1

Understanding Application Frameworks

In the late 1970's, yacc, a program that helps create compilers, was introduced in the UNIX operating system. Before yacc, the construction of compilers was a task that only a few computer science professors around the world could perform. Today, with yacc and other programs such as antlr, computer science students learn to build compilers during their undergraduate classes.

At the present time, only a select few vendors with large teams are able to construct complex application frameworks. Some examples of these application frameworks in the industry are J2EE application servers, web servers, and IDEs (Integrated Development Environments), such as Eclipse.

OSGi will change how complex application frameworks are built in the same way that yacc changed how compilers were constructed. Yacc, a compiler for compilers, allows the compiler author to focus on the specification of the language grammar, which the generated compiler then is able to parse. The OSGI framework, a framework for application frameworks, allows the framework author to focus on the specification of services, which are then assembled together to form new applications.

Maybe in some years from now, computer science courses will teach students how to build application frameworks, and the OSGi technology will be as ubiquitous as yacc, relational table normalization rules, indexing algorithms, and so many other tools and techniques that are part of the computer science curriculum.

This chapter covers:

1. What are application frameworks and what are the benefits of using them.
2. A quick introduction to the OSGi technology.
3. The benefits of creating your own application framework.

4. Why the OSGi technology is the best available technology for the creation of application frameworks.
5. Industry examples of existing application frameworks.

1.1 What are application frameworks?

Before we can understand application frameworks, we need to understand what a framework is.

Defined simply, a framework is the basic supporting or underlying structure of something, whether that structure is a building, a concept, a vehicle, an object, or an idea.

In the case of application frameworks, this *something* is software applications. Examples of software applications are document editors, anti-virus software, and a loan-approval application.

Putting it all together, an application framework is an application targeted to developers whose purpose is to provide a structure for the creation and execution of other applications**.**

There are two aspects to this: a design-time and runtime aspect. From a design-time aspect, application frameworks are development platforms. Developers input code into the framework, and the framework outputs applications.

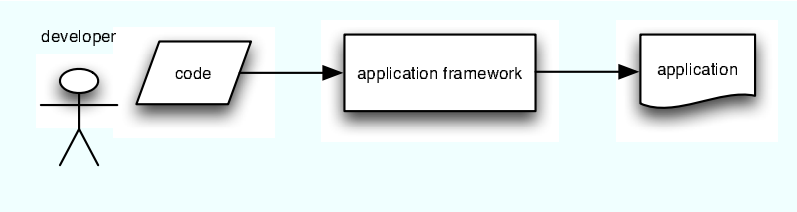

Figure 1.1 Developer inputs code into the application framework. Application framework outputs applications.

From a runtime aspect, application frameworks are application servers, that is, a program that hosts and serves applications. Developers input code into the framework, and the framework hosts the code as an application.

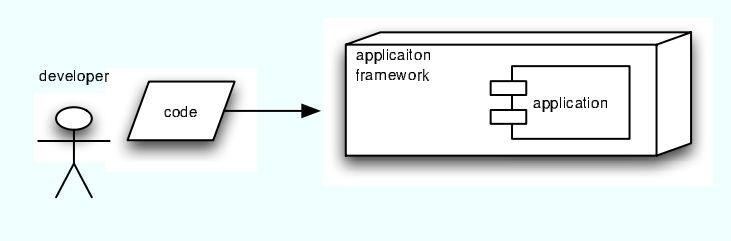

Figure 1.2 Developers input code into the application framework, and the framework hosts the code as an

application.

An industry-grade application framework product has both aspects. It has some form of a design-time tool, such as an IDE, that generates applications that are deployed into servers.

And why should you use an application framework?

Well, let's revisit our initial definition of a framework:

> "… supporting frame… of … a vehicle …"

Why is the supporting structure of a vehicle something important? It clearly sounds as if it is important, but let's see if we can articulate why that's the case. I can think of two main reasons:

1. It gives me the guarantee that I am sitting on top of something that is solid, something that has been designed properly, implemented suitably, and tested thoroughly. A framework helps to decrease defects. This is a runtime characteristic.
2. It gives the manufacture an opportunity to re-use the frame for different vehicles. A framework helps to improve productivity through re-use. This is a design-time characteristic.

It is no different for application frameworks. Application frameworks allow the creation of new applications in a form that is both efficient and with a high-degree of quality. We will discuss the advantages of application frameworks in details in later sections.

Now that we've covered what is an application framework, our next task is to understand the OSGi technology and how it can aid us on the usage and creation of new frameworks.

1.2. The OSGi technology

The Open Service Gateway initiate (OSGi) was formed in March of 1999 by a consortium of leading technology companies with the mission to define a *universal integration platform for the interoperability of applications and services*.

When I first read their mission, it gave me the impression of being both overly complex and somewhat beat-up. Haven't people already created a universal platform for applications before? As we will learn, no one has been able to do it successfully.

1.2.1. The problem domain

But first of all, let's investigate the underlying problem that these companies were facing. The initial members of the OSGi alliance were in a large part telecommunication equipment manufactures and service providers. They were interested on deploying software applications on small-memory devices. For example, consider a mobile phone as the device, and a location tracking application and an advertisement application as the software applications being deployed to the mobile. The location tracking application uses the mobile to verify the current location of the subscriber and informs the location to the advertisement application,

which retrieves selected advertisements that are suitable to the current location of the subscriber, such as promotions from near by restaurants.

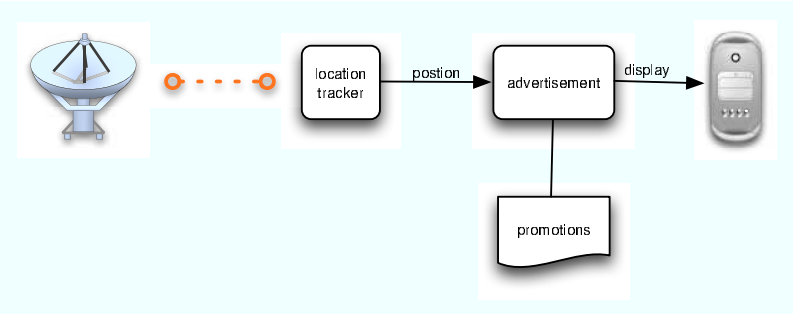

Figure 1.3 A location tracker application sends current position to advertisement application, which retrieves location-based promotions and displays it on the mobile.

This seemly simple interaction bestowed on the equipment and service providers several interesting problems. Firstly, the devices tend to have very different hardware and thus different programming APIs. Hence, each vendor had to program their applications to a specific device and then port to other devices. Secondly, not only the hardware provided different programming APIs, but also there is a large variation of their functions and capabilities. Some had more memory than others, some had disk, while others were completely disk-less, some have GPS, and some do not. Thirdly, the life-time of these devices are generally between one to two years (at the time), which means that new applications are likely to be created during this period, partly because of changing market demands. These new applications need to be dynamically deployed into the devices and join the existing eco-system of collaborating applications that are currently running in the device. And finally, due to the scarcity of the resources, these applications needed to closely cooperate with each other in a concise and, more importantly, lightweight manner.

No simple matter after all. Is there an existing universal platform that could help us, or is one in need to be created?

Again, let's go through the problems.

1.2.1.1. PROBLEM: COPING WITH DIVERSE PROGRAMMING APIS

The first problem is a problem of portability. We need a single programming platform that abstracts the application from the OS and the hardware. In other words, we need a virtual machine. Anything comes to your mind? Yes, of course. Let's use Java to solve our first problem.

1.2.1.2. PROBLEM: VARYING DEVICE CAPABILITY

The second problem is subtler. Let's consider a specific case. The advertisement application can retrieve the available promotions from different sources. If the device has ample

bandwidth, the data source could be remote, if the device does not have enough bandwidth, but if it has a disk, then the promotions could be retrieved in the background and cached in the local disk as the subscriber first enters a location.

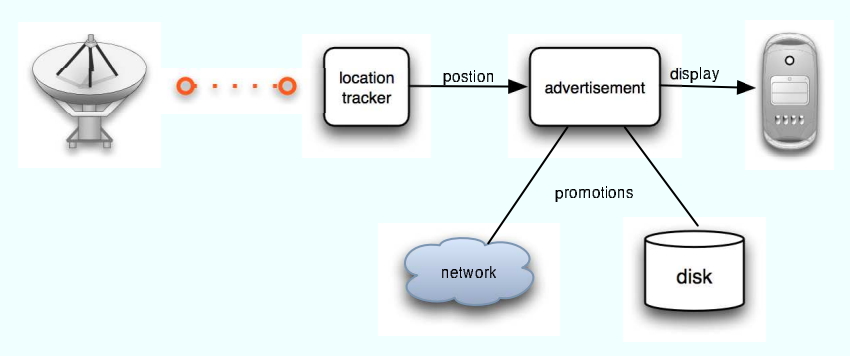

Figure 1.4. Advertisement application may retrieve promotions either from the network or from a local disk depending on the available bandwidth.

There is a clear service contract between the advertisement application and the data source; however, the implementation of this contract will vary depending on the device. In other words, this universal platform must make it easy for applications to decouple service contracts from the service implementation. The standard edition (SE) of Java does not have a service registry or a service management framework that could help us with this. Hence, this facility either needs to be implemented from scratch, or we could try to borrow something from the enterprise edition (EE) of Java. Let's hold on to this thought, and tackle it after going through the other problems.

1.2.1.3. PROBLEM: SUPPORTING DYNAMIC CHANGES

The third problem can be summarized by the following requirement: the platform must allow the dynamic deployment and un-deployment of applications in a secure form. Does (SE) Java have support for this? Not really. One could try to solve this with the Java class-loaders, but it would not be trivial and there is no simple way of un-loading classes after they have been loaded, aside from the fact that there is no concept of an application deployment unit. The closest concept to an application deployment unit is the idea of JARs (Java Archive Resource), however JARs by themselves do not provide all the metadata that is needed, such as a unique naming schema for the applications.

As the previous problem, we can implement our own solution for dynamic deployment of applications, or try to leverage something from Java EE. For example, web servers do have the concepts of web applications, which are defined as part of a WAR deployment unit file.

1.2.1.4. PROBLEM: PROVIDING A LIGHTWEIGHT SYSTEM

This brings us to the last problem. Whatever solution we pick it must be a lightweight solution. Yes, we could try to leverage a directory service, such as JNDI, from Java EE, or leverage the architecture from web servers, but these solutions would fail to consider the size and memory constraints enforced by the devices into the platform, making it less suitable for embedded solutions, and hence not a viable option.

1.2.2. The solution: a dynamic module system for Java

It is clear that Java must be used, but as we have seen, it needs to be expanded to support service decoupling and the dynamic management of Java modules.

Enter the OSGi Service Platform. In its most succinct definition, the OSGi Service Platform is a dynamic module system for Java. In the OSGi terminology, a Java module is called a bundle.

The OSGi Service Platform is composed of two main components, the OSGi framework and the OSGi services.

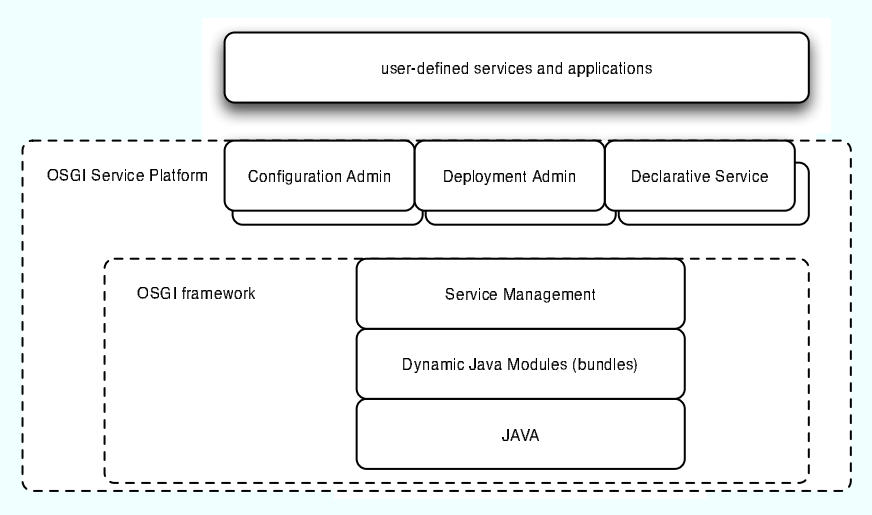

Figure 1.5 The OSGi Service platform is composed of the OSGi framework and of OSGI services.

1.2.2.1 THE OSGI FRAMEWORK

The OSGi framework provides its users with all the pieces that we discussed in the previous section:

1. A portable and secure execution environment based upon Java.
2. A service management system, which can be used to register and share services across bundles, and de-couple service providers from service consumers.

3. A dynamic module system, which can be used to dynamically install and un-install Java modules, which OSGi calls bundles.
4. A lightweight and scalable solution.

The OSGi framework is the core structure of the OSGi Service Platform. It can be seen as a backplane that hosts bundles, possibly containing services. If you consider that a bundle can be an application, then the definition of the OSGi framework is in accordance to our definition of application frameworks. That is to say that the OSGi framework is an example of an application framework.

Right now, we will leave the definition of the OSGi framework somewhat loose. Let's not worry what exactly are bundles, and services. We will discuss the framework, its concepts, and its APIs in details in the next chapters.

1.2.2.2 THE OSGI SERVICES

Along side with the OSGI framework, the OSGI Service Platform includes several general-purpose services. You can think of these services as native applications of the OSGi Service Platform.

Some of these services are horizontal functions that are mostly always needed, such as a logging service, and a configuration service.

Some are protocol-related, such as a HTTP service, which could be used by a web-based application.

And finally there are some services that are intrinsically tied to the framework, which will not work without them. Examples of these are the package admin service, which manages the dynamic module system itself, and the start-level service, which manages the bootstrap process of the framework.

In this book, we will study several OSGi services, including:

- Declarative services specification and RFC 124 (A Component Model for OSGi):

This specification, together with RFC 124, defines a non-Java XML-based language for manipulating OSGi. They are very useful for assembling components together and greatly increase the developer's productivity.

- Configuration Admin service specification:

The Configuration Admin allows the configuration of an OSGi Java module in a consistent manner. Without this specification, we could end up having to configure each OSGi bundle in a different way.

- Deployment Admin service specification:

The Deployment Admin provides a standard way of installing and uninstalling OSGi bundles into the OSGi framework. In the initial releases of the OSGi Service Platform, as odd as it may seem, there was no standard way of installing an OSGi bundle into the framework. Each vendor provided its own mechanisms, which ranged from command-line interfaces to web applications. The OSGi alliance later remedied this obvious shortcoming with the

Deployment Admin specification. To give the OSGi alliance credit, their initial rationale was that an OSGi Service Platform had to be managed by a vendor-specific management agent, which would be responsible for, amongst other tasks, the starting and shutting-down of the actual OS process and the installing and uninstalling of bundles. In practice, it was realized that many of the tasks of the management agent could still be standardized.

- Event Admin service specification:

The OSGi framework is event-full, that is, it generates and consumes several types of events, ranging from events that describe the lifecycle of the OSGi bundles, such as installed, running, etc; to service-related events, such as configuration events.

The Event Admin provides a simple publish-subscribe API to interact with the OSGi framework events as well as user-defined events. Although not commonly used yet, it is a great tool for creating high-scalable solutions.

- Permission Admin service specification:

The Permission Admin manages the security of the OSGi Service Platform. For example, it can be used to check if an OSGi bundle has permission to access a file.

The focus of the initial releases of the OSGi Service Platform had been on the OSGi framework, however gradually we see the OSGi services playing a more prominent role. This trend towards other components, such as services, built on top of the core OSGi framework is a reflection of the increasing popularity of the technology.

1.3. Benefits of building your own application framework

We have seen that if you use an application framework to construct a new application, you will gain the following benefits:

- Increased productivity on the development of new applications
- Increased quality in terms of less defects in these new applications

It should be clear why one decides to use an application framework. We do it everyday. When you create a web application and deploy it in a web server, you are actually developing a new application for your end-user leveraging features provided by the web server, such as JSP (Java Server Page) and HTTP. Likewise, when you create a JEE enterprise application and deploy it in an application server, such as Oracle's WebLogic Server, again you are creating a new application leveraging the features provided by the application server, such as message driven beans (MDB), and enterprise Java beans (EJBs). And, you are also using an application framework when you create an Eclipse IDE plug-in.

What may not be clear is why you, a developer, may also want to build your own application framework, rather than using one of the several application frameworks that exist today in the market. One of the main objectives of this book is to teach you how to create your own application framework, so we better make it clear why you would want to create one to begin with.

We will demonstrate the benefits of building your own application framework in the following sections.

1.3.1. Frameworks raise the abstraction layer – even further

Application frameworks are created to fulfill a specific application domain. In other words, the framework allows you to create new applications of a certain type, specific to a particular problem domain.

For example, web servers allow you to create web applications, that is, client applications that run on a web browser. JEE application servers allow you to create enterprise applications. Enterprise applications are applications that deal with business transactions. And finally the Eclipse IDE allows you to create rich client applications, that is, applications heavily focused around a good GUI experience.

You will notice that the three types of applications we've discussed so far, web, enterprise, and rich client, are horizontal in nature, and stirred towards a technology rather than a business.

Conversely, you, the developer, tend to work on applications that are focused on some vertical domain. One might be working on telecom applications, such as SNMP monitoring, fault detection, and mobile subscriber accounting. Or on financial applications, such as foreign exchange brokers, algorithmic trading, and trend analysis. Or manufacturing…

Wouldn't it have been better if the application framework that I am using had actually being created with my vertical domain in mind?

For example, let's consider the case of an algorithmic trading application. Algorithmic traders analyze stock quotes coming from the exchange market and stock orders coming from stockbrokers and depending on some specific algorithm determine if a stock should be sold or bought.

If we were to implement this scenario using a JEE application server, one way to model it would be to use a enterprise message-driven bean (MDB) to receive the events coming from the exchange market, and then have a enterprise session bean process the events, correlating some historical data being kept on an entity bean.

Consider the paragraph previous to the last one as our problem description, and consider the last paragraph as our implementation description for the problem. Do you see any clear linkages between the nouns used between the problem description and the implementation description? For example, is there a logical linkage from the noun *MDB* back to *exchange market*? Or can we link the noun *stock*, probably the most important entity of the problem domain to anything remotely similar in the implementation domain?

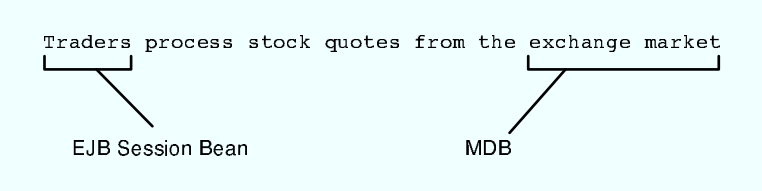

Figure 1.6 There is a lack of logical linkage between problem domain and implementation domain

Instead now let's say we are using a financial-based application framework. This framework does not have enterprise Java beans, in contrast it has the following concepts: exchanges, brokers, stock events, processors, and buy-order and sell-order events. The developer creates an application in this framework by assembling these concepts together and then binding them to specific configurations. For example, you could connect the exchange and broker to a processor, which receives stock quotes and orders, processes them, and then emits buy and sell orders. After assembling the application, you can bind the exchange to the New York Exchange market, and the broker to e*Trade.

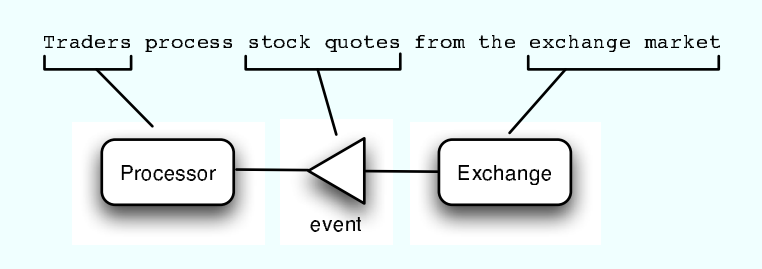

Figure 1.7 Mapping of problem domain to a domain-specific implementation

There is no question that the financial-based application framework is easier to use for this case. You could argue that my example is somewhat artificial. My response to this dispute is that the reason it feels artificial is because there are very few domain-specific application frameworks out there in the market, and hence it feels somewhat magical. But it should not feel magical. The creation of domain-specific frameworks is very doable when one uses the OSGi technology. Raising the abstraction layer to facilitate the creation of domain-specific applications is one of the main reasons why you should create your own application framework.

As Arthur C. Clarke formulated:

> "Any sufficiently advanced technology is indistinguishable from magic."

1.3.2. Frameworks focus on customer needs

In the previous section, I described a possible implementation of an algorithmic trading application using JEE technology. Remember how I used terms such as MDB, and EJB? For some of you that have used application servers before, these are well-known terms. For others, these terms shed no light to solving the problem.

Why do we need to learn these implementation artifacts? I remember when Java EE first came out there was a huge blueprint document to explain how to implement a reasonably simple pet store application.

One of the advantages of creating your own application framework is that you can keep it simple for your end-users. This is possible because you are likely to have a better understanding of your end-user than anyone else, and hence you should be able to implement your application framework considering your customer's 80% most important scenarios.

For example, using again our financial-based application framework, if by large most applications will always retrieve events from the New Your Exchange market, have it be the default exchange market, which gets used if no additional configuration is provided.

Keeping it simple for your end-users is not necessarily tied to raising the abstraction layer. You could potentially create an application framework that acts just like a JEE application, except that your application framework is configured for your end-users' default cases out-of-the-box. This application framework is easier to use for your end-user even though there are no new abstractions.

1.3.3. Frameworks are the right tools for the development of product families

One other reason for creating application frameworks is if you are in the business of developing product families.

Product families are individual products that are grouped together into a single bundle. Generally these products complement each other to fulfill a common business goal, and share a basic set of tools or features. A good example of a product family is Microsoft's Office.

If you are creating product families, then having each product being created from the same underlying application framework, will not only make sure that common code is re-used across the products, but also help guarantee a common look-and-feel to the products. It will also make sure that all products use a single configuration mechanism.

Note that differently than the previous two cases, where you had the option of either using some existing application framework or creating your own, in this case the option is to either create your own application server or not use one at all, as it is unlikely that an existing application framework already exists suitable for your product family needs.

I hope by now it is clear how useful it can be to create your own application server. In the next section we will discuss why OSGi is the right tool to use to do it.

1.4. Benefits of using OSGi

First we considered what are application frameworks and the benefits of using them. Then we looked at why you should create your own application framework. In this section we tie these issues together by explaining why the OSGi technology is the proper tool for the task of application framework building.

The question that we want to answer is the following: what is that the OSGi technology has that makes it the ideal tool for building application frameworks?

To answer this question, we need to consider what problems we face when building our own application framework.

In my experience, the problems we face are the following:

- Managing the complexity of large software systems
- Providing extensibility without eroding the system
- Lack of standardization

If the OSGi technology is able to help us address these problems, then it is the right tool for us. Let's look into each one of these problems in turn.

1.4.1. OSGi manages the complexity of large systems

We, as developers, have one time or another all faced the problem of complexity. Things are all good when we are working by ourselves, on a separate isolated piece of code. But as the team grows, from 1 to 10, and the code grows from a few thousand lines of code to several hundred thousand, so does the complexity of working with the code and the team, and the bad news is that the increase is not linear.

I am positive all of the following will sound familiar:

1. A simple change to the implementation of one component causes breakages through out the application, at apparently dissociated locations.
2. No one at the development team knows with certainty if an interface can be changed or not without breaking existing clients.
3. There are several closely related "versions" of the same utility functions through the application's source code.

How is the problem of managing large systems related to application frameworks?

Application frameworks are by their very nature complex large systems. They need to be in order to be able to support any non-trivial application.

Then, how can OSGi help us on managing the complexity of large systems?

OSGi can decrease the problem of complexity by allowing us to efficiently modularize our code.

Remember that the OSGi framework allows us to define Java modules, or bundles. Well, these bundles have formal versioned interfaces, which must be explicitly referenced by any consuming client. In fact, by defining formal contracts between producer and consumer of

code we are able to decrease the likely hood of all three problems raised in the preceding paragraphs.

For example, let's look at the first problem again in details. Consider bundle B, which contains three packages `p`, `q`, and `r`. Package `p` and `q` only contain implementation classes and do not need to be public, whereas `r` contains public user interfaces. There is a bug in class `p.C` that needs to be fixed. If we were using the OSGi framework, we could have specified that the packages `p` and `q` of bundle B is not public. This means that the only code that has visibility to these packages are within the bundle itself, which allows us to restrict testing to the bundle itself and to any consumers of the public package `r` when `p.C` is changed. By modularizing our code, we have better control over it, and know what the impact will be when something changes.

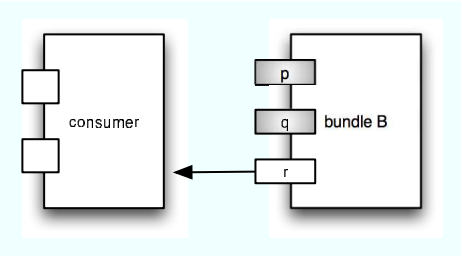

Figure 1.8 Bundle B with private packages p and q and public package r, which is being used by a consumer

In this particular case, could you have achieved the same results by meticulously coding and making sure that all Java classes are final, use the least open accessibility modifier (e.g., private members), etc? Perhaps, but would it be efficient or even possible to do these tasks in large scale, involving several people and thousands of line of code? Most certainly it would not.

Furthermore, keep in mind that in OSGi the contract between producers and consumers is specified declaratively, that is, not in Java. This gives us enormous potential for tooling. For example, we could find out the transitive closure of all classes that should be tested when a class is changed.

Nevertheless, as brilliantly stated by Fred Brooks in 1986, there are no silver bullets. It is still upon the developer to design adequately. For example, there would be no point on using the OSGi framework to achieve modularization and then make everything public.

We will address modularity in following chapters; so don't worry if the details are not clear yet. The main point of to understand is that the OSGi framework improves modularity, which in turn decreases the complexity of managing large projects and increases re-use.

1.4.2. OSGi provides extensibility without eroding the system

As you have probably guessed, a framework needs to be extensible. Actually, that is the whole point of a framework, a framework fulfills its value by being extended or customized by the user.

The problem with extensions is that it opens up your system. It is like a public API, however more problematic because people have greater flexibility with extensions than with public APIs. You will find that people do sometimes the unexpected with extensions.

Extending a system slowly helps erode it.

This is similar to software maintenance. As a software ages, fixing bugs becomes harder and harder. Every code change takes longer to be made and has a greater potential of causing other problems in the software.

The reason for this erosion is that both in the case of adding extensions and fixing bugs you are incorporating new code that was not made by the original authors of the software.

How can you restrain the erosion? You have to bound and control the new code.

How can OSGi help you with binding and controlling extensions? As we have seen, OSGi has this concept of services, service consumers and service providers.

A service is consists of an interface and an implementation. The service consumer only sees and uses the service interfaces, whereas the service provider supplies the service implementations and does not interact directly with the consumers.

Generally, extensions of an application framework will play the role of service providers, and the actual framework plays the role of the service consumer. The service interface defines the contract of the extension, in other words, it is the extension point.

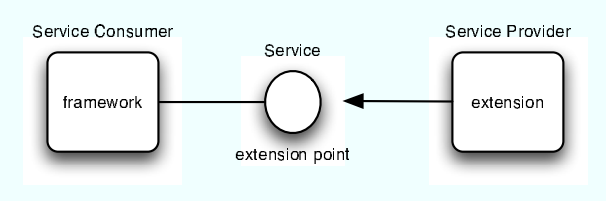

Figure 1.9 Extensions provide services through extension points. The framework consumes the services.

In this case the extension hooks itself to the lifecycle of the framework, the framework calls back into the extension when it is appropriate. One example is an extension that wants to be notified when events of a certain type are received by the framework.

Sometimes, extensions also act as the service consumers. The framework still defines the service interface, but it also provides the service implementation, which is then used by extensions. An example is a convert utility library that is provided by the framework to extensions.

Regardless of the approach, by keeping the extensions de-coupled from the framework as a separate service provider or consumer and by having a formal extension contract, we are able to isolate the extension code, and thus decrease the overall erosion of the system.

Furthermore, OSGi allows us to dynamically manage the service providers. For example, OSGi supports the shutdown of a service provider that might be misbehaving without impacting the rest of the system.

1.4.3. OSGi is standardized

One of the biggest problems that application frameworks face is that generally they are not part of any standard.

The first consequence of this is that there is less opportunity for re-use. For example, let's revisit our financial-based application framework. FIX is a protocol sued by financial institutions. If some vendor develops a FIX stack, we would like to integrate it into our framework. This would not be easy if the FIX stack and the financial-based application framework do not share any common grounds, which may result into the FIX stack being implemented in C and the application framework in Java.

The second consequence of application frameworks that lack standardization is that there is greater likelihood that a standard-compliant competing technology will eventually emerge and replace the non-standard application framework.

OSGi provides a clear answer to this problem. The OSGi alliance is composed of a worldwide consortium of technology companies. Members of this consortium are Ericsson, IBM, Motorola, NEC, Nokia, Oracle, SAP, and Siemens, amongst others.

Considering our example again, had both the FIX stack and the financial-based application framework used the OSGi framework, both would have been programmed in Java and their integration much simpler.

By building the application framework upon the OSGi framework, we can inherit some of the standardization qualities of OSGi. For example, we have seen that OSGi defines this concept of a Java module. If the financial-based application framework used the OSGi framework, then any OSGi Java module provided by any vendor would be able to be installed into the financial-based application framework. In other words, the financial-based application framework would have a standard-based deployment.

You may want to point out that there are application frameworks, like JEE application servers, that do implement standards, namely in this case Sun's JEE specification. And you would be right; in fact, the existence of the JEE specification is one of the reasons why JEE application servers enjoy very reasonable success today, and a good example of the importance of standards.

However, OSGi has one major advantage over other standards such as the JEE specification: simplicity. As we will see, you will be able to create an OSGi compliant Java module by adding a few lines to a Java's MANIFEST.MF file. Conversely, a JEE compliant module needs several JEE configuration files, and Java classes that implement technology specific Java interfaces.

Simplicity plays well with standardization. Vendors will be willing to invest some effort into standardization, but they are only willing to invest a large effort into standardization if it is mandatory for their business.

Need prove of this? Take a look at the Apache Software Foundation projects at http://www.apache.org/. You will notice that several projects, such as Derby, have already been made into OSGi bundles. However, how many will you find that support JEE in some fashion?

1.5. Categorizing application frameworks

There are three aspects that define an application framework.

First, what value or services does the framework provide to the applications being created or executed by it?

For example, the application framework may provide its applications with support for security and transactions, as it is the case of JEE application servers. Or the framework may provide a drag-and-drop editor that generates code in some programming language.

The next important aspect to understand is how does the developer make use of the application framework, in other words, what's the programming model that the framework provides to the developer so that the developer can make use of the framework's services.

Some examples of programming models are:

- Java API
- Request-response protocols
- Declarative languages
- XML

The final aspect is how to configure and manage the application framework.

For example, how would you configure the framework to enable user authentication when it is started? Or how do dynamically change the number of threads associated to the framework while it is running? Most importantly, how do you debug the framework in case of failure?

The first two aspects are important to the developer. The third aspect is important to the administrator.

Asking these three questions, we are able to get a good understanding of application frameworks. There are, obviously, other issues to understand, such as the deployment model, and security, which we will discuss in later chapters.

1.6. Industry examples of application frameworks

One of the best ways of learning about application frameworks is to study examples of existing ones. We will do this systematically through out this book. For the time being, let's take a high-level look at two very prominent examples in the industry: JEE application servers and the Eclipse IDE.

We will study these two examples by categorizing them using the concepts we learned in the previous section.

1.6.1. JEE Application Servers

A JEE application server is a program that hosts and serves applications that are critical to the business or enterprise, hence called enterprise applications.

The first question to ask is what services does the server provide to the hosted applications? The main JEE services are:

- Transaction management
- Database connectivity
- Java Object/Relation persistence
- Naming and directory of objects
- Security (authentication and authorization)
- XML processing, and XML to Java binding
- Java messaging
- Remote method invocation (RMI)
- Web Services
- HTTP management through Servlets
- HTML dynamic generation through JavaServer Pages (JSP)

There are no surprises here. Enterprise applications implement business logic that need to interact with databases in a transactional form. They also need to communicate with external clients via RMI, messaging and web-services. And finally they need to provide a web-façade to their clients. All of these are directly supported by the JEE services.

The second question to ask is how does the application developers program to this services, that is, what's the JEE programming model.

The JEE programming model is organized around special-purpose objects, generally referenced as enterprise Java beans (EJB).

The flavors of beans are:

- Session beans: these objects contain the business logic, and interact with the database. Their configuration defines their transactional behavior, and their security.
- Message-driven beans: these objects represent a messaging end-point.
- Entity beans: these objects are a result of the object/relational mapping to a database. Since JEE 5, entity beans have been deprecated and replaced by a persistence service.
- Servlets: these objects interact with HTTP requests and responses.
- JSP: these are dynamic web pages, eventually generated into Servlets.

As the server is seen as containing these beans, the server is said to be a container. Or rather, to be precise, the server defines different containers for the different types of beans. Session beans, message-driven beans, and entity beans are deployed into the EJB container. Servlets and JSP are deployed into the web container.

The JEE programming model is container-oriented, which is something similar to object-oriented. The developer organizes his application around objects that have special powers supported by the underlying container and its configuration. It is not in its essence service-oriented, although recently, as we have seen with the replacement of entity beans by the persistence service, it is moving towards a more service-oriented model.

The third and final question to ask is how does the administrator manage and configure the server and its hosted applications?

The applications are configured through the specification of XML files called deployment descriptors that are placed within the application package. These XML files contain information such as the transactional mode of the session beans, the concurrency mode of the message-driven beans, the database connection URL, and user credentials used for security. In JEE 5, Java annotations can be used as an alternative to deployment descriptors. Regardless, configuration in JEE can be quite overwhelming.

Likewise, server configuration tends to live in XML files, but obviously in this case outside of any application package. Server configuration includes thread pool settings, port number, and other server-wide concerns.

Both applications and servers are managed through the Java Management Extensions (JMX) technology, which consist of a protocol, based upon RMI, and an information model, somewhat similar to SNMP Management Information Base (MIB), named management beans or MBeans. The MBeans represents the managed object and map to running resources, such as the enterprise beans.

JEE application servers are quite powerful, and have been improved quite extensively since their inception. Their main criticism is around their complexity and heavyweight-ness.

1.6.2. Eclipse IDE

Eclipse is an integrated development environment, which is to say it is a highly-integrated collection of development tools, ranging from programming language editors with syntactic and type checking, debuggers, source outliners, template wizards, etc. Its main supported language is Java, but it also supports C, and several other dynamic programming languages, such as php, and ruby.

What you may not know is that Eclipse is build upon the OSGi framework. Eclipse has its own implementation of the OSGi framework, called Equinox, which is at the core of the Eclipse IDE. All other Eclipse features are OSGi bundles running on Equinox.

Let's go through the usual three questions as we did in the previous section.

What services does Eclipse provide? There are many, some of the main ones are:

- Resource (i.e. file) management

- Standard widget toolkit, such as buttons, text boxes, and tables
- Build and release management, integrated with Ant
- Team and versioning management, integrated to CVS
- Integrated resource search
- Help infrastructure
- Java incremental builder, Java editor, debugger, and type hierarchy user interfaces
- Plug-in management

How does the application developers program to these services?

Eclipse's programming model is centered on the idea of extension points. The developer creates plug-ins, in the form of OSGi bundles, which contain a XML file configuring the extension points that are being extended by the plug-in.

Listing 1.1 Extending Eclipse help context

```
<extension point="org.eclipse.help.contexts">                    #1
     <contexts
            file="contexts.xml">                                 #2
     </contexts>
</extension>
```

#1 the extension point name
#2 the file property points to the location of the help context for the plug-in

The Eclipse framework defines several extension points. An extension point consists of a name and a set of properties. As expected, the extension point generally provides a Java interface, which the plug-in must implement and configure the implementation class name as one of the properties in the extension point.

Finally, how does one configure and manage the framework? Configuration of Eclipse is done through preferences panels, whose values get persisted to directories associated to the plug-ins in question. This means that the configuration is also modular and kept separate per bundle.

Eclipse management makes use of the OSGi framework directly. You can select a plug-in and disable or un-install it, which essentially maps to stopping and un-installing the corresponding OSGi bundle from the OSGi framework.

Regardless if you like or dislike the Eclipse IDE, Eclipse's powerful, scalable, and simple extension mechanism, which happens to be the OSGi framework at its heart, is well recognized by all as being state-of-the-art.

1.6. Summary

Application frameworks are applications whose purpose is to provide a structure for the creation and execution of other applications. The main advantages of using an application

framework are that they improve the quality and decrease the cost of the applications being created and executed by them.

The OSGi Service Platform provides a dynamic Java module system for Java. It allows Java code to be modularized and to be managed as services. The OSGi Service Platform consists of the OSGi framework and of the OSGi services, for which some examples are the OSGi Configuration Admin service and the OSGi Deployment Admin service.

The reasons to build your own application framework instead of using an existing application framework in the market are:

- To provide infrastructure that is closer to the problem domain that is being solved
- To provide infrastructure that is simpler to use, because it is focused on a particular problem
- As a tool for the creation of product families

The creation of application frameworks is not a simple task. There are inherent problems that must be solved, these are:

- Application frameworks are complex and large software systems
- Application frameworks need to support their extensions in a controlled manner
- Application frameworks need to be standardized if they are to be long-lived and have greater opportunities of being extended.

OSGi helps us solve all of these problems. OSGi provides the means to achieve modularization, which helps decrease and manage the complexity of large systems. OSGi provides a native extensibility mechanism through the use of services. Finally, a consortium of several large companies is driven the OSGi specifications, hence aiding with its adoption as a standard.

When studying application frameworks, you should ask three questions: what services the framework provides? How do you program to these services? And finally, how do you manage and configure the framework?

There are several existing application frameworks in the market, two of the most popular ones being JEE application servers and the Eclipse IDE.

In the next chapter, we will give a brief pause to describing application framework building and drill down into the OSGi framework in details. It is essential to be well versed in the OSGi technology before proceeding with our studies of frameworks.