# Service-Orientation and Object-Orientation: Complementary Design Paradigms

**Modern Software Development Series**

Gary Stubbings BSc MBCS

Faculty of ACES
Sheffield Hallam University

## Abstract

Modern software development techniques evolve around the use of object-orientation (OO). However, there is an increasing drive to develop applications using distinct services that facilitate the reuse of already developed business processes through functions rather than objects. This paper discusses the benefits and influence of these paradigms with respect to how consideration should be placed on both when carrying out software design.

## Programming Methods

OO is a programming paradigm whereby state and behaviour are grouped together into real-world artefacts. A core advantage of OOP over procedural programming techniques is that it enables the encapsulation of behaviour within its structure. Service-oriented architecture (SOA) is a development method whereby business processes are decomposed into distinct modular units of work known as services. Although SOAs have a number of characteristics the most significant is to achieve loose coupling among interacting software agents in order to achieve non-platform dependant interoperability (Rosen, Lublinsky, Smith, & Balcer, 2008). The advantage of this is that it enables connectivity to legacy systems effectively facilitating the reuse of existing business processes.

There are many different schools of thought regarding the comparison of these methods, however, less subjective practitioners consider them as independent approaches as they work on different levels of abstraction to solve particular problems and can even both be used within the same solution. Furthermore, Erl (2007) outlines that the principals and patterns behind OO represent the significant sources of inspiration for SOA as many of the core concepts such as encapsulation, reusability and abstraction can be traced back to their OO counterparts.

A common distinction between these two paradigms considers that the OO is a style of programming that does not describe software architecture whereas SOA pertains to both software architecture and a business strategy through the use of technologies such as web services. In recent years SOA and SO have managed to achieve mainstream status which is mainly due to the emergence and successes of the web services framework (Erl, 2008).

Web services make functions available over standard internet protocols and although other technologies such as remote services[1] exist in distributed programming approaches it can be argued that SOA offers more flexibility when it is built entirely with web services (Manufacturing Business Technology, 2005). However, distributed computing approaches do present their own advantages through an increase in performance and a focus towards well defined OO principles whereby a clear modular structure is provided to enable support for abstract data types and encapsulated implementation of logic behind clearly defined interfaces.

[1] *OMG's Common Object Request Broker Architecture (CORBA) and Microsoft's Distributed Component Object Model (DCOM).*

## Current Approaches

One of the biggest problems in software development is that it is difficult to choose the best approach when both business requirements and technology are continuously changing. Marks (2008) suggests that the primary purpose of SOA within a business environment is as a means of connectivity rather than replacement of technology as it promotes the alignment of both requirements and technology by exposing existing functionality and enabling the development of  independent services in response to specific business needs.

Decomposition is the simplest approach to extending OO systems whereby existing functionality is exposed as a set of services for reuse in other parts of the business (Koskela, Rahikainen, & Wan, 2007). Through a combination of approaches SOA and OO can enable the orchestration of requirements without needing to worry about platform, however, the effectiveness of this is often dependent on how well defined the existing layers of abstraction are and even when successful is often criticised for increasing complexity. This is particularly true when it is deemed necessary to encapsulate security and provide state for partners through the use of application programming interfaces (APIs).

A typical approach for SOA introduces a new layer of abstraction that enables new and existing OO systems to be integrated into the business through an Enterprise Service Bus (ESB). The role of an ESB is to act as a universal connectivity middleware solution which enhances communication and simplifies integration (Rosen, Lublinsky, Smith, & Balcer, 2008). ESBs are a popular business solution due to their ability to retain existing investment in resources whilst providing additional tools[2] for interaction with other business processes.

## Complementary Paradigms

It is common concern that SOA is a difficult approach due to its complexity and each solution having its own unique requirements. However, as well as service frameworks a number of OO adopted concepts can also be used to address these issues. In some cases SOA solutions make use of APIs or provide data service layers (DSL) in order to address difficulties associated with service implementation and provide support for OO principals such as handling persistence and providing state for object-relational mapping (ORM)[3].

[2] *Such as that of Business Process Execution Languages (BPEL).*

[3] *A tool for linking objects to data sources enabling real-time transactions.*

ORM technologies such as that of Hibernate (Hibernate, 2009) support the ability to modify values in memory using OO constructs which directly communicate updates to data sources. Typically this functionality goes unutilised in SOA due to its stateless behaviour and can only be achieved by passing the persistence to a dedicated layer. A number of practitioners that follow this method of development begin to see the advantages from both paradigms, however, though lack of standards and methodology this approach is not without challenges and presents new complexities such as service to object mapping.

Although it is clear that SOA is not a solution for every challenge, Marks (Marks, 2008) suggests that SOA offers a great deal of business value when applied to the right area such as through the use of external services as support mechanisms for connectivity to suppliers and partners'. This kind of service often serves to save time in development and provide seamless transactions between different companies in multiple locations whilst reducing the amount of resources required by moving focus away from new development and more towards service integration.

The integration of services into OO systems is becoming more common and although this is no new concept it can be argued that market trend has moved towards a greater use of external services in line with the advancement of telecommunication capabilities. However, a common problem in existing OO systems is that service integration is seldom considered in the OO design stages.

## Object-Orientation's Contribution to Service-Orientation

It can be argued that much of what seems to be modern innovation is actually the rediscovery of existing approaches and as a result of previous software development many of the benefits of SOA have already been addressed (Zdun, 2008). All of these existing methodologies underpin the need to consider software development and reusability on different levels by focusing upon the layers of abstraction within SOA, however, it could be considered that this focus should not only include SO but OO as well in order to help modernise the basic concepts from which others are built.

SOA is often considered an evolution of OO as it shows hereditary concepts such as abstraction which is exhibited through strict service separation as well as encapsulation of implementation detail (Erl, 2008). Although both paradigms adhere to different programming models OO analysis is often used to develop semantic models. A business perspective considers that SOA evolved from OO to address a new problem in the industry and to adapt to current technology (Koskela, Rahikainen, & Wan, 2007); however, it can be argued that in fact SO and OO complement each other to support SOA and that despite their differences SO still retains many of the original advantages of OO in contrast to earlier development methods.

## Contrast between Object-Orientation and Service-Orientation

Fundamentally, in OO the objects are aware of each other's existence and are intelligent in that they actively work together to complete business tasks. As a result OO is often used for building the internals of applications while SO encourages a combination of a number of external services where functionality is decoupled and interaction is relatively straightforward.

Although SO and OO have many of the same concepts such as association and granularity it is clear that they also have their own unique aspects and methods of handling remote invocation and as a result it can be argued that the context in which these principles are used is somewhat different (See Table 1).

| Contrast | Object Orientation | Service Orientation |
|---|---|---|
| **Concepts** | Modelling, Architectural Design, Programming | Modelling, Architectural Design |
| **Exposure** | Methods | Services |
| **Focus** | Component-level | Business-level |
| **Communication** | Primarily internal | Internal and external (Interoperable) |
| **Standards** | Extensive standards with high maturity | No standards for specific design patterns |
| **Complexity** | Medium to high with a more controlled environment | High, specifically where there is little control over technology |

*Table 1 – A contrast between OO and SO*

Although these two paradigms can be used alongside each other to develop semantic models it can be argued that most of the existing approaches and frameworks show a clear separation of these paradigms whereby focus is placed on either one or the other and rarely the two together.

## Existing Patterns and Approaches

A number of frameworks and standards for services have emerged from the evolution of software practices. Decision modelling is an approach adopted from software engineering concepts which has led to the development of SOAD (SOA Decision Modelling), a set of concepts and RADM (Reusable Architectural Decision Model), a set of recurring decisions for SOA (Zimmerm, 500 Recurring Decisions for SOA, 2009). However, although these frameworks help with the identification of business requirements it can be argued that they are too generalised and will not fit every solution.

As well as these techniques the IT Infrastructure Language (ITIL) provides an extensive set of concepts and best practices for service management (OGC ITIL, 2008), nevertheless, although it is clear that ITIL has many advantages it can be argued that the approach is more focused towards IT management and that through its use many businesses end up missing pragmatic solutions. This is underpinned by many implementers accusing ITIL as over-complicating requirements where simple solutions are available.

A number of patterns that are based on software engineering approaches such as that of command (Gamma, Helm, Johnson, & Vlissides, 1994) and object system layer (Goedicke, Neumann, & Zdun, 2001) provide structure through standards for development and are commonly used to resolve complexities found in the integration process. However, existing studies such as that of Yang et al. (Yang & Papazoglou, 2004) argues that these approaches are somewhat lacking for service composition and in later works (Papazoglou & Heuvel, 2007) unifies these principles and concepts to promote an approach for extending conventional SOA approaches.

## Future Consideration

Evidence shows that both paradigms have a place in modern system development, however, it can be argued that because of an enterprise-centric perspective which focuses more towards business and management needs only a subset of OO principles exist within SO. This perspective has arguably shaped SOA into a business solution with focus on what is happening right now as opposed to what might happen tomorrow. Further investigation could outline more detail regarding the relationship between OO and SO with focus towards identifying a cross-section of their distinct advantages and the origins of their principles.

A number of practitioners believe that OO still remains the most suitable option for designing application components (Dori, 2007) and that through experience OO along with its ensuing practices[4] still has much to offer SO. Related studies outline that software engineering techniques traditionally used for OO still have many approaches that could be adapted for use in service development and as a result it can be argued that the exploration of existing and standardised approaches with consideration of these two paradigms together rather than separately could lead to the development of a simpler integration method that supports service ready object-oriented systems.

## References

Dori, D. (2007). SOA for services or UML for objects: Reconciliation of the battle of giants with Object-Process Methodology. *International Conference on Software – Science, Technology and Engineering.*

Erl, T. (2008). Service-Orientation and Object-Orientation: A Comparison of Design Principles. *SOA Magazine* (XVI).

Erl, T. (2007). *SOA Principles of Service Design.* Prentice Hall.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Goedicke, M., Neumann, G., & Zdun, U. (2001). Object system layer. *Pattern Languages of Programs.* Irsee.

Hibernate. (2009). *Relational Persistence for Java and .NET*. Retrieved 2 21, 2010, from Hibernate: https://www.hibernate.org/

Koskela, M., Rahikainen, M., & Wan, T. (2007). *Software development methods: SOA vs. CBD, OO and AOP.* Helsinki University of Technology.

Manufacturing Business Technology. (2005). BEA showcases new SOA platform in China. *Manufacturing Business Technology , 23* (2), 49-49.

Marks, E. A. (2008). *Service-Oriented Architecture (SOA) Governance for the Services Driven Enterprise: Business, IT, and Funding for the Evolving Business.* John Wiley & Sons .

OGC ITIL. (2008). *Service Strategies.*

[4] *Such as Design Patterns, Component-Based Development and Aspect-Oriented Programming.*

Papazoglou, M. P., & Heuvel, W.-J. v. (2007). Service oriented architectures: approaches, technologies. *The VLDB Journal* , 389 – 415.

Rosen, M., Lublinsky, B., Smith, K. T., & Balcer, M. J. (2008). *Applied SOA: Service-Oriented Architecture and Design Strategies.* Wiley.

Yang, J., & Papazoglou, M. P. (2004). Service components for managing the life-cycle of service compositions. *Inf. Syst.* , pp. 97 – 125.

Zdun, U. (2008). Pattern-Based Design of a Service-Oriented Middleware for Remote Object Federations. *ACM Trans. Intern. Tech , 8* (3).

Zdun, U. (2006). Patterns of component and language integration. *Pattern Languages of Program Design* .

Zimmerm, O. (2009). *500 Recurring Decisions for SOA*. Retrieved 11 2009, from http://soadecisions.org/soad.htm