As we saw earlier, we cannot avoid changes, and for large applications the domain model may become increasingly fine-grained and complex. Modifying such code to accommodate changes can become very complex, so the need was felt to have a solution that can help us take advantage of the domain model but manage changes in the long run without spending too much time and effort, and help keep our applications flexible and open for future modifications.

Service Orientation

Before understanding the term *service orientation*, let us consider a fictional scenario—Mr. Bhalla owns a small Marketing and Sales company with a few employees. Bhalla realized that his business and clients were growing quickly, and he was unable to handle all of the phone calls and paper work on his own. So he decided to hire an assistant who could do the basic office work, and handle paperwork and other office-related tasks for him. After some interviews he decided to select Ms. Akriti Katoch for the job. Mr. Bhalla was initially a bit skeptical as to whether Akriti could handle the work, so he decided to train her for some time. He told her that she simply needs to follow his instructions carefully. For example, to get a particular file, he would give her details of the file name, file color and the rack where it could be found. Similarly for each task set out to her, he would give her all of the details, so that she could work more efficiently.

For some time, everything went well. But after a while, Mr. Bhalla realized that giving Akriti all of the details was getting cumbersome. For example, if he had to study the files for his client Newport Inc, he would ask Akriti to "*get the first 10 of the blue files on the 5th rack, shelf number 3*". And if, after going through those 10 files, Mr. Bhalla could not find the required information and wanted to see the other files for the same client, he would call Akriti and ask her to "*get the next 5 blue files on the 5th rack, shelf number 3*". This was a waste of time for both Bhalla and Akriti. But as Akriti started gaining experience, he decided to give "less" information to her, and let her do most of the job herself. So he would give her brief instructions, and instead of "*get me the first 10 of the blue files on the 5th rack, shelf number 3*", he would just say "*give me all the files for Newport Inc*". He realized that this method was more efficient, and once all of files were on his desk, he could find the relevant information himself.

Basically, Bhalla moved from a fine-grained model to a coarse-grained one; towards a more "message" based system instead of a method-calling system (give me "this" based on "this, this and this" parameters). This coarse-grained model and message-based system is basically the essence of service orientation.

Service orientation means that the application's business logic is wrapped up and presented as a service to an outside client. This service is complete in itself. It doesn't need complex object relationships or **Component Oriented Middleware** (**COM**)-like middleware to render itself to its consumers. With the advent of web services, SOA architecture has become extremely easy to implement.

Service orientation was born out of this need for better change management, process alignment, and improved efficiency in automating complex and changing business rules. Object orientation focuses more on breaking the business model into an object model and interaction between different business objects according to business rules. It focuses more on how to best implement a particular business model as a domain model in an application.

As OOAD evolved, it became more and more involved with the actual implementation part of the architecture. But with time, the need for better integration and faster response time for business software was felt. It was hard for a new application to communicate with the existing legacy systems (old software), and cross-platform integration became complex and time consuming. A need for an easier solution, that is, an architecture that would solve these problems from a business point of view, and is also easy to implement, was felt. Thus, Service Oriented Architecture was born.

SOA helps businesses to manage changes in their applications better and faster and get a higher Return On Investments (ROI) by promoting the re-use of different application components.

Let us look at the following diagram, which depicts the usual communication between different components in an object-based software system: