


Services

Agenda

- Overview
- Component services

What is a Service?

- In the software sense, a service is an application created primarily or solely to be called by other programs 

- Programs that uses the service are called clients

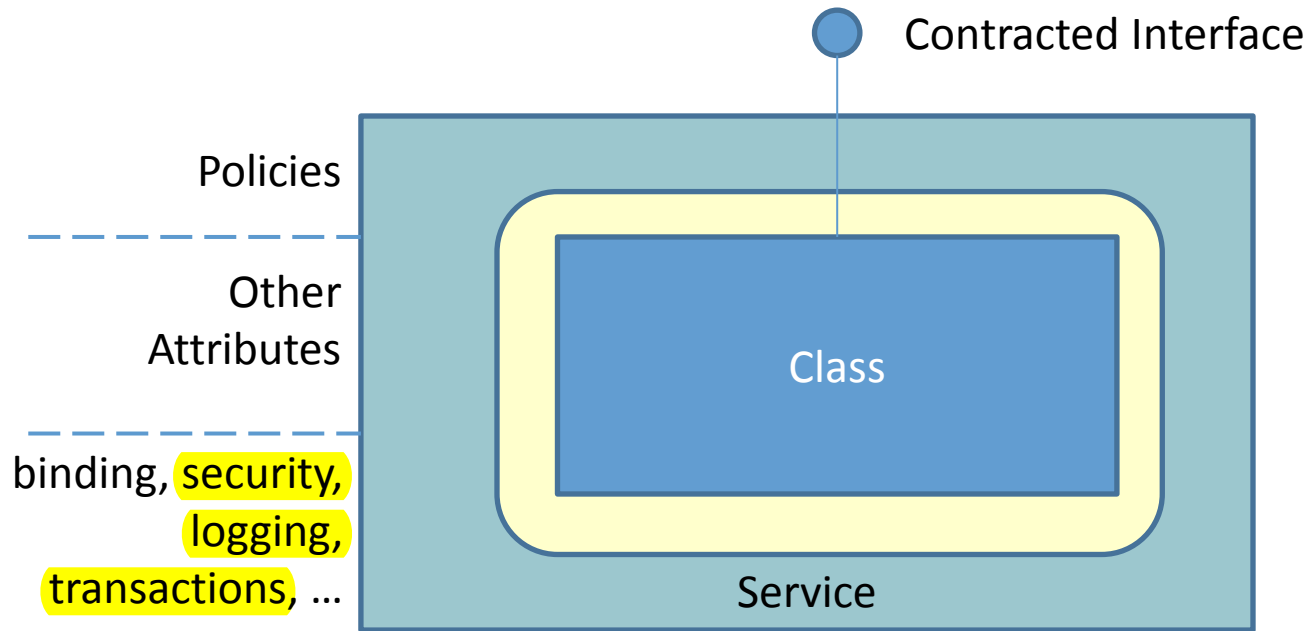
- From Wikipedia:

*The term **service** refers to a discretely defined set of contiguous and autonomous business or technical functionality.*

- From The Advancement of Structured Information Sandards (OASIS):

A mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.

Service in a Nutshell



*From:
Microsoft .Net
Architecting
Applications for the
Enterprise,
By D. Esposito and
A. Saltarello*

- **A service is a class with something wrapped around it.**
- This “something” is typically a run-time environment that provides additional functionalities like:
 - Searching, binding, security, logging, transactions and messaging.



Business Services



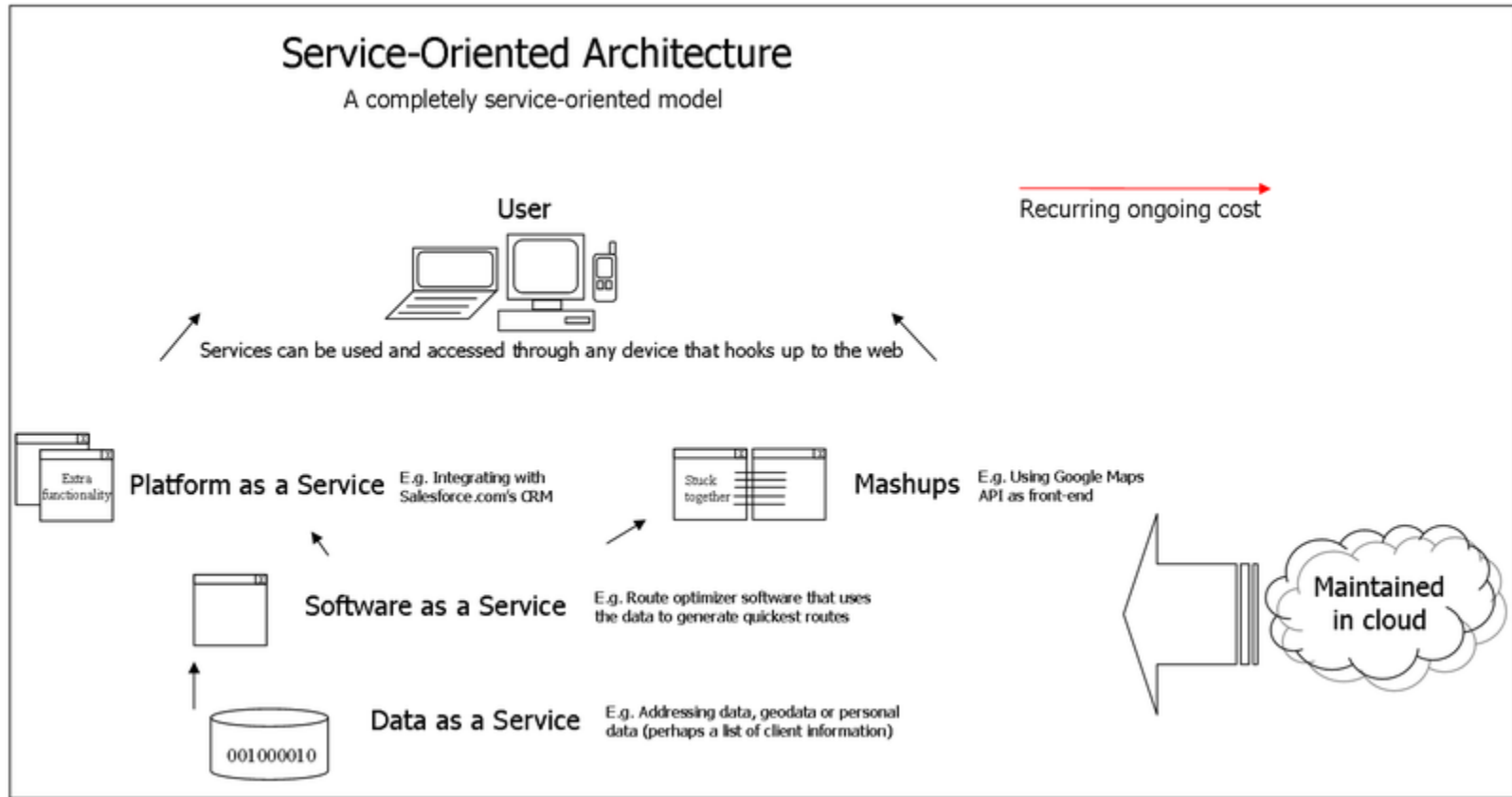
- Service Engineering:
 - An Enterprise Architecture Team will develop the organization's "Service" model first by defining the top level Business Functions.
 - Once the Business Functions are defined they are decomposed into "Services" which represent the processes and activities needed to manage the assets of the organization in their various states.
 - An example here would be the decomposition of
 - Business Function:
 "Manage Orders"
 - into "Services" such as:
 - "Create Order",
 - "Fulfill Order",
 - "Ship Order",
 - "Invoice Order",
 - "Cancel/Update Order",
 - etc.

SOA

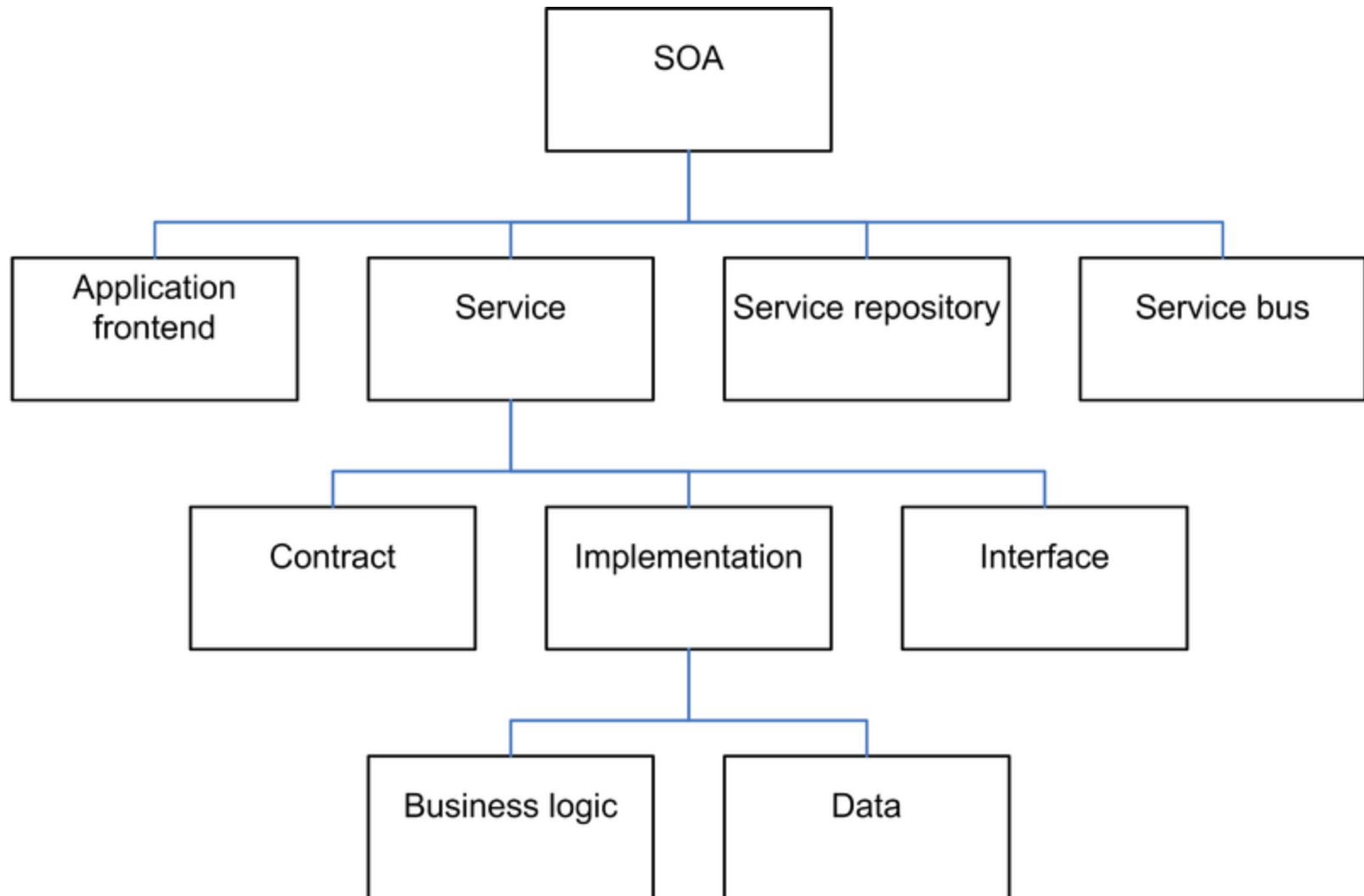
From Wikipedia:

- **Service-oriented architecture (SOA)** provides methods for systems development and integration where systems group functionality around business processes and package these as *interoperable services*.
- SOA also describes IT infrastructure which allows different applications to exchange data with one another as they participate in business processes.
- **Service-orientation aims at a loose coupling**  services with operating systems, programming languages and other technologies which underlie  applications.
- SOA separates functions into distinct units, or services, which developers make accessible over a network in order that users can **combine and reuse them in the production** of business applications.
- **These services communicate with each other by passing data from one service to another,** or by coordinating an activity between two or more services.
- Many see SOA concepts as built upon and evolving from older concepts of distributed computing and modular programming.

SOA Architecture



SOA Elements



The Different Kinds of Technical Services

Here we will define 3 kind of services

*Not part of KPU1,
but ONK*

- **Remote services** (distributed applications)

Are accessed over the network and are often required to support many concurrent clients. They can be implemented by use of low level TCP/IP programming, but in .net the main technologies used are:

- **Web services**
 - .Net Remoting
 - Enterprise Services
- } WCF

*Can also be
used locally*

- **Local services**

Are generally designed to support only applications running on the same machine or the user.

- **Windows service** 

*Can also serve
remote clients*

- **Component services**

Are low level services to be used by components:

- ContextBound objects
- Enterprise Services

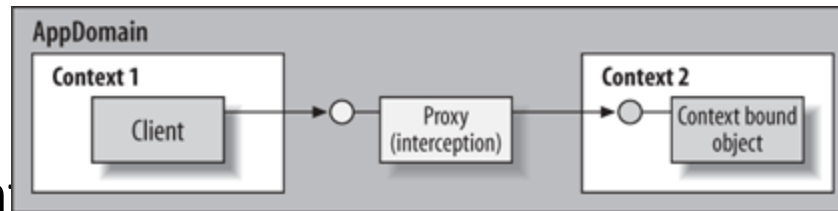


COMPONENT SERVICES




What are Component Services?

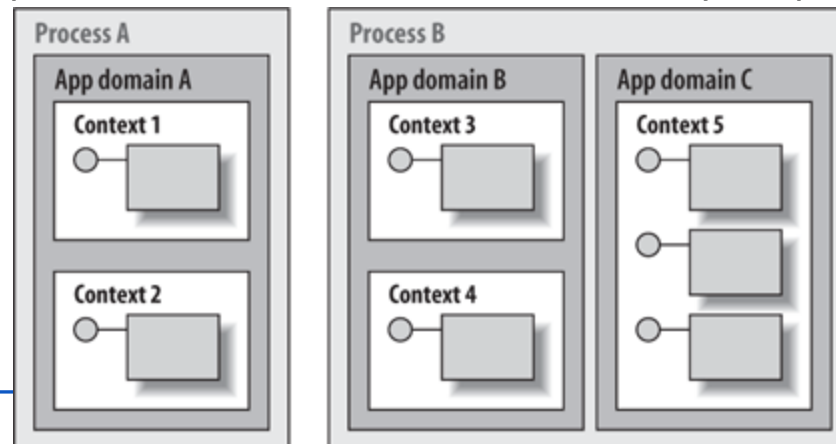
- They are services the framework/operation system offer to components.
- .NET provides component services via call interception.
- To intercept a call, .NET must insert a proxy between the client and the object and do some pre- and post-call processing.



- Context and Interception
 - Interception can only take place when calling from one context to an object in another context.
- .NET comes with a predefined set of component services
 - But you can extend the set of services, as .NET allows you to provide your own custom component services.

What is Context?

- The context is the innermost execution scope of an object.
- Components indicate to .NET which services they require using special context attributes.
- All objects in the same context are compatible in terms of their component services requirements 
 - So no need for pre- and post-call processing when these objects access one another.
- You can define a context as a logical grouping of objects that rely on the same set of services.
 - Calls into the context are intercepted to ensure that the objects always get the appropriate runtime environment they require to operate.



Contexts and Object Types

- .NET objects are classified into two categories:
 - those that care about component services and want to use them
 - and those that don't.
- An object that wants to use component services is called a context-bound object.
- Context-bound objects always execute in the same context.
- The designation and affinity of a context-bound object to a context is decided when the object is created and is fixed for the life of the object.
- To qualify as a context-bound object, the object must derive directly or indirectly from the abstract class

ContextBoundObject:

```
public class MyClass : ContextBoundObject  
{...}
```

Component Services Types

- .NET provides two kinds of component services:
 - context-bound services
 - Enterprise Services.
- The context-bound services are available to any context-bound object.
 - .NET offers only one such service: the synchronization domain.
 - Using the Synchronization context attribute, a context-bound component gains automatic synchronization and lock sharing.

```
[Synchronization]
public class MyClass : ContextBoundObject
{
    public MyClass( )
    {}
    public void DoSomething( )
    {}
}
```

.NET Enterprise Services

- .NET Enterprise Services are a set of component services designed to ease the development of Enterprise applications.
- .NET offer more than 20 Enterprise Services.
- .NET components that take advantage of Enterprise Services are called *serviced components* because they derive from the class `ServiceComponent`, defined in the `System.EnterpriseServices` namespace.
- For example, if you want .NET to maintain a pool of your objects, and you want there to be **at least 3 but no more than 10 objects in the pool**, use the `ObjectPooling` attribute:

```
using System.EnterpriseServices;  
[Transaction]  
[ObjectPooling(MinPoolSize = 3,MaxPoolSize = 10)]  
public class MyComponent : ServiceComponent  
{...}
```

You can also configure the pool parameters (and any other services) after deployment by using the COM+ Component Services Explorer

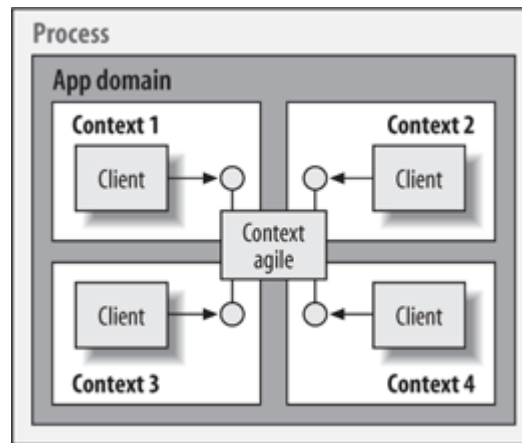
The .NET Context

- Every new app domain starts with a single context, called the *default context*.
 - The default context provides no component services at all!
- There is no limit to the number of contexts an app domain can contain.
- A given context belongs to exactly one app domain.
- A context can host multiple context-bound objects.
- Every context has a unique ID (an integer) called the *context ID*.
- *Every .NET context has a context object associated with it*
 - But you don't need to interact with the context object.
 - Every object can access the context object of the context it's executing in by using the `CurrentContext` static read-only property of the `Thread` class

Assigning Objects to Contexts

There are two kinds of .NET types:

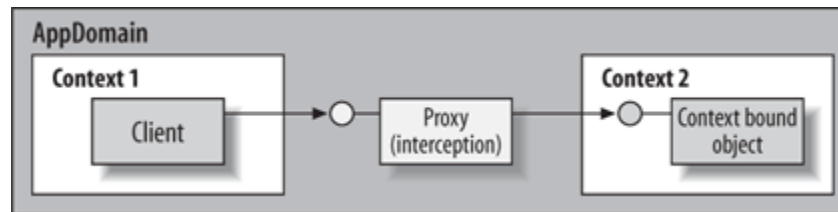
- Context-agile
 - The default behavior in .NET
 - Context-agile objects have no interest in component services
 - They can execute in the contexts of their calling clients because .NET doesn't need to intercept incoming calls to them.



- Context-bound
 - Context-bound objects must derive directly or indirectly from the abstract class `ContextBoundObject`:

Context-bound Objects

- Are bound to a particular context for life.
- The decision regarding which context the object resides in takes place when the object is created:
 - If the creating client's context is "good enough" for the object's needs then the new object is placed in its creating client's context.
 - If the object requires some other service that the creating client's context doesn't support, then .NET creates a new context and places the new object in it.
 - Note that .NET doesn't try to find out if there is already another appropriate context for the object in that app domain.
 - The algorithm is simple: the object either shares its creator's context or gets a new context.

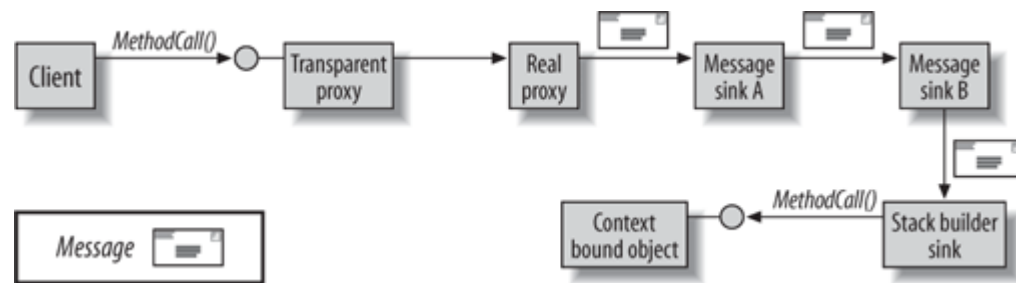


The Call Interception Architecture

- In .NET, the proxy has two parts:
 - a transparent proxy and a
 - real proxy.
- The transparent proxy exposes the same public entry points as the object.
- When the client calls the transparent proxy, it converts the stack frame to a message and passes the message to the real proxy.
- The real proxy knows where the actual object resides.
- The real proxy doesn't know about formatters, channels, or context interceptors
 - it simply passes the message to a message sink.

Cross-context sinks

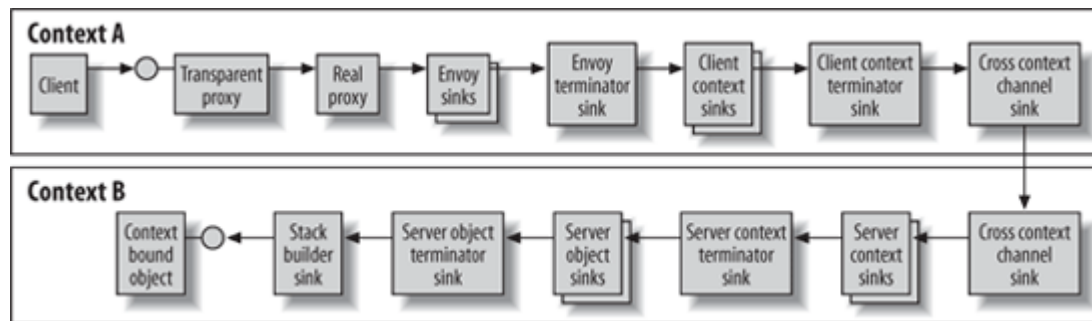
- In the case of a cross-context call .NET uses an internal channel called CrossContextChannel, which is also a message sink.
- However, there is a difference in component services configuration between the client and the object, and it's up to the sinks to compensate for these differences.
- .NET installs as many message sinks as required between the client's context and the object



- *The .NET context interception architecture is similar to the Decorator design pattern and is a private case of aspect-oriented programming.*

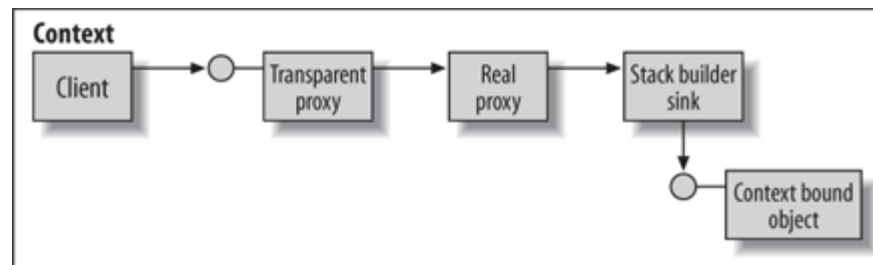
Client-side and server-side sink chains

- Server-side sinks intercepting all calls into the context.
- Server-side sinks intercepting calls to a particular object are called server object sinks.
 - The server is responsible for installing server-side sinks.
- Client-side sinks are installed by the client
 - and they affect all calls going out of the context.
- Client-side sinks installed by the object are called envoy sinks.
 - An envoy sink intercepts calls only to the particular object with which it's associated.
- The last sink on the client's side and the first sink on the server's side are instances of the type `CrossContextChannel`



Same-Context Calls

- What happens if a client in the same context as the object passes a reference to the object to a client in a different context?
 - If the same-context client has a direct reference to the object, how can .NET detect that and introduce a proxy between the object and the new client?
- .NET solves the problem by always having the object accessed via a proxy, even by clients in the same context.
- When the same-context client passes its reference to the transparent proxy to clients in other contexts, .NET detects that and sets up the correct interception chain between the new clients and the object.



References

- SOA

http://en.wikipedia.org/wiki/Service-oriented_architecture

- System.EnterpriseServices Namespace

<http://msdn.microsoft.com/en-us/library/vstudio/9b84cbdx.aspx>