SOA without API

By Sergey Kucherov
(March 22, 2005)

Service Oriented Architecture (SOA) and web services will not be able to support stable application architecture. Not without a consistent programming environment with clear set of limitations and rules called Application Program Interface (API).

It's not that I'm so smart, it's just that I stay with problems longer.

Albert Einstein

Service Oriented Architecture has conquered minds of middle management and now it having a victorious march through directors and executives. Evidently, it is too late to stop it, because SOA is the new buzz-wave, which has to serve its purpose as many ones before it. However, if you do not want to sit and watch this big wave crash the shore, there is something you can do.

First, we need to find out what is the problem (if there is one). Let us do it by going back in time and see how the game gets started.

Object-Oriented Programming

It was almost exactly 20 years ago when a new paradigm in the software programming started its ascending. It was called the Object-Oriented programming (OOP) and it promised us better code, fewer errors, reusability, and painless application design.

Actually, object-oriented programming was (and remains) an easy concept to grasp for most software developers. The idea is simple enough – to divide complex algorithms by creating objects, which do all the work. Objects are combination of data and functionality. They are isolated from each other, they are aware of their state, and each object belongs to a class.

There are two most important goals of software developer – reduce dependencies and reuse the code. In my opinion, OOP was the greatest innovation in the history of software engineering, because it has changed the way programmers approach these two goals.

Object-oriented languages enabled software developers to reduce dependencies between functions and data by encapsulating it in objects. Inheritance, from the other side, has provided them with easy way to reuse code.

The most successful case for reusing of object-oriented code was provided by companies like Microsoft and Borland who have developed large class libraries creating a foundation for developing robust DOS and Windows applications. Libraries like Turbo Vision, MFC, and VCL provided an application programming interface (API), which created object-oriented layer between application and the operating system.

However, outside of these widely used APIs, very few classes were consistently reused. Programmers usually have more reasons to develop a new class instead of reusing the existing one. The advantages of OOP start fading once you move farther away from the foundation API.

Components

Approximately ten years and ten millions lines of code later, desperate software developers decided they need a new flavor of the OOP. They were looking for objects, which are more predictable and reusable in complex development projects (the ones with databases, networks, and advanced user interfaces.) The new type was found – it called components. A component is an object, which you can test before put them into your program.

Components have moved us one more step farther toward reducing dependencies, because components are language independent. For example, you can call a component written in C++ from the program written in Visual Basic.

For the last ten years, we have seen evolution of both components and class libraries. We have visual components, server components, and distributed components. It is safe to say that components have changed the face of modern application architecture.

There is only one promise components have failed to deliver – code reusing. Again, as in case of class libraries, with exception of the well-structured component-based APIs, it is still much safer to write new code instead of using than use existing one.

Service-Oriented Architecture

Another ten years flies by, and it is time for the new wave of the object/component reusing ideas. This one called Service-Oriented Architecture (SOA). Armed with new weapon called web services, SOA once again promises… yes, you are right – reducing dependencies and increasing of code reusability. But will it deliver?

You will find variety of definitions, explanations, speculations, and contradictions about what SOA is and what it is not. [1,2] Allow me to suggest the simple definition:

The application or system has Service Oriented Architecture if:

a) It prefers to execute functions by invoking web services rather than calling an internal code.

b) If no web service is available for the function, then SOA application implements it internally, but will consider exposing it as a web service.

It is easy to see, that web services liberate software from one more dependency burden – web services are platform independent. You can applications running on AS/400+Java and Windows 2003+.NET talking to each other using SOAP protocol.

However, as the two generations before it, web services and SOA will not move us one inch closer to the goal of reusing source code.

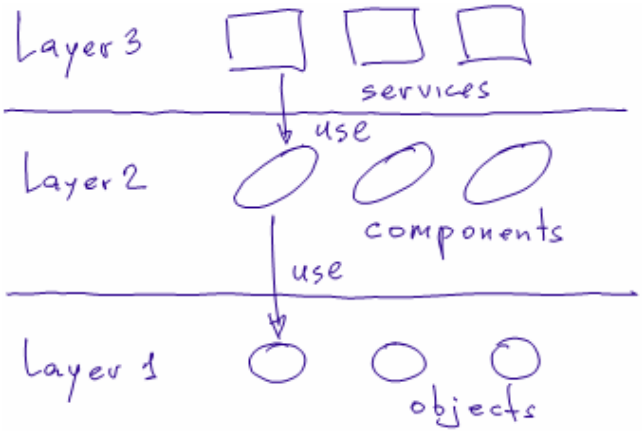
Application Programming Interface

If you have a car, then you probably can drive it wherever you want. All you need is gas, map, and a good road. However, if you want to use public transportation such as bus or train, then you need additional information to plan your trip. You need to locate stations, learn schedules and fares, and buy tickets.

This is just additional troubles for user. There much more work for a transportation agency, which needs to build the stations, plan the schedules, and control the traffic. Any attempt to share public transport without thorough organization will lead to a disaster.

The very same fate is waiting for any attempt to share and reuse of software components without providing a consistent programming environment with clear set of limitations and rules. The public specifications of such environment called Application Programming Interface (API).

An API is an essential part of Service-Oriented architecture, as well as any other software architecture, because it can be used as independent logical layer:



Individual components and objects, which are part of the API, are obeying its rules, not defining them. Here is an example: say you have a class, which can print data in table format. You may consider using this class in your program to output query results to printer. However, you do not know if this class can handle additional features like different colors, line styles, and fonts.

Now, you know that all users have Microsoft Office installed. Excel classes are available through COM interface, and you know that Excel can handle any kind of formatting. As result, you will more likely to use Excel API to create and print spreadsheets, than rely on a standalone class which can only do so much.

Many SOA evangelists and managers believe that they can just plug together a bunch of loosely coupled web services to get a nice Service-Oriented architecture up and running. They may even succeed, but not before the design included a strong, completed, and consistent outline of all web services, which will be used by the application – API.

So, if it is your job to make the SOA application work, then make sure the web services API is the part of the design. Because I assure you, that no matter how good the design is, without API any Service-Oriented Architecture is designed to fail.

References

[1] Steve Swartz - Talking about SOA
http://channel9.msdn.com/Showpost.aspx?postid=56393

[2] What is Service-Oriented Architecture?
http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html