

COMPONENTS AND SOA

WHITE  
PAPER

## COMPONENTS AND SOA

While service oriented architecture (SOA) has become a ubiquitous buzzword, the word component – or object – is not used as widely. Yet an effective SOA-based enterprise application relies on the use of well-defined, modular software components. What is a software component? For that matter, what, really, is an SOA?

SOA essentially implies an application architecture made up of loosely coupled “services” (for example the various software features that are used in creating and processing a customer order), and service “consumers” (the users and applications that need to create customer orders, for example). Most business software applications can, of course, create customer orders. But a business application that is made up of services

allows one to easily connect the processes that need to create customer orders, and easily configure the steps that are taken to actually create the customer order. The loosely couple nature of SOA services also means it’s relatively easy to substitute the components that implement each step in the process. This process tends to be very rigid in non-SOA applications, and it is because of this rigidity that SOA is becoming more popular.

In the middle 1980s, IFS was designing its product in the same way as everyone else – as a large, monolithic, one-size-fits-all block of programming. In the early 1990s, when the concepts of software modularity and re-use became popular, IFS did the only thing you can do with a large monolithic application architecture, which is to abandon it and start over from scratch, using an object-oriented – or component-based – model. This allows for the building and deployment of applications on a very modular level. While the term “module” can be used interchangeably with the term “component,” we will use the word component to describe the granular

pieces of functionality an SOA relies on. The term “module” can perhaps be better-used to describe functional groupings of components in an SOA. For instance, a Work Orders module of an enterprise application might be made up of any number of individual software components.

But what is the precise definition of a component? It would be most accurate to describe both a conceptual definition and a technical definition.

### Conceptual Definition

A component is a small module of software – a set of features that provides some particular functionality used inside of an application. Components can operate on a very low level within an application, doing very mundane things like inserting something into a database or pulling it back out again. But higher-level components that operate closer to the user interface of an application do more meaningful things, representing something tangible like a customer order or a portion of a customer order. Sometimes software designers will refer to these components as documents, which represent not only data but the processes that act on that data. They can perform functions like creating a new customer record or releasing it into a fulfillment process.

On a functional level, components allow for the benefits of a service-oriented architecture because they deliver a level of abstraction, hiding the details of precisely how data is being manipulated, but at the same time offering the manipulated data to other components in a standardized fashion. SOA depends on components with well-defined interfaces, allowing other components to know exactly how to tell it to do things or ask it questions. And SOA delivers agility and helps manage change because the independent nature of components – combined with the well-defined method for communication between components – means that sections of an application can be reconfigured or replaced with new functionality or external applications.

### Technical Definition

From a technological standpoint, components are software objects that interact with each other to comprise an application. The use of components stems from the idea of breaking large problems down into smaller, easier to solve units. By designing modular solutions to smaller problems, one gains greater flexibility in the way these solutions are assembled to tackle larger problems. A Service Oriented Architecture builds on this concept by imposing certain principles that allows these modular components to more readily work together.

For example, a quality SOA must be made up of software components that have a well-defined interface, that allow them to communicate with each other, and at the same time hides the underlying details of how they operate.. The interface between components is what gives them “plug and play” (sometimes called “composable”) functionality with other components and applications. The calling interface between components can also be exposed and assigned an Internet Protocol address, creating software services characteristic of an SOA.

The benefit of having software components that are very granular is that this allows the creation of software services at whatever level in an application hierarchy is required given a specific need. Applications built from components or software objects that lack the appropriate level of granularity will not yield a SOA that delivers the flexibility and agility necessary to react to change quickly. Monolithic applications might support service portals into modules like customer orders or human resources, but the lack of a flexible set of underlying components limits how quickly these applications can be reconfigured to meet new requirements.

### Role of components in SOA

Components allow for the “loose coupling” between services and service consumers necessary for a SOA because they take small elements of functionality and compartmentalize it. It is this highly modular approach that allows components to stand alone and operate as part of a loosely-coupled collection of stand-alone software services.

The individual components are the pieces and elements of functionality that can be exposed as services. Components – and the well-defined way that they communicate with each other – deliver the agility and flexibility that is making SOA the current paradigm in enterprise applications.

Components provide the abstraction necessary for an SOA. What we mean by abstraction is that components hide from the rest of the application the details of exactly how the component is going about undertaking the tasks it performs. The rest of the application is aware only of the rules about how to communicate with the component. These rules are referred to as the software service’s contract – or in more technical terms its calling interface. The information contained in that service contract spells out the basics of how the component fits into the application, What mechanism does the application use to communicate with the component, what data does the application provide to the component and what data does it get back? What does this software service do?

## Granularity of components key

A more monolithic application may use a service contract to hide the details of its functioning and try to expose elements of the application as software services. But those services will be very dependent on other functionality throughout the application, and it will be difficult to build services at the level and with the type of functionality an end user of the application might require.

As an example of how granularity of components and software services allows for the agility that an SOA should deliver, consider the relatively simple process flows around purchasing and purchase requisition. Picture an application that has one set of software components that processes requisitions, one set of components for purchase orders and another set of components that converts a requisition to a purchase order.

In a monolithic architecture, these processes are rigidly bound together. The monolithic application may use parameters and switches to allow these processes to be configured to a degree, but the rules that govern these transactions make them inseparable. In a SOA, each of those steps is made up of various components assembled into individual services, loosely coupled so that the purchase order service does not need to know a lot about the requisition service or the service that transitions the requisition into a purchase order. These services can then be composed into business processes using tools like Business Process Execution Language (BPEL).

BPEL governs the movement of data between one service and the next, controlling it in a highly orchestrated way. Through BPEL or other tools, functionality can be changed and reconfigured – or even replaced with a different application, by linking components in an application orchestration layer. This delivers a lot of flexibility as my business needs and perhaps even my business model changes. As change occurs, the owner of an SOA-based application can recompose their business process in a very agile way.

Using our purchase order / requisition example, the rules used to determine how a requisition gets approved is defined in the process layer, not within the service (or underlying components). This makes it easier to change the rules that govern business processes, because the rules are independent of the software components. It also means whole services can be substituted as needed.

## Composability and industry standards

In order for an SOA-based application to provide services that are composable in this fashion, an application must consist of flexible components to provide the requisite level of granularity to allow reconfiguration, or recomposition.

IFS WHITE PAPER  
COMPONENTS AND SOA

Ideally, the calling interface and method of composing and recomposing components in an SOA should comply with open standards. Proprietary systems for composing an SOA-based application may provide some benefit, but will still lock a system owner into a single vendor and software system.

Some industry standards already exist in some industries and some business processes. As time goes on, industry groups can define standards for the granularity and calling interfaces between components across the scope of enterprise applications. It is these industry standards, rather than proprietary standards put forth by individual vendors, that will ensure that an investment in SOA-based applications will hold its value well into the future. As those standards are set, it will be good news for companies like IFS, as we can easily change the calling interface for our components to comply with new requirements.

Standardization will also be good news for end-users of SOA applications because they will have more choices about where they get specific application service modules. If one vendor provides superior functionality in the area of wholesale distribution, for instance, but IFS is stronger in after market service, standards will make it that much easier to blend disparate systems into a cohesive whole.

By Rick Veague

*Rick Veague is Chief Technology Officer with IFS North America, and is based in the Schaumburg, Ill. headquarters. In this role, Veague provides direction for IFS' use of Service-Oriented Architecture (SOA) and works with IFS' leading customers to leverage SOA to provide state-of-the-art ERP.*

## About IFS and IFS Applications

IFS (XSSE: IFS), the global enterprise applications company, provides ERP solutions which enable organizations to respond quickly to market changes. The solutions allow resources to be used in a more agile way to achieve better business performance and competitive advantage.

Founded in 1983, IFS has 2,600 employees worldwide. With IFS Applications™, now in its seventh generation, IFS has pioneered component-based ERP software. The component architecture provides solutions that are easier to implement, run and upgrade. IFS Applications is available in 54 countries in 20 languages.

IFS has over 500,000 users across seven key vertical sectors: aerospace & defense; automotive; high-tech; industrial manufacturing; process industries; construction, service & facilities management; and utilities & telecom. IFS Applications provides extended ERP functionality, including CRM, SCM, PLM, CPM, enterprise asset management, and MRO capabilities.

If you need further information, e-mail to [info@ifsworld.com](mailto:info@ifsworld.com), contact your local IFS office or visit our web site:

**[www.IFSWORLD.com](http://www.IFSWORLD.com)**

This support offer has been made in order to respond to the requirements of IFS' customers. Since the customers' requirements may be different in some markets, variations of this offer may exist.

IFS and all IFS product names are trademarks of IFS. The names of actual companies and products mentioned herein may be the trademarks of their respective owners. The example companies, organizations, products, domain names, email addresses, logos, people and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person or event is intended or should be inferred.

This document may contain statements of possible future functionality for IFS' software products and technology.

Such statements of future functionality are for information purposes only and should not be interpreted as any commitment or representation.