What does Service-oriented Computing really mean?

by Dominik Kuropka
Dagstuhl, Germany 2005
Seminar on SOC (05462)

If you take a closer look at current service-oriented architectures and their standards like Web-services and compare them to other distributed computing approaches, then you will come to the following conclusion: Technically SOC does not provide any new possibilities or solutions which were not already available or implementable with the old approaches. It would be for example not a problem to implement most services by using for example CORBA. For this reason we have to conclude that the major innovation in SOC is not a technical one, but more a philosophical one. The philosophical innovation is the move from the object-oriented paradigm to a service-oriented one.

To understand the innovation of SOC, firstly the difference between object-orientation and service-orientation has to be understood. Both paradigms encapsulate functionality in a similar way. In object-orientation functionality is encapsulated by object or class definitions providing methods. In service-orientation functionality is provided by procedures of a service. Furthermore both paradigms are message oriented. The major philosophical difference between both approaches is the handling of states. Objects are designed to hold a state (represented by their attributes) and are therefore called stateful. In contrast to this services are not explicitly designed to hold a state, for this reason they are intended to be stateless. However in real-world a stateless service without any access the databases or memory is not very useful since this would mean, that all information which are needed to perform a service have to be send to the service each time the service is used. For this reason it is useful to allow services to have access to some more or less external state informations. However this should be done with care to be aligned with the service-oriented philosophy. Furthermore services are intended to be tailored according to business needs and not technical issues. This usually lead services to be inherently coarse grained instead of fine grained like objects.

The question to answer is: What is the result of this difference in state handling and tailoring and why is this an innovation? To answer this question, just take a look at object-oriented implementations. Object-oriented approaches tend to represent every thing of interest as an object. This results in a high number of objects which have to be put into relation to each other. For example, if you want to integrate an external system of a producer in your enterprise resource application to create automatically a request for an offer, you have to deal with a lot of objects. To be more concrete: You have to find the offer factory object and request a new offer object. Now you have to repeat the following steps for every desired product to complete your offer: find the right product object, put the product object in relation to the offer object. Finally you have to request a calculate offer method to trigger calculation and you have read attributes like final price by using proper methods to get your data. In the end you have coupled you system tightly with the system of the producer. A lot of messages have to be exchanged to create the offer and a complex procedure has to be kept. Furthermore, if you do not duplicate the data (which is rather unusual for object-oriented approaches) you have to keep a stable connection to the external system each time you want to access the offer. This high coupling and complexity in usage is a reason why object-oriented approaches have been identified as bad solutions for cross-border electronic business.

The philosophy of service-oriented approaches is to make service as stateless, tailored according business issues and large grained as possible and useful. This leads to a lower amount of communication events since the exchanged messages are larger and hold more information, just like in real non-electronic business. The above example implemented with a service-oriented approach would lead to a service with two procedures. One procedure to download a product catalogue and an other to get an offer. The first procedure get either nothing or some criteria for product selection as input. It delivers a large message, the document catalogue as output. By using this catalogue the local system can select the needed products and put their numbers into a list. This list has to be sent to the

Dagstuhl Seminar Proceedings 05462
Service Oriented Computing (SOC)
http://drops.dagstuhl.de/opus/volltexte/2006/522

get offer procedure of the service. This procedure creates and calculates an offer and sends back the whole offer as a document message. So in contrast to the object-oriented approach there is no complex choreography and there is no need for sending a large amount of micro-messages for method invocations. Instead in the service-oriented approach only a small amount of large document messages has to be exchanged. This enables a decoupling of the involved systems. Since they only need to communicate to exchange the few document messages and after the transaction no further communication is needed to get for example access to old data, since relevant documents should always be stored on each side. Finally it should be mentioned that the above mentioned service is not perfectly stateless, since it has access to external data like the product data.

However as often, practice behaves differently to the theory. In context of service-oriented computing this means, that services are often not tailored according business needs. Instead they are tailored by technicians according already implemented object-oriented systems. This results in the fact that existing classes are just wrapped into services to provide a service-oriented interface as cheap and fast as possible. This approach does not lead to real service-oriented computing nor it leads to any benefits like loose coupling of systems. The reason for this is that in such approaches there is still a need for exchange of a large amount of micro-messages to communicate with the now service-encapsulated objects and the services are fine grained. This is not the intended use of service-orientation, rather this ends up in something what can be called as “CORBA through port 80”.

The following statements summarizes the key message of this abstract:

- Service-oriented Computing means that services are coarse grained, ideally stateless and tailored according business needs. This leads to a loose coupled and document oriented communication.
- Simply using service-oriented architectures for implementation does not necessary lead to “proper” Service-oriented Computing.
- Every functionality can be seen as a service. However it is not useful to transfer the idea of service-orientation on every functionality.