>A client has interfaces to request services and receiving responses. A client has a response interface for each of its request interface.</i></p> <p><i>A server has an interface to accept service requests and provide responses.</i></p> <p><i>A transport has interfaces to accept service requests and provide responses, and an interface to make request for service and accept responses.</i></p> |
| Interaction | <p>Clients block for responses to their service requests. Servers and transports do not block for service requests. Transports do not block when making requests for service.</p> |
| Behavior | <p>Servers reply to a request by generating a response.</p> <p>Transports reply to a request by generating a response. Transports forward all requests to a server through their make service request interface, and the responses received on accept responses interface to the response interface for the corresponding client.</p> <p>Clients immediately switch to receiving a response after sending service requests.</p> |
| Topology | <p>A client's request interface is connectable to a transport's request interface, and its response interface is connectable to a transport's response interface.</p> <p>A transport's make request interface is connectable to a server's request interface and its accept response interface is connectable to a server's response interface.</p> <p>Each server should be connected to one transport, and vice versa.</p> <p>A client cannot be connected to a transport via more than one request/response interfaces.</p> |

Table 6: Layered style characteristics

| Characteristic | Layered style (LS) |
|-----------------------|--|
| Data | Call <i>parameters</i> and call <i>results</i> are exchanged. |
| Structure | <p><i>Layers</i> contain interfaces for <i>call invocation parameter</i>, <i>call invocation results</i> and an interface each for <i>call processing parameter</i> and <i>call processing results</i>. In a layer, each call invocation parameter interface is always used in conjunction with a call invocation result interface. Similarly, a layer's call processing parameter interface is always used in conjunction with its call processing result interface.</p> <p>Call <i>routers</i> contain an interface for <i>call invocation parameter</i> and <i>call invocation result</i>, and interfaces for <i>call processing parameter</i> and <i>call processing result</i>.</p> |
| Interaction | Layers and routers block for sending on call invocation parameter interface. Layers and routers do not block on call processing parameter interfaces. Layers block for receiving call invocation results. |
| Behavior | Layers and routers reply to a call processing parameter through a call processing result. Routers forward all call processing parameters to their call invocation parameter interface and the call invocation result to the corresponding call processing result interface. Layers immediately switch to receiving a call invocation result after sending call invocation parameters. |
| Topology | <p>A layer's call invocation parameter interface is connectable to a call router's call processing parameter interface, a router's call processing result interface is connectable to a layer's call invocation result interface. A router's call invocation parameter interface is connectable to a layer's call processing parameter interface, and a layer's call processing result interface is connectable to a router's call invocation result interface.</p> <p>In a graph of connected layers, cycles are not allowed.</p> |

Table 7: Layered client-server style (and layered-client-stateless-server) characteristics

| Characteristic | Layered client-server style (LCS) |
|--------------------|--|
| Data | <i>Requests and responses</i> are exchanged between clients and servers. All requests are treated as call parameters, and responses as call results. |
| Structure | <p>A <i>client layer</i> has interfaces to <i>request</i> services and receiving <i>responses</i>, which are mapped to its call invocation parameter, and call invocation result interfaces.</p> <p>Call <i>routers</i> contain an interface for <i>call invocation parameter</i> and <i>call invocation result</i>, and interfaces for <i>call processing parameter</i> and <i>call processing result</i>.</p> <p>A <i>proxy layer</i> has interfaces to accept service <i>requests</i> and provide <i>responses</i>, mapped to its call processing parameter and call processing result interfaces, and to <i>make requests</i> for service and <i>accept responses</i>.</p> <p>A <i>transport</i> has interfaces to accept service <i>requests</i> and provide <i>responses</i>, and an interface to <i>make request</i> for service and <i>accept responses</i>.</p> <p>A <i>gateway layer</i> has interfaces to accept service <i>requests</i> and provide <i>responses</i>, and to <i>make requests</i> for service and <i>accept responses</i> mapped to its call invocation parameter, and call invocation result interfaces</p> <p>A <i>server layer</i> has an interface to accept service <i>requests</i> and provide <i>responses</i> mapped to its call processing parameter and call processing result interfaces.</p> |
| Interaction | As in the layered style, client layers, proxy layers, transports, and gateway layers block for responses to their service requests. Also similar to the layered style, server layers, proxy layers, gateway layers, and transports do not block for accepting service requests. Transports do not block when making requests for service. |
| Behavior | Servers reply to a request by generating a response. Transports forward all requests to a gateway through their make service request interface, and the responses received on accept responses interface to the response interface for the corresponding proxy. Clients, proxies, transports, and gateways immediately switch to receiving a response after sending service requests. Proxies forward service requests to a transport through their make service request interface, and the responses received on accept response interface to the response interface of the corresponding client. Gateways forward all requests to a server through their make service request interface, and the responses received on accept response interface to the response interface of the transport. |
| Topology | <p>A client layer is connected to exactly one proxy layer through a call router.</p> <p>Some gateway layer is connected to every server layer through a call router.</p> <p>A transport is connected to a gateway layer, and vice versa.</p> <p>The proxy layer's make request interface is connectable to a transport's request interface, and a transport's response interface is connectable to a proxy's response interface.</p> <p>A transport's make request interface is connectable to a gateway's request interface and a gateway's response interface is connectable to a transport's accept response interface.</p> |

Table 8: Client-cache-server style (and Client-cache-stateless-server style) characteristics

| Characteristic | Client-cache-server style (CXS) |
|--------------------|--|
| Data | <i>Requests and responses</i> are exchanged between clients and servers. |
| Structure | <p>A <i>client</i> has interfaces to <i>request</i> services and receiving <i>responses</i>. A client has a response interface for each of its request interface.</p> <p>A <i>cache</i> has an interface to <i>accept resource requests</i> from and <i>provide resources</i> to clients, and interfaces to <i>make resource requests</i> to and <i>accept resources</i> from repositories. Each cache has an interface to provide resources to a client iff the client makes resource requests to it.</p> <p>Each cache has an interface to accept resources from repositories iff it requests resources from it.</p> <p>A <i>server</i> has an interface to accept service <i>requests</i> and provide <i>responses</i>.</p> <p>A <i>transport</i> has interfaces to accept service <i>requests</i> and provide <i>responses</i>, and an interface to <i>make request</i> for service and <i>accept responses</i>.</p> |
| Interaction | Clients and caches block for responses to their service requests. Caches, servers and transports do not block for service requests. Transports do not block when making requests for service. |
| Behavior | <p>Servers reply to a request by generating a response.</p> <p>Transports reply to a request by generating a response. Transports forward all requests to a server through their make service request interface, and the responses received on accept responses interface to the response interface for the corresponding client.</p> <p>Caches do not provide resources unless requested by a client. A cache may respond to a request from a client without forwarding the request to a transport.</p> <p>Clients, and caches immediately switch to receiving a response after sending service requests.</p> |
| Topology | <p>A client's request interface is connectable to a cache's request interface, and a cache's response interface is connectable to a client's response interface.</p> <p>A cache's make request interface is connectable to a transport's request interface and a transport's response interface is connectable to a cache's accept response interface.</p> <p>A transport's make request interface is connectable to a server's request interface and a server's response interface is connectable to a transport's accept response interface.</p> <p>Each server should be connected to one transport, and vice versa.</p> <p>Each client should be connected to one cache, and vice versa.</p> <p>A cache cannot be connected to a transport via more than one request/response interfaces.</p> |

Table 9: Layered client-cache-stateless-server style characteristics

| Characteristic | Layered client-cache-stateless-server style (LCXS) |
|-----------------------|--|
| Data | <i>Requests and responses</i> are exchanged between clients and servers. All requests are treated as call parameters, and responses as call results. |
| Structure | <p>A <i>client</i> has an interface to <i>request</i> services and receive <i>responses</i>.</p> <p>A <i>cache layer</i> has interfaces to accept service <i>requests</i> and provide <i>responses</i>, and to <i>make requests</i> for service and <i>accept responses</i>, which are mapped to its call invocation parameter, and call invocation result interfaces.</p> <p>Call <i>routers</i> contain an interface for <i>call invocation parameter</i> and <i>call invocation result</i>, and interfaces for <i>call processing parameter</i> and <i>call processing result</i>.</p> <p>A <i>proxy layer</i> has interfaces to accept service <i>requests</i> and provide <i>responses</i>, mapped to its call processing parameter and call processing result interfaces, and to <i>make requests</i> for service and <i>accept responses</i>.</p> <p>A <i>transport</i> has interfaces to accept service <i>requests</i> and provide <i>responses</i>, and an interface to <i>make request</i> for service and <i>accept responses</i>.</p> <p>A <i>gateway layer</i> has interfaces to accept service <i>requests</i> and provide <i>responses</i>, and to <i>make requests</i> for service and <i>accept responses</i> mapped to its call invocation parameter, and call invocation result interfaces</p> <p>A <i>server layer</i> has an interface to accept service <i>requests</i> and provide <i>responses</i> mapped to its call processing parameter and call processing result interfaces.</p> |
| Interaction | As in the layered style, cache layers, proxy layers, transports, and gateway layers block for responses to their service requests. Also similar to the layered style, server layers, proxy layers, gateway layers, and transports do not block for accepting service requests. Clients block for responses. |
| Behavior | Server layers reply to a request by generating a response. Transports reply to a request by generating a response. Transports forward all requests to a gateway layer through their make service request interface, and the responses received on accept responses interface to the response interface for the corresponding proxy layer. Clients, cache layers, proxy layers, transports, and gateway layers immediately switch to receiving a response after sending service requests. Proxy layers forward service requests to a transport through their make service request interface, and the responses received on accept response interface to the response interface of the corresponding cache layer. Gateway layers forward all requests to a server through their make service request interface, and the responses received on accept response interface to the response interface of the transport. |

Table 9: Layered client-cache-stateless-server style characteristics

| Characteristic | Layered client-cache-stateless-server style (LCXS) |
|-----------------------|--|
| Topology | <p>A client's request interface is connectable to a cache layer's request interface, and a cache layer's response interface is connectable to a client's response interface.</p> <p>Each client is connected to one cache layer, and vice versa.</p> <p>A cache layer is connected to exactly one proxy layer through a call router.</p> <p>Some gateway layer is connected to every server layer through a call router.</p> <p>A transport is connected to a gateway layer, and vice versa.</p> <p>The proxy layer's make request interface is connectable to a transport's request interface, and a transport's response interface is connectable to a proxy layer's response interface.</p> <p>A transport's make request interface is connectable to a gateway layer's request interface and a gateway layer's response interface is connectable to a transport's accept response interface.</p> |

Table 10: Virtual machine style characteristics

| Characteristic | Virtual machine style (VM) |
|-----------------------|---|
| Data | Interpretation <i>expression</i> and interpretation <i>results</i> are exchanged. |
| Structure | <i>Clients</i> and <i>virtual machines</i> contain an interface for interpretation <i>expression</i> and interpretation <i>results</i> . Each client's (and virtual machine's) interpretation code interface is used in conjunction with an interpretation result interface. |
| Interaction | Clients block for interpretation results. Virtual machines do not block on interpretation expression. |
| Behavior | Virtual machines reply to interpretation expression through an interpretation result. Clients immediately switch to receiving an interpretation result after sending interpretation expression. |
| Topology | A client's interpretation expression interface is connectable to a virtual machine's interpretation expression interface, a virtual machine's interpretation result interface is connectable to a client's interpretation result interface. Each client has a connected virtual machine. |

Table 11: Remote evaluation style characteristics

| Characteristic | Remote evaluation style (REV) |
|-----------------------|---|
| Data | <i>Requests and responses, interpretation expression and interpretation results are exchanged.</i> |
| Structure | <p><i>Clients contain an interface for interpretation expression and interpretation results.</i></p> <p><i>A server has an interface to accept service requests and provide responses. Servers and virtual machines also contain an interface for code interpretation and communicating results of interpretation.</i></p> <p><i>A transport has interfaces to accept service requests and provide responses, and an interface to make request for service and accept responses.</i></p> |
| Interaction | <p><i>Clients block for responses to their service requests. Servers and transports do not block for service requests. Servers block for results of interpretation. Virtual machines do not block for expression.</i></p> |
| Behavior | <p><i>Virtual machines reply to interpretation expression through an interpretation result.</i></p> <p><i>Servers reply to a request by generating a response. Servers make a code interpretation request immediately after receiving a request from a client. After this servers immediately switch to receiving interpretation results. Interpretation results are returned as a response.</i></p> <p><i>Transports forward all requests to a server through their make service request interface, and the responses received on accept responses interface to the response interface for the corresponding client.</i></p> <p><i>Clients immediately switch to receiving a response after sending service requests.</i></p> |
| Topology | <p><i>A client's request interface is connectable to a transport's request interface, and its response interface is connectable to a transport's response interface.</i></p> <p><i>A transport's make request interface is connectable to a server's request interface and its accept response interface is connectable to a server's response interface.</i></p> <p><i>Each server should be connected to one transport, and vice versa. A server's interpretation expression interface is connectable to a virtual machine's interpretation expression interface, a virtual machine's interpretation result interface is connectable to a server's interpretation result interface. Each server has a connected virtual machine.</i></p> <p><i>A client cannot be connected to a transport via more than one request/response interfaces.</i></p> |

Table 12: Code-on-demand style characteristics

| Characteristic | Code-on-demand style (COD) |
|-----------------------|---|
| Data | <i>Requests and responses, interpretation expression and interpretation results are exchanged.</i> |
| Structure | <p><i>Clients and virtual machines contain an interface for interpretation expression and interpretation results. Clients also contain an interface to request interpretation code and obtain responses to these requests. A client has a response interface for each of its request interfaces.</i></p> <p><i>A server has an interface to accept service requests and provide responses.</i></p> <p><i>A transport has interfaces to accept service requests and provide responses, and an interface to make request for service and accept responses.</i></p> <p><i>Each client's (and virtual machine's) interpretation code interface is used in conjunction with an interpretation result interface.</i></p> |
| Interaction | <p><i>Clients block for responses to their service requests, and for results of interpretation. Servers and transports do not block for service requests. Virtual machines do not block for interpretation expression.</i></p> |
| Behavior | <p><i>Virtual machines reply to interpretation expression through an interpretation result.</i></p> <p><i>Servers reply to a request by generating a response.</i></p> <p><i>Transports reply to a request by generating a response. Transports forward all requests to a server through their make service request interface, and the responses received on accept responses interface to the response interface for the corresponding client.</i></p> <p><i>Clients and transports immediately switch to receiving a response after sending service requests. Clients immediately switch to receiving interpretation results after issuing interpretation expression. Clients may make an interpretation request immediately after receiving a response to their request for interpretation expression from a server.</i></p> |
| Topology | <p><i>A client's request interface is connectable to a transport's request interface, and its response interface is connectable to a transport's response interface. A client's interpretation expression interface is connectable to a virtual machine's interpretation expression interface, a virtual machine's interpretation result interface is connectable to a client's interpretation result interface.</i></p> <p><i>Each client has a connected virtual machine.</i></p> <p><i>A transport's make request interface is connectable to a server's request interface and its accept response interface is connectable to a server's response interface.</i></p> <p><i>Each server should be connected to one transport, and vice versa.</i></p> <p><i>A client cannot be connected to a transport via more than one request/response interfaces.</i></p> |

Table 13: Layered client-code-on-demand-cache-server style characteristics

| Characteristic | Layered client-code-on-demand-cache-server style (LCCODXS) |
|-----------------------|--|
| Data | <i>Requests and responses, interpretation code and interpretation results are exchanged.</i> |
| Structure | <p><i>Clients and virtual machines contain an interface for interpretation code and interpretation results.</i></p> <p><i>Clients also contain an interface to request interpretation code and obtain responses to these requests. A client has a response interface for each of its request interfaces.</i></p> <p><i>A server has an interface to accept service requests and provide responses.</i></p> <p><i>A transport has interfaces to accept service requests and provide responses, and an interface to make request for service and accept responses.</i></p> <p><i>Each client's (and virtual machine's) interpretation code interface is used in conjunction with an interpretation result interface. Similarly, a layer's call processing parameter interface is always used in conjunction with its call processing result interface.</i></p> <p><i>A cache layer has interfaces to accept service requests and provide responses, and to make requests for service and accept responses, which are mapped to its call invocation parameter, and call invocation result interfaces.</i></p> <p><i>Call routers contain an interface for call invocation parameter and call invocation result, and interfaces for call processing parameter and call processing result.</i></p> <p><i>A proxy layer has interfaces to accept service requests and provide responses, mapped to its call processing parameter and call processing result interfaces, and to make requests for service and accept responses.</i></p> <p><i>A transport has interfaces to accept service requests and provide responses, and an interface to make request for service and accept responses.</i></p> <p><i>A gateway layer has interfaces to accept service requests and provide responses, and to make requests for service and accept responses mapped to its call invocation parameter, and call invocation result interfaces</i></p> <p><i>A server layer has an interface to accept service requests and provide responses mapped to its call processing parameter and call processing result interfaces.</i></p> |
| Interaction | <p><i>Clients block for responses to their service requests, and for results of interpretation code.</i></p> <p><i>Servers and transports do not block for service requests.</i></p> <p><i>Virtual machines do not block for interpretation expression.</i></p> <p><i>As in the layered style, cache layers, proxy layers, transports, and gateway layers block for responses to their service requests. Also similar to the layered style, server layers, proxy layers, gateway layers, and transports do not block for accepting service requests.</i></p> |

Table 13: Layered client-code-on-demand-cache-server style characteristics

| Characteristic | Layered client-code-on-demand-cache-server style (LCCODXS) |
|-----------------------|--|
| Behavior | <p>Virtual machines reply to interpretation code through an interpretation result.</p> <p>Clients immediately switch to receiving interpretation results after issuing interpretation code. Clients may make an interpretation request immediately after receiving a response to their request for interpretation code from a server.</p> <p>Server layers reply to a request by generating a response. Transports reply to a request by generating a response. Transports forward all requests to a gateway layer through their make service request interface, and the responses received on accept responses interface to the response interface for the corresponding proxy layer. Clients, cache layers, proxy layers, transports, and gateway layers immediately switch to receiving a response after sending service requests. Proxy layers forward service requests to a transport through their make service request interface, and the responses received on accept response interface to the response interface of the corresponding cache layer. Gateway layers forward all requests to a server layer through their make service request interface, and the responses received on accept response interface to the response interface of the transport.</p> |
| Topology | <p>A client's interpretation code interface is connectable to a virtual machine's interpretation code interface, a virtual machine's interpretation result interface is connectable to a client's interpretation result interface.</p> <p>Each client has a connected virtual machine.</p> <p>A transport's make request interface is connectable to a server's request interface and its accept response interface is connectable to a server's response interface.</p> <p>Each server should be connected to one transport, and vice versa.</p> <p>A client cannot be connected to a transport via more than one request/response interfaces.</p> <p>A client's request interface is connectable to a cache layer's request interface, and a cache layer's response interface is connectable to a client's response interface.</p> <p>Each client is connected to one cache layer, and vice versa.</p> <p>A cache layer is connected to exactly one proxy layer through a call router.</p> <p>Some gateway layer is connected to every server layer through a call router.</p> <p>A transport is connected to a gateway layer, and vice versa.</p> <p>The proxy layer's make request interface is connectable to a transport's request interface, and a transport's response interface is connectable to a proxy layer's response interface.</p> <p>A transport's make request interface is connectable to a gateway layer's request interface and a gateway layer's response interface is connectable to a transport's accept response interface.</p> |

Table 14: Mobile agent style characteristics

| Characteristic | Mobile agent style (MA) |
|-----------------------|--|
| Data | <i>Mobile agent, agent execution request, and agent transportation request are exchanged.</i> |
| Structure | <i>Agent hosts contain an interface for agent execution, agent transportation request, and interfaces for agent transportation, and agent arrival. Virtual machines contain an interface for interpretation expression and interpretation results.</i> |
| Interaction | Agent hosts do not block for agent transportation requests or agent arrival. Virtual machines do not block for interpretation expression. |
| Behavior | Virtual machines reply to interpretation expression through an interpretation result. Upon receiving an agent transportation request, an agent host immediately sends mobile agent to another agent host. |
| Topology | An agent host's agent execution interface is connectable to a virtual machine's interpretation expression interface, a virtual machine's interpretation result interface is connectable to an agent host's agent transportation request interface. An agent host's agent transportation interface is connectable to another host's agent arrival interface. Each agent host has a connected virtual machine. |

Table 15: Event-based interaction style characteristics

| Characteristic | Event-based interaction style (EBI) |
|-----------------------|---|
| Data | <i>Events, subscription, and unsubscription</i> requests are exchanged. |
| Structure | <p>A <i>subscriber</i> contains an interface to <i>manage</i> subscription and unsubscription of events, and to <i>consume</i> events. For each interface to consume events, the subscriber must have a subscription management interface.</p> <p>An event <i>distributor</i> contains interfaces to <i>accept</i> events, <i>manage subscription</i>, perform <i>unsubscription</i>, and <i>distribute</i> events. For every interface to distribute events, the event distributor must have a subscription and an unsubscription interface.</p> <p>A <i>publisher</i> contains an interface to <i>publish</i> events.</p> |
| Interaction | Publishers do not block upon publishing events. Event distributors do not block for accepting events, managing subscription, performing unsubscription, or upon distributing events. A subscriber does not block while listening for events. |
| Behavior | An event distributor can perform unsubscription or distribute events only after receiving a subscription from a subscriber. Upon accepting an event from a producer, an event distributor distributes the event to subscribers listening for that event. After performing unsubscription, an event distributor cannot distribute events to that subscriber. |
| Topology | <p>A publisher's <i>publish</i> interface is connectable to a distributor's <i>accept</i> interface.</p> <p>A subscriber's <i>subscription</i> interface is connectable to a distributor's <i>subscription</i> interface, its <i>unsubscription</i> interface is connectable to a distributor's <i>unsubscription</i> interface, a distributor's <i>distribute</i> interface is connectable to a subscriber's <i>listen</i> interface.</p> <p>There is exactly one distributor.</p> <p>Each publisher is connected to one distributor.</p> |

Table 16: C2 style characteristics

| Characteristic | C2 style (C2) |
|-----------------------|--|
| Data | <i>Notifications and requests</i> are exchanged. |
| Structure | <i>Components</i> contain an interface for <i>producing</i> and <i>consuming notifications and requests</i> . For every produce notification interface, a component must have a consume request interface, and vice versa. All components should either have produce notification and consume request interfaces or produce request and consume notification interfaces or both. <i>Connectors</i> contain interfaces for <i>producing</i> and <i>consuming notifications and requests</i> . For every produce notification interface, a component must have a consume request interface, and vice versa. |
| Interaction | Components and consumers do not block for consuming requests and notifications. |
| Behavior | Connectors broadcast all notifications received on each of their consume notifications interfaces to all their produce notifications interfaces, and vice versa. |
| Topology | A component's (connector's) produce notification interface is connectable to a connector's (component's) consume notification interface, its produce request interface is connectable to a connector's (component's) consume request interface. Connected components and connectors cannot form a loop. |

Table 17: Distributed objects style characteristics

| Characteristic | Distributed objects style (DO) |
|-----------------------|--|
| Data | Method invocation <i>parameters</i> and <i>results</i> are exchanged. |
| Structure | <p>An <i>object</i> contains interfaces to <i>invoke</i> methods and obtain <i>results</i> of the invocation, and an interface to <i>process</i> invocations and <i>produce</i> method results. For every interface to invoke methods, an object contains an interface to obtain results of the invocation.</p> <p>A <i>transport</i> contains interfaces to process invocations and produce method results, and an interface to <i>invoke</i> methods and obtain <i>results</i> of the invocation. For every interface to process method invocations, a transport contains an interface to produce results of the invocation.</p> |
| Interaction | Objects block to obtain results of invocation. Objects and transports do not block to process invocations. |
| Behavior | Objects and transports produce a result for every method invocation. They immediately switch to receive a result after invoking methods. |
| Topology | <p>An object's (transport's) invoke interface is connectable to a transport's (object's) process interface. A transport's (object's) produce interface is connectable to an object's (transport's) results interface.</p> <p>Each object is connected to a transport through its process interface, and each transport is connected to a single object through its invoke interface.</p> <p>An object is not connected to itself via a transport.</p> |

Table 18: Brokered distributed objects style characteristics

| Characteristic | Brokered distributed objects style (BDO) |
|-----------------------|---|
| Data | <i>Object location requests, object location, method invocation parameters and results</i> are exchanged. |
| Structure | <p>An <i>object</i> contains interfaces to <i>invoke</i> methods and obtain <i>results</i> of the invocation, to <i>request</i> object location and obtain object <i>locations</i>, and an interface to <i>process</i> invocations and <i>produce</i> method results. For every interface to invoke methods, an object contains an interface to obtain results of the invocation.</p> <p>A <i>transport</i> contains interfaces to <i>process</i> invocations and <i>produce</i> method results, and an interface to <i>invoke</i> methods and obtain <i>results</i> of the invocation. For every interface to process method invocations, a transport contains an interface to produce results of the invocation.</p> <p>A <i>broker</i> contains an interface to <i>register</i> and <i>process</i> object location requests and to provide object <i>locations</i>.</p> <p>A <i>broker transport</i> contains interfaces to <i>process</i> location requests and <i>produce</i> object locations, and an interface to <i>request</i> object location and obtain the object <i>location</i>. For every interface to process location requests, a broker transport contains an interface to produce results of the invocation.</p> |
| Interaction | Objects and transports block to obtain results of invocation. Objects and transports do not block to process invocations. Brokers and broker transports do not block for location requests. Broker transports block to obtain object locations. |
| Behavior | Objects and transports produce a result for every method invocation. They immediately switch to receive a result after invoking methods. Brokers and broker transports produce an object location for every location request. |
| Topology | <p>An object's (transport's) <i>invoke</i> interface is connectable to a transport's (object's) <i>process</i> interface. A transport's (object's) <i>produce</i> interface is connectable to an object's (transport's) <i>results</i> interface.</p> <p>Each object is connected to a transport through its <i>process</i> interface, and each transport is connected to a single object through its <i>invoke</i> interface.</p> <p>An object's (broker transport's) <i>request</i> interface is connectable to a broker transport's (broker's) <i>process</i> interface. A broker transport's (broker's) <i>location</i> interface is connectable to an object's (broker transport's) <i>location</i> interface.</p> <p>Each broker is connected to a broker transport through its <i>request</i> interface, and each broker transport is connected to a single broker through its <i>request</i> interface.</p> |