Issue 2/2009(4)

Java eXPress

NEWSLETTER FOR JAVA DEVELOPERS

Introduction to Grails

TeamCity : pre-tested commit

Layering

XML in Java - XStream Library

Graphical Modelling Framework

GroovyMag 06/2009 review

>> Express killers Part III

>> J2ME: Object Serialization - Part II

>> The problems of large J2EE applications

I18N

It has happened. It is almost 1 year when I started community driven initiative in Poland to create good newspaper about Java. First 4 issues (one every 3 months) were published only in Polish, however the readers' response was so great that I decided to give that magazine to every Java developer. That's why you can see our last issue (in Poland it was June issue) in English.

There was so many people that helped me to get to this point that I am afraid of forgetting somebody. I would like to say thank you to my wife, who gave me time to do this. In addition she gave some of her own time to review articles and help me making many other decisions.

I would like to thank Marek Podsiadly, who is guy who wrote most of the "new" webpage for javaexpress.pl and dworld.pl. Also big thanks to Jakub Sosinski (kontakt@vriltek.com), who is responsible for all graphics. I wouldn't be able to publish English issue without translators. Starting from Pawel Cegla, who translates most of the articles to English, Magdalena Rudny, Lukasz Baran and Boguslaw Osuch who joined the team and contributed to this issue.

Last, but not least I would like to thank our partners, e-Point and ISOLUTION who support our effort. And Adobe, who gave us InDesign to make our DTP much easier.

If you would like to join our team or contribute article, or maybe your company can support us, please write to me at kontakt@dworld.pl.

See you in 3 months,
Grzegorz Duda

Schedule

Editor: Grzegorcz Duda
Proofreading: Agnieszka Wadowska-Duda
Web page: Marek Podsiadly
Graphic design: Jakub Sosinski
Translation: Pawel Cegla, Magdalena Rudny, Lukasz Baran, Boguslaw Osuch

GeeCON 2009

Jakub Dżon

Inception

The idea of realizing an international conference dedicated to Java language was born in minds of its organizers more than an year ago, when the lack of such event was bothering them the most. The idea was simple – organize large conference during which the speakers would be people well known among Java developers and the attendees would come mostly from Central Europe. Another feature of the conference was going to be its „mobility" each year (as it was supposed to be a recurring event from the beginning) it was supposed to be held in different Central European city. After many long preparations the date and place of the first edition of GeeCON conference was established to May, 7-8, Kraków, Multikino.

The invitations to give a speech were sent to many people, amongst which some agreed to come. Unofficial motto of the conference was "from community to community" so Call-for-Papers was announced, thanks to which we managed to invite another people with interesting presentations.

Day I

The first conference day begun (for attendees) at 8:30 AM with registration and breakfast. The opening lecture was Simon Ritter's "JavaFX: The Platform for Rich Internet Applications", after which the conference ran on two tracks. The other speakers that day were Alef Arendsen, Corneliu Vasile Creanga, Miško Hevery, Waldemar Kot, Luc Duponcheel, Jacek Laskowski, Václav Pech, Szczepan Faber, Piotr Walczyszyn and Hans Docter.

At lunchtime, the attendees were provided with hot lunch, ones quality was a subject of disagreement. In my opinion lunch was great;). The first day of the conference ended at 6PM.

Day II

The whole second day of the conference ran on two parallel tracks. Because of some personal issues one of invited speakers - Michael Hüttermann was unable to attend. At the last moment we managed to ask Antonio Goncalves to conduct second lecture instead of Michael; the subject was Glassfish application server. All the other speakers on that day were Arjen Poutsma, Adam Bien, Stephan Janssen, Giorgio Natili, Paweł Wrzeszcz, Tomasz Kaczanowski, Thomas Enebo, Bruno Bossola, Lubomir Petrik and Jakub Podlesak.

The last day of the conference was planned to be shorter than the first one, but more exciting – at the end, among all the attendees who filled up the satisfaction surveys, were drawn some gifts (netbook,

IntelliJ Idea licenses, books and JDD 2009 and Confidence 2009 passes).

Accompanying events

Apart from attending lectures, the conference guests could take part in workshops held at AGH University of Science and Technology the day before the conference. Additional trainings were prepared by Sun Learning Systems and split into four paths taking place between 11 AM and 6 PM.

During the conference attendees could have participated in Java mini-quiz at JavaBlackBelt booth. The prizes in this contest were books provided by Helion publisher.

Google company also provided conference participants with additional entertainment by drawing its gadgets among the people who filled up coupons given away earlier.

On the evening of 7th of May the participants were able to take part in "Beer Certification Path" , organized by Compendium Certification Center. During that event attendees had to visit four designated Cracow pubs and drink one free beer at each of them. Drinking a beer was certified with a stamp – all four stamps made JAVA caption. Collecting all of the stamps was granted with gifts from Sun Microsystems.

Participants' opinions

Below I would like to quote some opinions given by the attendees in satisfactions surveys.

"It's really good idea to have Java conference in central Europe"
"Amazing work!"
"Lots of valuable information for everybody"
"No failures. Everything was fine!"
"...the greatest Java conference in Poland, very good speakers"
"In general, you did great job!"
"You brought really great speakers"
".. in the fact whole conference is very good and I'm glad to be here, hope to see you next year!"
"I'm impressed what organizator had made. Great job!"
"Most speakers were good (fresh subjects, well-prepared) or VERY GOOD! In general: Good work!"

The guilty ones

For the idea and realization of the GeeCON 2009 are responsible the members of Polish and Czech Java User Groups and the GiK association; Adrian Nowak, Radosław Holewa, Jakub Dżon, Grzegorz Duda, Andrzej Grzesik, Marcin Gadamer, Miroslav Kopecky, Adam Dudczak and Adam Parchimowicz.

Introduction To Grails

Mateusz Mrozewski

In this article I would like to introduce Grails framework using a practical example and a way of generating the application based on a model called scaffolding. We will configure our work environment step by step and run our first application. It will serve as an entry point to further work with Grails.

What is Grails?

Grails is a framework based on Groovy language. It introduces the "convention over configuration" principle. The dynamic behaviour of Groovy together with integrated Java solutions (we use the same virtual machine) allow us to develop quickly not only a prototype but a fully functional application. Adding DSL (Domain Specific Language) to that and we will get an excellent package to work with. What is worth mentioning is the fact that Grails is not reinventing the wheel. Some similarities to other frameworks (not only Java based) can be observed, but "under the hood" we can find widely used solutions: Spring, Hibernate, Jetty, HSQLDB, SiteMesh. Learning curve is very beneficial for almost every programmer, regardless of experience and abilities. Let us check it out in practice.

First application

In order to start working with Grails we have to spend a little time preparing the environment. Nice surprise, it only takes a moment. What do we need? I suggest using NetBeans 6.5 and Grails 1.1. This set gives us an environment to work without any additional installation and configuration. We download NetBeans 6.5 (http://netbeans.org) and Grails 1.1 (http://grails.org). NetBeans has by default the ability to create Grails projects. The package with Grails contains, besides the framework and Groovy, Jetty web server, HSQLDB database and other necessary components (Hibernate, Spring, SiteMesh). The only thing we have to do after installing NetBeans and extracting Grails to any location, is point NetBeans to this location. We select Tools -> Options from the menu, Miscellaneous -> Groovy tab and fill out the Grails Home field.

After preparing the environment it is time to create our first project. We select File -> New Project... from the menu and then Groovy -> Grails Application. We type AddressBook as the project name (we will prepare a simple address book). We click Finish and we are ready. Our first Grails project can be run. We can do it by clicking the Play button or pressing the F6 key. A welcome screen will appear that assures us we did everything correctly.

> Grails is a framework based on Groovy language.

Let us start implementing our address book. We will create the model in the beginning. The classes of the model are called domain classes in Grails. A domain class is an object representation of an entity from a database. From the context menu of the project we choose New -> Grails Domain Class..., type the name Contaxt and click Finish. A new class called Contact.groovy is created in grails-app/domain directory. There we can define attributes which describe a contact from our address book. To make things simple we will define 4 attributes: name, phone, email and address. Our class should look like below:

```
class Contact {
   String name
   String phone
   String email
   String address
   static constraints = {}
}
```

To create a working application we will need a controller, because Grails uses the MVC pattern. In order to do that, select New -> Grails Controller... from the project context menu, use Contact as the name and click Finish. There will be a new class ContactController.groovy created in the grails-app/controllers directory (surely you have noticed that NetBeans shows these directories as “Controllers” and “Domain Classes”). The newly created controller has a default action – index, but we will use a Grails feature which is called scaffolding.

The scaffolding feature allows us to generate user interface and business logic to perform basic CRUD operations on our data (Create, Read, Update, Delete). In order to do that we have to modify our controller to look like this:

```
class ContactController {
   def scaffold = true
}
```

Now we are ready to run the application again (Play or F6). This time we should see a welcome page with all available controllers. Only ContactController will be available in our case. After selecting it we should see an empty list of contacts as well as an icon allowing us to add a new one. We can give it a try right now. Our application is not very complex but it is fully functional. The one thing that the application lacks is data validation. We can use the built-in validation which works well together with scaffolding. To define the constraints let us add constraints section in our domain class. It is used to define not only what values could the attributes have but also order them in the generated application. The constraints section should look like:

```
static constraints = {
   name(blank:false)
   phone(matches:"^\\+*\\d+")
   email(email:true)
}
```

The order of the attributes in this section will be used on the screen displaying the details of the record, the list view and on the add/edit forms. We define that the name attribute cannot be empty. You should remember that blank can be used only for text values and states whether the value may be an empty string. There is a nullable constraint as well, but it states if the value may be null. In case of the phone attribute we use a regular expression to validate it (an optional plus sign at the beginning and one or more digits). For the ema-

> “under the hood” we can find widely used solutions: Spring, Hibernate, Jetty, HSQLDB, SiteMesh.

il attribute we use a built-in e-mail address validation. Plain addresses are not validated, that is why we do not put this attribute in the constraints section (we should if we wanted to change its order). After running the application again we can test the validation. The validation messages are not very clear (especially for the phone number), but it works.

What next?

Grails and Groovy have been very popular recently, so everyone interested in them should not have any problems finding information about them, other programmers’ experiments, tips and add-ons. Below is a short list of links which could be used as a starting point:

Official Grails website – http://grails.org

Official Groovy website – http://groovy.codehaus.org

Jacek Laskowski website – http://jacek-laskowski.pl

Chlebik’s website – http://chlebik.wordpress.com

You can visit my blog http://tech.mrozewski.pl in order to discuss Grails, find some useful information e.g. contact.

The problems of large J2EE applications

Jarosław Błąd

The problems of large internet applications J2EE can be generally divided into two categories. The first consists of problems connected with the implementation of systems, the second – the problems that occur during use and maintenance of systems. A lot has been already said and written about the implementation of large systems, I will focus exclusively on the second group of problems, rarely mentioned in the literature. My article will be based on my experiences gained in *e-point* company, concerning solving problems of maintaining and using big internet applications.

At first let's define, **what is „large system"**.

The basis criterion is the **volume of users, transactions and data**. If one of the elements is big, then we can talk about large system. Usually, although not necessarily, there is also some level of system logic complication in large systems, so called **business logic**. Sometimes the size of the system is perceived through the functions it serves the final user. However, I don't think it is right. Google as a search engine is a good example to prove it, as its interface offers users little functionality but surely we can't say it is small system. And finally: large system can be defined as business-critical for a client, i.e. its failures cause notable financial loss.

Now, let's think, **in which areas can occur problems in large systems**.

First of all, they can lie in the **code of the application.** The second type of problems concerns **software** application, which our application has to use. These are problems occurring in http servers, application servers, data base, MQ systems or outside systems, with whom or system communicates. Another areas of problems are **operational system and network protocoles**, where can also hide some surprises, and of course also **hardware**, on which everything operates, can also be a source of problems.

The number of errors is falling according to the presented hierarchy of system elements – the most of errors is in an application, the least in hardware. However, the closer to the hardware the problem is, the more serious and harder to solve it becomes.

When we know, where the problems can occur, lets see, **what can hurt us**.

The first thing is the **system stability**. Users expect that in longer period of time a system will be carrying out certain business functions, bringing potential benefits. The unstable working system automatically causes distrust of final users, and at the same time is a reason for losing credibility of system owner.

The next thing is **performance**. It is differently viewed from the point of view of final user than from the side ordering the system. The final user expects mostly the fastest response time. On the other hand, the side ordering system will expect mostly throughput, so the highest number of business transactions in time unit. Therefore, I propose to assume that the performance means some kind of throughput of the system with the accepted response time.

There is also something called **system ava-**

> The number of errors is falling according to the presented hierarchy of system elements – the most of errors is in an application, the least in hardware.

ilability, describing how long the system is available for users. It is connected mainly with the stability and performance.

And two additional aspects: **security**, whose importance is obvious, and **system administration**. There are systems built is such a way that unables administration. With some system scale and load, or large number of servers, clusters etc., the administration of such system can be difficult, and costs of its maintenance extremely high.

Comparing these two aspects, i.e. the areas of problems and its features, we get quite a big map of problems (Illustration 1).

	Application	Application software	Operation system/ system protocols	Hardware
Stability	X		X	
Efficiency		X		
Availability			X	
Security	X			
Administartion				X

Illustration 1. The area of problems

It is impossible in such a short article to discuss the whole aboved mentioned map of problems. Therefore I will present three problems I came across in *e-point.*

I would like to begin with something gentle, that is **problems concerning concurrency of applications.**

Assume that we have the system, which works stable for a few months after implementation. Constantly there is more and more users, and the load of system elements is bigger as well. However, it works without bigger problems. Till one day. The growing number of users causes the appearance of some problems – the system freezes for couple of seconds, sometimes for couple of minutes. Some of these cases end with the total „death" of the system and we must to restart the computer. However, what is important, during such „death" we can observe quite a big drop of load on the processors, and we don't see a lot of operations I/O, neither in the data base nor on the application server.

We begin to observe in the application server – specifically in logs of the application – exceptions with deadlocks, so the information which is indirectly returned by JDBC driver, that the data base detected a deadlock of transaction which had been done, and that the transaction had been rolled back. There are also the exceptions illustrating that JTA transactions on the application server are timing out. The analysis of the situation was not simple, but finally we managed to discover what were the reasons for such condition.

First of all, it turned out that in two particular business transactions we had unfavorable intertwinement of readings and writings. It concerned the operations on

> There are systems built is such a way that unables administration.

the data base. Already this fact could lead to deadlocks, but in this particular case it had to be something more. It was the full scan one of the table in the data base. The table was occurring in both transactions. When we were requested the reading of one record, the optimizer considered it is necessary to scan the whole table. Logically the operation of the optimizer was correct. However, from the point of view of performance it caused drastic drop of concurrency, which caused blocking of the key transactions..

When we discovered the cause of the problem, it was not difficult to solve it. Together with changing the application and uploading on the product system it took us several hours. However, the analysis itself and looking for the cause of the problem took us several days.

The solution was simple. We moved the modifying operations at the end of these transactions, which was possible from the point of view of business requirements. Then we prepared a special hint for data base optimizer which caused that the preferable way of access to the table was the use of index, not the full scan of its content.

Let's sum up the problems of concurrency.

During studies we were said that when we have problems with deadlocks, we should just block resources in the established sequence, and everything will get back to normal, particularly there will be no deadlocks. Unfortunately, this academic solution works only theoretically and it is difficult to put in into practice, even in small systems. Why? First of all, in a real business system we are dealing with thousands of transactions intertwining each other, and each of them keep some resource – usually till several dozens. And, what is worse, practically we don't have any control over blocking these resources, because usually we operate on the data base. If we want to select/take a line from the table, usually we expect that only this one will be locked. But it does not have to be necessarily true. It can be a page of table as well (lines in the data base take are grouped usually in bigger units of data) or even the whole table. If we have no control over it, it is absolutely not possible to establish any sequence, because that's no use anyway. Sometimes it results also from the business requirements that it is not possible to turn some transactional operations, which actually pins down the academic approach towards the problem.

If it is so, what can we do in practice? We can, above all, test the system with teh help of real business scenarios. The point is to observe the users' activity after implementation of the system, and with the achieved data build business scenarios. In such a way, after introducing changes in the application or after extension of the system, we can carry out certain scenarios in a lab, overloading the system and observing all the parameters of concurrency, i.e. what is happening in the data base and on the application server. Extremely important is also monitoring concurrency parameters systematically during operation of the production system.

Another topic is **contact with outside systems**. Let me remind that we have the stable, working for 2 years system. There

> The solution was simple. We moved the modifying operations at the end of these transactions

is no performance problems and the users are satisfied. At the client's request we implement a function confirming a transaction via SMS. After the implementation, the system works stable for couple of weeks and suddenly is unavailable for users. On our side, on every element of the system a complete drop of load is visible. In the system nothing is happening, if not to mention front-end http server accepting numerous failed requests. What we can observe is the saturation of threats in the application servers. All threats on the server which could process requests are in use Nothing is happening in logs. What are we trying do do?

Because of the fact that on the first site there is nothing we can draw conclusions from, we decide to restart the system. The system wakes up, but after 2 minutes is dead again. We restart it once again and the same thing happens. What helps is removing SMS module. The system wakes up and works correctly. We know already where is the problem. It turned out that the new functionality breaks something dramatically. After analyzing the situation we found out that the direct cause of the problem was the unavailability of the SMS gate. However the gate did not refused the connection but simply everything wasn't working according to the requests sent by us, which were simply frozen. TCP timeouts on the level of the operational system are very long, so during that time nothing was able to break, so we had no exceptions. The real source of the problem lied somewhere else. It resulted from insufficient separation of our system and the SMS gate. We expected certain separation but it was not sufficient. In our solution, when a user was making transaction, his/her data were to be found in the proper structure in the data base, and additionally the SMS was in a separate table, which then was read and sent by totally asynchronous process. Unfortunately this process caused locking writings to the table with messages, which in case of problems with the SMS gate resulted in freezing of the whole system.

What kind of solutions we can have in this case?

Such problems it is solved usually through the introduction of queuing. It can be for example JMS. Moreover, it is necessary to introduce timeouts with communication with such outside systems. Not only on the logical level, but various, also on the TCP level. However we could not use queuing, so we had to use a trick with the data base. We simply organized an access to the data base in such a way that the reading of messages table will not lock any writings.

Contact with outside systems we are integrating or cooperating with, is the most frequent cause of problems in large systems. Always, when in such interactions a network is considered, a lot can happen. Another examples of such contact points are:

Data base – very often we think that data base is the integral part of the system, but looking from the point of view of the application it is the same point of contact as in case of backend system or a SMS gate. Therefore we have to configure well appropriate timeouts and half connections

> Unfortunately, this academic solution works only theoretically and it is difficult to put in into practice

parameters, in order to avoid friction during between application severs and the data base.

E-mail servers – sometimes they are also in bad condition, in their own specific way

Outside systems – modern built system practically do not exist regardless other outside systems. There is always some other system – smaller or larger – taking part in operation of our system.

Logging events subsystem – an interesting fact is that it can also give us a hard time. I know an example of application server, which was logging events but when a log was 2GB the server disappeared, i.e. JVM disappeared and was not able to be switched on again. The solution was to clear the log, but it took a lot of time to get to this solution.

It seemed that **threats** does not **saturate** so quickly, especially when we have a solution of 10 virtual machines with 50 threats in each of them. It gives us 500 threats waiting to accept the request from the Internet and serve the users. On the other hand when there is simultaneously 100 requests per second served, and individual threats begin to block each other, it is a matter of several seconds, sometimes two minutes that all the threats will be blocked, which will cause a complete freeze of the system.

Summing up the discussion on the problems with outside systems, I would like to focus on the layer of network. It will not happen much here. TCP/IP heaps in operation systems, firewalls, routers, switches, VPN canals for systems, with which we integrate in a way... Everything can stop working and usually in the at least expected moment. What is worse, it can fake to work, which in case of network happens quite often. Additionally there are other damages, e.g. broken wire, which sticks only a bit.

Another topic is **memory in application servers as an example of performance restriction**. Quite a straightforward example. As usual I begin with the description of the idyll: we have a stable working system for several months, systematically growing number of users, the system is slightly more loaded, but it does not make our life a misery. On the application server there is 70% load of processor. Our client is satisfied. However in moments of increased activity we can observe alarming symptoms. At first the respond time of the system is longer and very often there is maximal load of the processor on the application servers, which sometimes cause the freezing of the system. In logs we do not observe anything unusual, so it seems that they are classic symptoms of the simple system overload. Java simply overload the processor and it should get more power.

In such case we have two methods this classic problem with efficiency of Java. We can optimize it more than already the optimized application, which needs changes in the source code and further tests. Therefore it is a risk but above all it rises further exploitation costs of the system, which is often forgotten. The second solution is adding hardware, which seems easy and unproblematic. It can be done quickly – adding a new machine, installing suitable software, adding a machine to cluster and waiting for positive effects.

We chose the second solution. After adding some new hardware, a significant improvement has been observed, but not such

“Luckily, modern applications have tendencies to allocate a large number of objects that are left for the garbage collector.”

as we expected, so further than linear. We analyzed the problem once again. If Java had been the reason for the overload, we began to observe thoroughly the work of the virtual machine. And what were the conclusions? The cause for the problem was not the low processor performance, but small amount memory for the application, and because of that too frequent activating the garbage collecting process.

Luckily, modern applications have tendencies to allocate a large number of objects that are left for the garbage collector. With every request we create a lot of objects and then we expect that they will be nicely destroyed. In case of the virtual machine, with which we were dealing with, it wasn't so and the garbage collector had more work to do. Actually, it was the processor which was operating the garbage collection process instead of executing the application code.

Which solution did we use? First of all we extended the physical memory on servers and we set up couple of virtual machines on each of the physical servers. Why so? Why didn't we, for example, increase the amount of memory on the virtual machine? The reason for that was the 32-bit machine and it was simply impossible to do. On our operation system we could extend the memory heaps only to 1600 MB, so less than 2 GB. As we couldn't do more, we had to find a different solution.

As it is quite interesting example, I will analyze it in details.

First of all: what is on the memory heap on the application server where our application is installed? Actually, there are two types of objects. Long-lived objects, connected with the application server, users' session and cache memory, which are not deleted from the memory during the garbage collection process, or it happens very rarely, e.g. for a user's session. On the other hand we have short-lived objects, so generally all that are connected with every request sent to the server. These objects, practically with every call of the garbage collector are destroyed from the system memory.

What is crucial for the performance of the whole system is the time of work dedicated for the garbage collecting process (Garbage Collector time, GC_{time} variable) and the time between each call of the garbage collector (Allocation Failure Distance, $AF_{distance}$ variable, defining how often it is called).

How to define the load of the processor for the application and for the garbage collector? In the first case we take the time of the gargabe collecting (GC_{time}) and divide it by the sum of garbage collecting time and the gaps between calls of garbage collector ($AF_{distance}$), so the total time needed for garbage collecting of memory and normal work (see: Illustration 2).

$$CPU_{GC}\,[\%] = \frac{GC_{time}}{GC_{time} + AF_{distance}}$$

Illustration 2.

In case of the application we have the opposite dependence, so we divide the time the processor spares on the operation of the application ($AF_{distance}$) by the time in which the garbage collector is not working (see: Illustration 3).

$$CPU_{APP}\,[\%] = \frac{AF_{distance}}{GC_{time} + AF_{distance}}$$

Illustration 3.

> We will see if it is possible to guarantee the client these successive nines in practice

When it comes to the garbage collecting time, with rough approximation we can say that it is directly proportional to the size of the memory used by the long-lived objects. At least such a dependency can be observed for the algorithm *mark and sweep*.

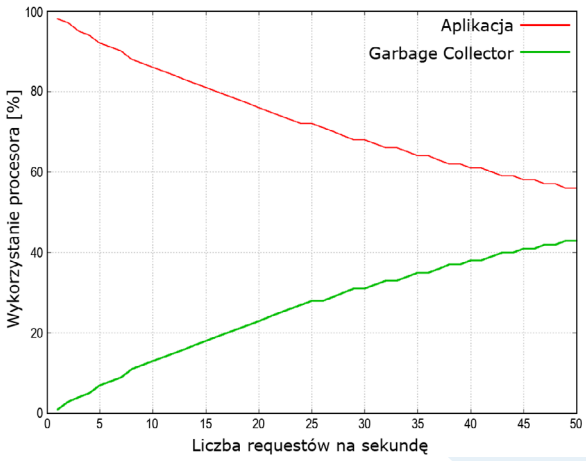


Illustration 4.

However the distance between particular calls of the garbage collecting process ($AF_{distance}$) can be defined by subtracting the size of the long-lived objects (LongLivedObjectsSize) from the size of the heap set on the server (HeapSize) and dividing everything by the product of the number of requests per seond (RPS) and the size of short-lived objects generated by every requests (ShortLivedObjectsSizePerRequest). The dependecy is illustrated on illustration 4.

Now, let's go back to the example of our application, which has the 2GB heap. We assume that the application receives requests. In our case, the memory for long-lived objects is about 700 MB, which is less than 1/3 of our heap. For every request there is a bit over 1 MB. We set experimentally that the time garbage collecting time of the 1 MB heap will take 2-3 milliseconds. On the basis of the earlier presented dependency we can set the time the processor uses for application and for garbage collecting process, which is presented on the illustration 5.

$$AF_{distance} = \frac{HeapSize - LongLivedObjectsSize}{RPS * ShortLivedObjectsSizePerRequest}$$

Illustration 5. The time the processor uses for application and for garbage collecting process of memory. Description: *wykorzystanie procesora* – the load of the processor; *liczba requestów na sekundę* – the number of requests per second

With 50 requests the garbage collecting process takes 40% of the processing time, so to put it mildly, there is not much left for work of the application. And the trick is to maximally flatten this curve. If we don't have possibilities to do anything in our virtual machine (e.g. a change in the garbage collecting algorithm or changing other parameters), we simplest way is adding some additional memory to make the space for short-lived objects much larger. And we did so.

At the end of this article I suggest to take a look on the topic of availability, so the mythical nines...

In some documentations for application servers it is described how to achieve the successive nines – the first, the second... and even the eighth one! We will see if it is possible to guarantee the client these successive nines in practice, and if so – how.

The first nine - availability of the system on the level of **90%**. It means 73 hours unavailability in month. So we have a lot of time to repair something or even to buy new hardware in the supermarket and install everything one again, or use a hosting

> **99,99%** means less than 5 minutes unavailability in month, so a bit impossible.

center. To create such a system, the team doesn't need to have advanced knowledge, and it can be serviced by any administrator. A piece of cake ;-)

Let's see what will happen if we add the second nine. Well, **99%** of availability means a bit over 7 hours in reserve. It gets complicated. Firstly, the hardware from the supermarket is not enough. The necessary minimum we have to guarantee is the standby server, prepared in such a way that it can replace the potentially broken one in the production system any time. Because in fact we don't have time for reinstalling everything. The next thing is to take care of the effective cooperation of certain system elements, e.g. take into account aspects of concurrency, interactions with other systems, so that everything will compose a whole. And finally: such a level of availability needs to be supervised 24/7. Is it then possible for one administrator to fulfill such task? Let's assume that it is – there are some that would deal with it alone, but certainly they would need automatic mechanisms for monitoring the system, necessary on this level. However still, despite some difficulties, two nines are not hard to achieve.

Let's complicate it a little, adding another nine. **99,9%** of availability gives us 43 minutes in month for breakdowns. In my opinion, this is the challenge only for professionals. First of all, a redundancy of the hardware and both system and application software is necessary. It means we ha have to guarantee high availability clusters – on http servers, application servers and the data base servers. Naturally, it concerns also the hardware – we have to have guaranteed clustered switches, firewalls, disk arrays etc. What we need is a precise project and a precise implementation of such system, because we cannot afford a bigger breakdown. Another thing is the analysis of the state before breakdown – the system need to be monitored to react quickly in case of any problem. What is important is a permanent prevention and not putting out "fire" in case of breakdown. Additionally, we have to guarantee experienced development and hosting teams, who had dealt with such tasks before. They have to work and monitor the system constantly and react quickly in case on any breakdown symptoms. Moreover we have to equip the administrators with automatized repair procedures and take into account that 43 minutes is a very short time, and when we add stress it turns out that a human cannot do much apart from pressing the stop/start button. However, the implementation of such button in large systems is not easy. Switching off and on again all elements of the system, which works on several dozens of machines and is composed of 100 or 200 software elements, takes usually couple of minutes, even if it's well built. Moreover, it is required to prepare the system for reconfiguration in the fly. It concerns the application itself as well as its components on which it operates. What's more? Double monitoring will be needed, preferably monitoring of the monitoring. Although perhaps it is a requirement adequate to the next nine. Summing up the topic of three nines – speaking from my experience, it is possible.

And so we got to the fourth nine and he-

> This means that problems (the equivalents of the explosion of supernova) can happen every day

re...a surprise – I will not describe how to achieve it, because I can't do it myself. **99,99%** means less than 5 minutes unavailability in month, so a bit impossible. Unfortunately sometimes this is what the clients expect, if not 100% availability and, of course, for free. Such approach can be understood. However it is surprising that J2EE software suppliers promise not only four but even more nines! In their documentations they present wonderful scenarios, large charts, extended diagrams of various hardware, clusters and so on. What they promise, is for me nothing but an abstract idea.

Why then do I think that in practice it is impossible? Because even the best hosting centers provide nowadays the services with 99,95% availability, so they leave 21 minutes in reserve for breakdowns on their side. And yet we have to add the time for the breakdowns on our side. Therefore it is visible that not much can be done, even if the components are working on the level of 99,99% availability, so only a bit higher.

Let's think how much would cost each of the successive nine? According to the calculations made in the USA, every next nine increases the costs of creating a system one time, and double annual costs of its maintaining. According to my personal calculations, this regularity works in case of the first three nines. Therefore the decision to make the system available is a business decision, preceded with detailed calculations. Each company should calculate individually, how much it can lose and how much it can save on the more available system.

And finally the summing up.

There is a very popular opinion that THIS cannot happen in our system, and the probability that THIS is going to happen is close to 0. There is nothing more wrong. Let's calculate this. In the medium internet system there are about 500 hits per second, each of the users downloading 10 elements – let's say there are flash, graphics etc. We have 3600 seconds in an hour, 24 hours in a day, 365 days in a year, and we maintain the system for 5 years, because it is the average lifespan of the internet system. The conclusion is that we have about 800 billions of chances that something may go wrong. In the Milky Way, according to the latest calculations, there is 400 billions of stars, so the big scale is very adequate here. This means that problems (the equivalents of the explosion of supernova) can happen every day.

The basic question is not what can happen but when it can happen. And we should be prepared for this. I hope I managed to present that in large systems we have to deal with really big problems, but there is also a great satisfaction when we are able to overcome them. I thing there is a lot of people with the experience as a developer, who will find such satisfaction solving problems in such systems. I am such a person.

Jarosław Błąd
e-point SA

Development Director
jaroslaw.blad@e-point.pl

Manages technical department, organizes environment for programmers, web developers and hosting administrators. Responsible for implementation of IT systems and their technical maintenance.

Interested in IT systems implementation processes, developers' team management and relational databases.

GMF - HOW TO CREATE A GRAPHICAL EDITOR IN A FEW MOMENTS

JAKUB JURKIEWICZ

It is widely known that a good drawing can express more than a thousand words. That is why data visualisation capability is so important. Creating tools that allow us to present data in a visual way has been expensive and work-consuming. These problems were the basis to create GMF (Graphical Modeling Framework) which is one of the projects developed under the Eclipse project. As the name suggests GMF is a technology to operate on a data model in a graphical way. Working with GMF consists mainly of creating and editing XML files (which can be done by using special wizards and editors) and the resulting graphical editor is generated automatically and ready to use without any additional work. It is worth mentioning that many options (e.g. zooming in and out, Outline view, printing capability etc.) are available “out-of-the-box” and it is not required to spend our time to implement them. Figure 1 shows an editor created using GMF. With the release of the newest Eclipse Ganymede platform a new version of GMF was released as well - 2.1 (and this one will be presented in this article).exa

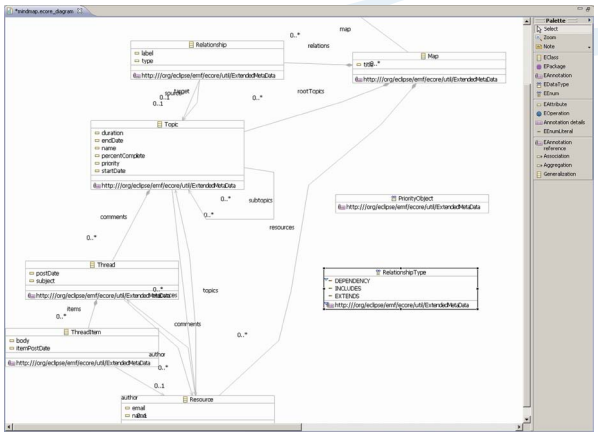


Figure 1. Screenshot showing an example of an editor using GMF.

GMF components

The name of GMF can be easily divided into two parts: the first one refers to graphical operations, the second - modeling. EMF (Eclipse Modeling Framework) is the most often used for modeling in Eclipse and that is why this technology is the base of GMF. On the other hand GEF (Graphical Editing Framework) is used to do graphical operations in Eclipse and is the base for building a visual editor in GMF. Despite the fact that GMF depends on EMF and GEF, only little knowledge of these two is required to start working with GMF. Only when creating advanced editors you have to know EMF and GEF better. In this article we will focus on GMF components (if it is possible), leaving out the details of the other technologies on which GMF depends.

How does it work - a bit of theory

Before moving to a concrete example it would be good to get to know with components of GMF - a diagram with dependencies of the components is shown on figure 2.

To start working a model is needed on which we will operate. Model has to be defined in the form of an ecore file, which is later used to create other elements. If creating this file directly is too difficult we can use an editor from GMF which allows us to create a model in a visual way (a screenshot of this editor is shown on figure 2). To use this feature right click on the select ecore file in the Package Explo-

> GMF is a technology
> to operate on a data model
> in a graphical way.

rer view and choose Initialize ecore_diagram diagram file from the menu - it will generate a file with an ecore_diagram extension, which we could edit in a visual way (and the changes will be propagated to the right ecore file). When our model is ready we need a genmodel file from which we will generate the code of our model and edit project. At this point our work with the model is done and we start working with standard GMF elements. We generate gmfgraph file from the ecore file - it will contain the definitions of graphical elements which will appear in our editor. To put the defined elements in the editor we need appropriate tools available on the editor palette - we define them in a gmftool file, which also is generated from the ecore file. After that we have to connect all these elements together - this is done in a gmfmap file (in this file we define e.g. which model element is to be cre-

Listing 1

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="company"
    nsURI="gmf.example" nsPrefix="gmf.example">
  <eClassifiers xsi:type="ecore:EClass" name="Company">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="companyName"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="employees"
upperBound="-1"
        eType="#//Employee" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ownedComputers"
upperBound="-1"
        eType="#//Computer" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Manager" eSuperTypes="#//
Employee">
    <eStructuralFeatures xsi:type="ecore:EReference"
name="managedDevelopers" upperBound="-1"
        eType="#//Developer"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Employee">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="usedComputers"
eType="#//Computer"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Computer">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Developer" eSuperTypes="#//
Employee"/>
</ecore:EPackage>
```

> The main component of GMF is the model stored in the ecore file

ated with the selected tool and what graphical element should represent it). Next a gmfgen file is generated from gmfmap in which we can set some properties of the editor. When the gmfgen file is ready we can generate editor's code as an Eclipse plugin.

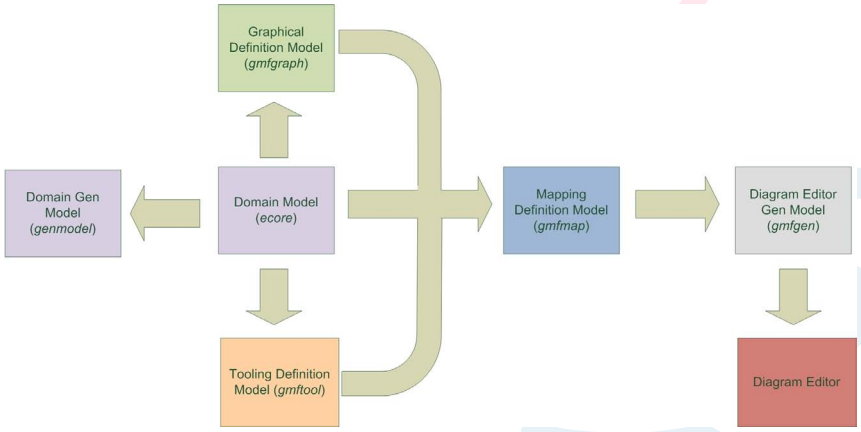


Figure 2. A diagram with dependencies of GMF components.

Creating an editor

After a theoretical introduction it is time to try creating an editor using GMF. To use GMF it has to be installed in Eclipse. As GMF is a part of Eclipse Ganymede, we can easily download it using P2. Select Help -> Software updates... from the main menu. Open the Available Software in the window tab and expand the Ganymede Update Site node, check the Models and Model Development category and select Graphical Modeling Framework SDK. Click the Install... button in the upper right window corner and after a little while we will be asked for confirmation, click Next, on the next page accept the license (I accept the terms of the license agreement) and click Finish. Eclipse will ask to restart the environment, click Yes and GMF should be a part of our IDE. We can start working after successfully installing GMF. Let us start with creating a new project by selecting File -> New -> Project from the main menu and choosing New GMF Project from the list of available projects. Give a name of the project in the wizard (gmf.example in this article) and click Finish. The main component of GMF is the model stored in the ecore file, that is why the first step to create an editor is to define a model, which we would like to edit. Our model will be quite simple for the sake of learning, not to complicate things. Our model is shown on listing 1 as an ecore file. In order to use it, create a new file with the ecore extension under the model directory (company.ecore). We should see an error message that the file is invalid, but do not worry about this fact. Open the file in text mode (in the Package Explorer view from the context menu select Open With -> Text Editor).

Type or paste the contents of the listing into the opened editor, save the file and model is ready as an ecore file (it is worth opening the ecore file in the Sample Ecore Model Editor). It would be good to know the model, because it will be needed later in the article. To continue the work we will need a model as Java code and an edit project (which allows us to easily operate the model). That is why we will generate a genmodel file from the ecore file -

> Let us start with the visual side of our editor

EMF Model wizard available under File -> New -> Other -> Eclipse Modeling Framework -> EMF Model. Give a name of the file (company.genmodel) and select that it should be created in the model directory of our project. By clicking Next we move to the page on which we select the type of our model (in our case Ecore model). On the next page give the path to the model file - it is best to click Browse Workspace and select the company.ecore file. Click Next and Finish - company.genmodel file should appear in the model directory. Open the file created by the wizard and right clicking the top element select the options: Generate Model Code and Generate Edit Code. Now it is time to leave out EMF and use GMF exclusively.

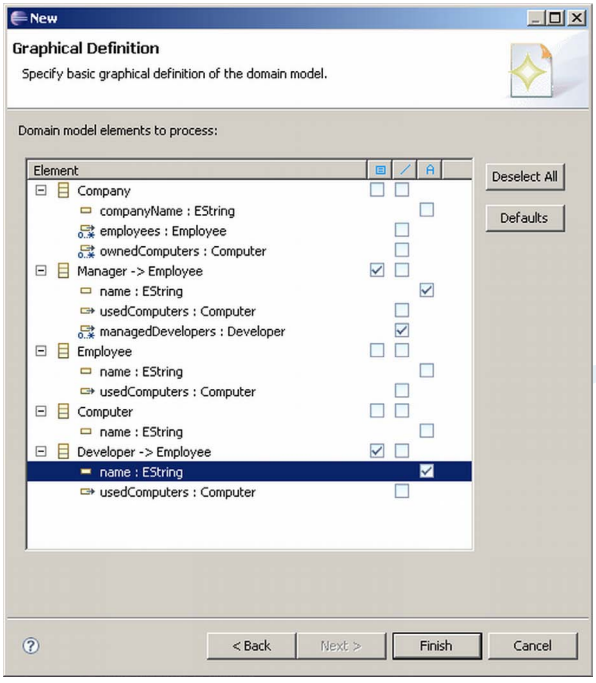


Figure 3. A page of the graphical definition wizard showing which model elements will be shown.

Let us start with the visual side of our editor. Right click on the ecore file and select New -> Other -> Simple Graphical Definition Model from the context menu (Graphical Modeling Framework category). Give the name - e.g. company.gmfgraph - in the wizard (it is best to put the file in the model directory so all elements of our editor would be in one place) and click Next. On the next page select the element which will be a container for all the elements - in our case Company and click Next. After that select the elements from the model which are to be shown. At this time we want to create Manager and Employee objects as well as links between them and their names (the result of this is shown on figure 3). After selecting the needed options we close the wizard by clicking Finish.

It is easy to notice a new file in the model directory, with a gmfgraph extension. It would be good to take a look at its contents, because all changes regarding the graphical elements of our editor will be done there.

The next step is defining the tools which will be available on the editor's palette. In order to do this, right click on the ecore file and select New -> Other... -> Simple Tooling Definition Model (Graphical Modeling Framework category). Give the name of the file - e.g. company.gmftool (it is best to put the file in the model directory as before) and click Next. Again choose what is the container for our editor (Company) and move to the next page. The last step of the wizard allows us to define which model elements should have tools created on the palette. We want to add

> The next step is defining the tools

Manager, Employee elements and links between them, so the result should look like figure 4. Click Finish to close the wizard.

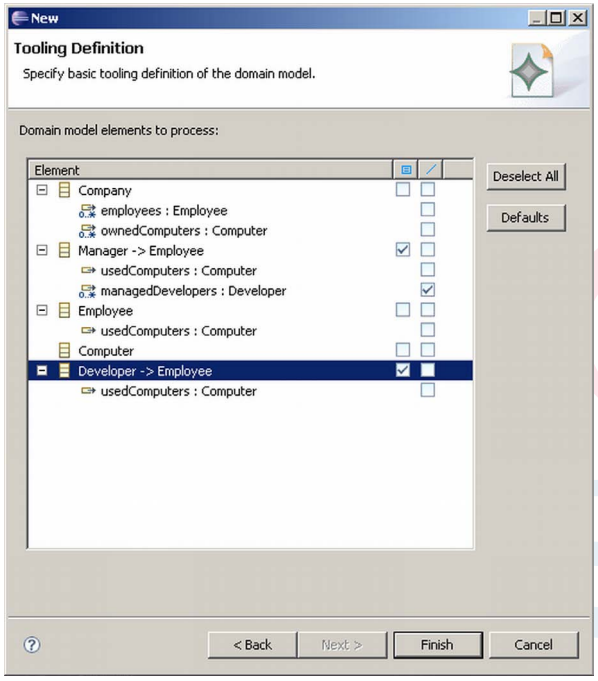


Figure 4. A page of the tool definition wizard showing for which model elements tools will be created.

Having the model, graphical elements definitions and tools, we have to connect them all together. Right click on the ecore file and select New -> Other -> Guide Mapping Model Creation (Graphical Modeling Framework category). As with the other wizards, give the name of the file storing the mapping definitions - e.g. company.gmfmap (again it is best to save it in the model folder). On the second page select the container (Company) and click Next. After that select the tools definition; click the Browse Workspace... button, choose the gmftool file and click Next. The last page is the most important one, because it allows us to define which elements will be the nodes in our editor and which will be links. Remove Computer (Manager; ownedComputers) from the Nodes section and usedComputers : Computer (ManagerManagedDevelopers; <unspecified> from the Links section; the page should look like on figure 5. Click Finish to close the wizard.

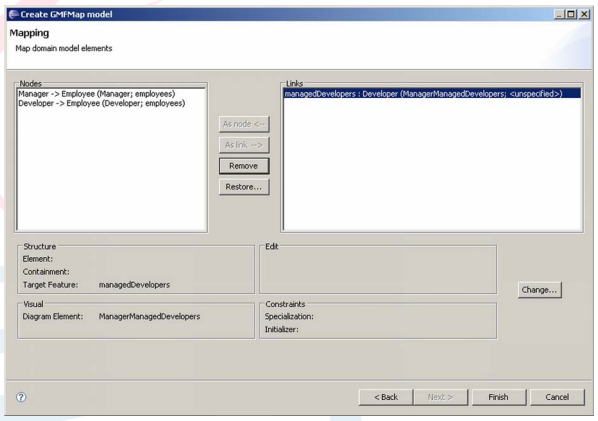


Figure 5. A page of the wizard showing nodes, links and attributes.

We have to look into the generated gmfmap file and check (in the Properties view) whether Diagram Label attribute is set for the Feature Label Mapping elements. If they are not set (the wizard does not always guess are intentions) we will have to enter the following values for them:

- Diagram Label DeveloperName for the element of that type under the Node Mapping <Developer/Developer> node
- Diagram Label ManagerName for the element under the Node Mapping <Manager/Manager>

We can check if the right tools match the right elements:

> The mapping is ready

- Node Mapping <Developer/Developer> node should have the attribute Tool set to Tool Developer
- Node Mapping <Manager/Manager> node should have the attribute Tool set to Tool Manager
- Link Mapping node should have the attribute Tool set to Tool ManagerManagedDevelopers

The mapping is ready, we only have to generate the generator model. Right click on the gmfmap file and select Create generator model... Give the name of the file - e.g. company.gmfgen (again select model as the destination folder), click Next until the last page and close the wizard clicking Finish.

Now we can generate the code of our editor. Right click the gmfgen file and select Generate Diagram Code. A new project should appear in our workspace. It contains the code of the editor as a plugin. In order to test our editor we have to run Eclipse with that plugin (as well as with the plugin containing our model and edit plugin). Click Run -> Run Configurations from the main menu, select Eclipse Application from the list on the left side and click New launch configuration. A new configuration is created which allows us to run a new instance of Eclipse with selected plugins (we will not get detailed about it here). Type in the name of our configuration and check if our plugins are selected on the Plug-ins tab. Apply the changes and run our new configuration. Create a new project in the new Eclipse instance, right click on the project and select New -> Other -> Company Diagram (we can change that name). Give the name of the file and click Finish. Our editor should open in which we could add new elements from the palette on the right side (figure 6).

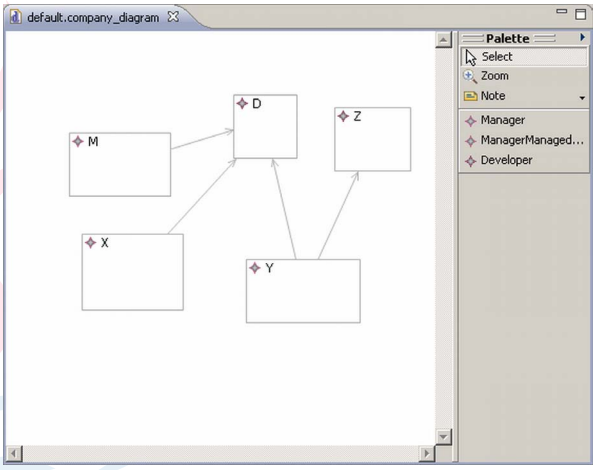


Figure 6. A screenshot of the editor right after generating it.

Adding new elements

We managed to create a fully functional editor in a short time without writing a single line of code. Instead we created and edited a few XML files (using wizards and editors simplifying this task). Now it is time to do something without wizards - let us add the capability to create new Computer elements (again we will not write any code).

Let us think for a moment - where will we make changes? First, we have to define new graphical elements (gmfgraph file modification). Second, we have to add new tools to the palette (gmftool file modification). Third, we have to map model elements to the graphical elements and show which tool will be used to create those elements (gmfmap file modifica-

Let us think for a moment

tion). Knowing that we can start working. Let us start new graphical elements. After opening the gmfgraph file we will see that the main element is Canvas. Its contents are:

- shapes available in the application (Figure Gallery Default node)
- nodes - graphical representation of model elements
- diagram labels - nodes labels
- connections - lines showing relations between model elements.

Let us assume that we want Computer elements be displayed as ellipses. Let us define their appearance first: begin with creating a figure descriptor which will contain all information about our new figure. In order to do this right click on the Figure Gallery Default and select New Child -> Figure Descriptor. A new node will be created which we will name in the Properties view (e.g. ComputerFigureDescriptor). Now we have to define our figure appearance (even many base figures). Right click on the newly created descriptor and select New Child -> Ellipse. Give the new node a name e.g. ComputerEllipse and change other properties if you like. We would like that the figure had a label, so right click on the ComputerEllipse and select New Child -> Label. Let us name it ComputerNameLabel. We have to create an access rule to the label in order to refer the label later (we will see the usage for that) - right click on the ComputerFigureDescriptor and select New Child -> Child Access. Select the element which we would like to have the access to in the properties - in our case the Figure property set to ComputerNameLabel. The appearance of the figure is defined, now we have to create a node definition for it. Right click on the Canvas and select New Child -> Nodes Node. Give a name for the new node (e.g. ComputerFigure) and connect it with our figure by giving the name of the figure descriptor (ComputerFigureDescriptor) as the Figure property. Create a diagram label node for the created label, so the editor would be «aware» that we want to display it. Right click Canvas and select New Child -> Labels Diagram Label. Give it a name (e.g. ComputerName) and connect it with a figure descriptor like previously. The access rule is useful now - set the Accessor property to Child Access getFigureComputerNameLabel.

The graphical definition is ready, we can define the tools which we would like to add (in our case we would like to add Computer elements, so we need one nie tool on the palette). Open the gmftool file, right click on the Tool Group node and select New Child -> Creation Tool from the menu. Give a name for the new node (Title attribute), e.g. ComputerTool and a description. We have to define icons for our element which will appear on the palette and in the editor - default (in our case) or selected by us. In order to add default icons right click the newly created node and select New Child -> Large Icon Default Image as well as New Child -> Small Icon Default Image. That is the end of gmftool file modifications.

The last piece of work to do is to connect the model, the graphical definition and to-

> The last piece of work to do is to connect the model, the graphical definition and tools

ols which means modifying the gmfmap file. After opening it we will see that the main node is called Mapping and stores all defined mappings. We will add a Top Node Reference mapping because it concerns the top node of the editor. Right click Mapping and select New Child -> Top Node Reference. In this new node we have to set where to store the element represented by the node (Containment Feature attribute). In our case it will be ownedComputer. After that right click the new node and select New Child -> Node Mapping and there we set the mapping details: Element attribute - element from the model (Computer), Diagram Node attribute - node name from the graphical definition (ComputerFigure), Tool attribute - tool name defined in the gmftool file (ComputerTool). We would also like to display Computer name as a label, so we right click on the node and select New Child -> Feature Label Mapping. We have to set the Features attribute to name for the new node (because this Computer attirbute we would like to display) and Diagram Label property should point to the diagram label defined in the gmfgraph file (ComputerName).

That is the end of our work. Now it is time to regenerate gmfgen file, editor code based on that and check whether our editor works as we expected. The result of our work is shown on figure 7.

Summary

Basic capabilities of the GMF project were presented in this article as well as how easily and quick one can create a fully functional graphical data model editor. GMF features are of course broader, that is why I encourage to get familiar with samples and tutorial available on the internet and experiment with the technology.

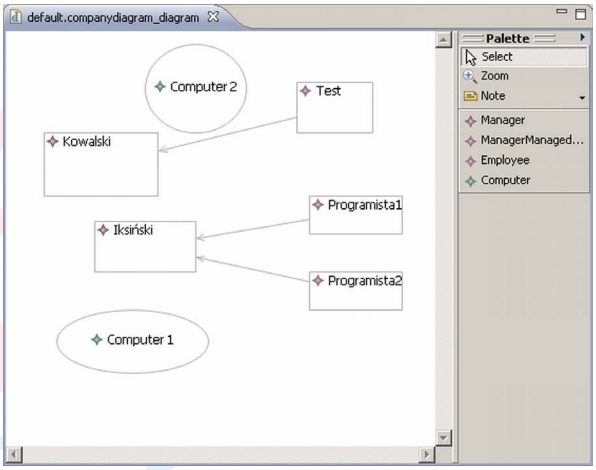


Figure 7. A screenshot showing the editor after changes.

Internet

- GMF home page: http://www.eclipse.org/gmf/
- GMF tutorial: http://wiki.eclipse.org/index.php/GMF_Tutorial
- GMF FAQ: http://wiki.eclipse.org/Graphical_Modeling_Framework_FAQ
- GMF in 15 minutes: http://www-128.ibm.com/developerworks/opensource/library/os-ecl-gmf/
- Article about an additional GMF view: http://eclipser-blog.blogspot.com/2007/06/gmf-project-in-5-minutes-with-gmf.html
- A collection of posts about EMF: http://eclipse-po-polsku.blogspot.com/search/label/EMF

J2ME: Objects Serialization

Adam Dec

Environment setup and project creation

The utility used to build our projects will be, of course, Maven 2.x [1]. To make it a little more sophisticated, we will use `maven-antrun-plugin` in the source generation phase (`generate-sources`). Ant's responsibility will be to generate code using `org.castor.anttask.CastorCodeGenTask` class.

Before going further, I would like to recommend reading an excellent article about Maven that was published in the second issue of Java Express, Dec 2008 (*Maven 2 – how to make work easier, pt I*. Rafał Kotusiewicz).

Let us create two projects:

1. CastorSourceGenerator

2. CastorMsgFactoryME

They may be created either from the command line or by using an Eclipse plugin (M2 [3]). However, I strongly recommend using the following commands:

```
mvn archetype:create -DgroupId=com.sonic.gen -DartifactId=CastorSourceGenerator
```

```
mvn archetype:create -DgroupId=com.sonic.factory -DartifactId=CastorMsgFactoryME
```

Maven creates two directories named exactly as the values given in artifactId parameter. Both directories contain default project structure that includes some HelloWorld classes:

```
src/main/java/com/sonic/gen/App.java
```

```
src/test/java/com/sonic/gen/TestApp.java
```

and POM (Project Object Model) file whose sample content is shown in the listing 1.

If you want to import templates as a pro-

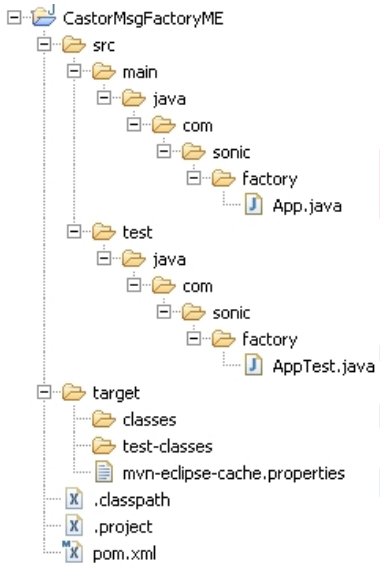
Listing 1. Sample pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.sonic.factory</groupId>
  <artifactId>CastorMsgFactoryME</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>CastorMsgFactoryME</name>
  <url>http://maven.apache.org</url>
  <dependencies>
  <dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>3.8.1</version>
  <scope>test</scope>
  </dependency>
  </dependencies>
</project>
```

> The utility used to build our projects will be, of course, Maven 2.x

ject in Eclipse, then in each of the directories (CastorSourceGenerator, CastorMsgFactoryME, CastorTester), you should invoke the command:

`mvn eclipse:eclipse` or `mvn idea:idea` if you are using IntelliJ environment.



Picture 1. Projects structure in Eclipse 3.5M3

In the first case two files will be created: `.project` and `.classpath`. Also, a new `target` directory will be created and `mvn-eclipse-cache.properties` file in it.

Having our templates imported as Java projects in Eclipse (*File -> New -> Java Project... -> Create project from existing source*), we should get project structure shown in the picture 1.

First of all, take a look at the `CastorSourceGenerator` project. Let us create `MySourceGenerator` class, that will control the default way in which the output code is generated. The fully qualified name of this class should be given as the value of `org.exolab.castor.builder.jclassPrinterTypes` attribute in the `castorbuilder.properties` file. The class should implement `org.exolab.castor.builder.printing.JClassPrinter` interface; you can find an example how

Listing 2. JClassPrinter implementation

```
package org.exolab.castor.builder.printing;

import org.exolab.javasource.JClass;
import org.exolab.javasource.JComment;

public class WriterJClassPrinter implements JClassPrinter {
    public void printClass(final JClass jClass, final String outputDir,
            final String lineSeparator, final String header) {

        // hack for the moment
        // to avoid the compiler complaining with java.util.Date
        jClass.removeImport(„org.exolab.castor.types.Date”);

        // add header
        JComment comment = new JComment(JComment.HEADER_STYLE);
        comment.appendComment(header);
        jClass.setHeader(comment);

        // print
        jClass.print(outputDir, lineSeparator);
    }
}
```

> it would be a good idea to make a copy of this directory

to do it in `org.exolab.castor.builder.printing.WriterJClassPrinter` (see the listing 2). We will use `WriterJClassPrinter` class as a template for building our own implementation.

Rename `App.java` file to `MySourceGenerator.java`, delete the unimportant stuff and add `JclassPrinter` implementation to this class. Most of the classes that we are going to use are located in `org.exolab.javasource` package and their names reflect very well their purposes. So, in order to create a method, create `JMethod` object. If you want to add parameter to an object, create `JParameter` object.

1. At this stage do not worry if the project is not compiling yet. It is time to download Castor sources (note that it must be version 1.2!) [4]. Now, go to the `codegen` directory (before that it would be a good idea to make a copy of this directory, e.g. `codegen-me`). We will change three files: `Jtype.java`, `CollectionMemberAndAccessorFactory.java` and `SourceFactory.java`.

Now, keeping in mind how signatures of serialization/deserialization methods look like:

```
public void write(final java.io.DataOutputStream dos) throws java.io.IOException

public void read(final java.io.DataInputStream dis) throws java.io.IOException
```

we should add the following declarations in `org.exolab.javasource.JType` class:

```
/** JType instance for a void (void). */
public static final JPrimitiveType VOID = new JPrimitiveType("void", "void");

 /** JType for a DataInputStream. */

public static final JPrimitiveType DATA_INPUT_STREAM = new JPrimitiveType("java.io.DataInputStream", "java.io.DataInputStream");

/** JType instance for a DataOutputStream. */

public static final JPrimitiveType DATA_OUTPUT_STREAM = new JPrimitiveType("java.io.DataOutputStream", "java.io.DataOutputStream");
```

2.Find `org.exolab.castor.builder.factory.CollectionMemberAndAccessorFactory` class. What is particularly important is to remember the code that was generated for collections such as `java.util.Vector` and `java.util.Hashtable` should call only methods that in J2ME API [5]. Except for the `public synchronized Object clone();` method, almost all methods from Java 1.1 [6] are included in CLDC 1.1 (JSR 139).

Also, you should make sure that the resulting output code contains calls only to methods from the following table:

J2ME
void addElement(Object obj)
Object elementAt(int index)
void insertElementAt(Object obj, int index)
Enumeration elements()
void removeAllElements()
boolean removeElement(Object obj)
void removeElementAt(int index)
void setElementAt(Object obj, int index)

3.1 First change will be in the method:

```
private void createGetAsArrayMethod(final CollectionInfo fieldInfo,
```

> At this stage do not worry if the project is not compiling yet

```
    final JClass jClass, final boolean
useJava50,

    AnnotationBuilder[] annotationBu-
ilders) { ... }
```

According to **JSR 139** in the class `java.util.Vector` there is no such method as `Object[] toArray()` and `Object[] toArray(Object[] a)`.

It means that we should either get rid of `createGetAsArrayMethod(...)` method body or look for:

```
private void createGetAndSetMethods
(final CollectionInfo fieldInfo,
final JClass jClass,
final boolean useJava50,
final AnnotationBuilder[] annotation-
Builders) { ... }
```

and remove the line:

```
this.createGetAsArrayMethod(
fieldInfo, jClass, useJava50, annota-
tionBuilders);
```

3.2 The same operation should be done in the following methods:

```
private void createGetAsReferenceMe-
thod(final CollectionInfo fieldInfo,

    final JClass jClass) { ... }

private void createSetAsReferenceMe-
thod(final CollectionInfo fieldInfo,

    final JClass jClass, final boolean
useJava50) { ... }

private void createSetAsCopyMethod(
final CollectionInfo fieldInfo,

    final JClass jClass) { ... }
```

In my case all the implementations of these methods were removed and `createGetAndSetMethods` method looks now as shown below:

```
private void createGetAndSetMethods(
final CollectionInfo fieldInfo,
final JClass jClass, final boolean
useJava50, final AnnotationBuilder[]
annotationBuilders) {

    this.createGetByIndexMethod(fiel-
dInfo, jClass);

    this.createSetByIndexMethod(fiel-
dInfo, jClass);

}
```

3.3 The next change is done in:

```
protected void createGetByIndexMethod(
final CollectionInfo fieldInfo, final
JClass jClass) { ... }
```

In the line:

```
String value = fieldInfo.getName() +
„.get(index)”;
```

the word `get` should be replaced with `elementAt` and this results in:

```
    String value = fieldInfo.getNa-
me() + „.elementAt(index)”;
```

So, what we did here was replacing `Object get(int index);` with `Object elementAt(int index);`

3.4 Now, look for the following method:

```
protected void createAddByIndexMetho-
d(final CollectionInfo fieldInfo,

    final JClass jClass) { ... )
```

The following piece of code:

```
sourceCode.append(fieldInfo.
getName());
sourceCode.append(„.add(index, „);
sourceCode.append(fieldInfo.
  getContentType().
  createToJavaObjectCode(parameter.
getName()));
sourceCode.append(„);”);

void add(int index, Object element)
```

should be replaced with this one:

> In my case all the implementations of these methods were removed

```
sourceCode.append(fieldInfo.
getName());
sourceCode.append(
  „.insertElementAt(„);
sourceCode.append(fieldInfo.
  getContentType().
  createToJavaObjectCode(parameter.
getName()));
sourceCode.append(„, index);”);

void insertElementAt(Object obj, int
index)
```

3.5 Remove the body of this method:

```
protected void createIteratorMethod(
final CollectionInfo fieldInfo, fi-
nal JClass jClass, final boolean
useJava50) { ... }
```

And the fragment of this method implementation:

```
private void createRemoveAllMethod(
final CollectionInfo fieldIn-
fo, final JClass jClass) { ... }
sourceCode.append(„.clear();”);
```

should be replaced wtih

```
sourceCode.append(„.removeAllEle-
ments();”);
```

This means replacing `void clear();` with `void removeAllElements();`

3.6 Replace the body of the method:

```
protected void
createRemoveByIndexMethod(final
CollectionInfo fieldInfo, final JClass
jClass) { ... }
```

Below there is the piece of code to be replaced:

```
JMethod method = new JMethod(„remove” +
fieldInfo.getMethodSuffix() + „At”,
  fieldInfo.getContentType().getJType(),
  „the element removed from the
  collection”);
method.addParameter(new JParameter(
  JType.INT, „index”));
JSourceCode sourceCode = method.
  getSourceCode();
sourceCode.add(„java.lang.Object obj =
this.”);
sourceCode.append(fieldInfo.getName());
sourceCode.append(„.remove(index);”);
if (fieldInfo.isBound())
  this.createBoundPropertyCode(fieldInfo,
sourceCode);

sourceCode.add(„return „);
if (fieldInfo.getContentType().getType()
== XSType.CLASS) {
  sourceCode.append(„(„);
  sourceCode.append(method.
    getReturnType().getName());
  sourceCode.append(„) obj;”);
} else {
  sourceCode.append(fieldInfo.
    getContentType().
    createFromJavaObjectCode(„obj”));
  sourceCode.append(„;”);
}

jClass.addMethod(method);
```

And here you can see how it should look after the operation:

```
JMethod method = new JMethod(„remove” +
fieldInfo.getMethodSuffix() + „At”, Jtype.
VOID, „the element removed from the
collection”);
method.addException(SGTypes.
INDEX_OUT_OF_BOUNDS_EXCEPTION, „if the
index given is outside the bounds of the
collection”);
method.addParameter(new JParameter(
  JType.INT, „index”));
JSourceCode sourceCode = method.
  getSourceCode();
this.addIndexCheck(fieldInfo, sourceCode,
method.getName());
sourceCode.add(„this.”);
sourceCode.append(fieldInfo.getName());
sourceCode.append(„.removeElementAt(
  index);”);
if (fieldInfo.isBound())
  this.createBoundPropertyCode(fieldInfo,
sourceCode);
jClass.addMethod(method);
```

Briefly, all the calls to: `Object remove(int index)` are changed to calling:

```
void removeElementAt(int index)
```

And `this.addIndexCheck(...)` will add a short piece of code:

```
// check bounds for index
if (index < 0 || index >= this.VECTOR
.size()) {

throw new IndexOutOfBoundsExceptio-
n(„getElement: Index value ,” + index +
```

> Huh, this was the last 'replacement operation'!:)

```
„' not in range [0.." + (this.VECTOR.size() - 1) + „]");
}
```

3.7 The next replacement concerns the method:

```
private void createRemoveObjectMethod(final CollectionInfo fieldInfo, final JClass jClass) { ... }
```

In its body we replace the line

```
sourceCode.append(„.remove(„);
```

with

```
sourceCode.append(„.removeElement(„);
```

3.8 Remove the body of the method:

```
private void createSetAsArrayMethod(final CollectionInfo fieldInfo, final JClass jClass, final boolean useJava50) { ... }
```

3.9 The last change should be done in the method:

```
protected void createSetByIndexMethod(final CollectionInfo fieldInfo, final JClass jClass) { ... }
```

In this method we change: `Object set(int index, Object element)` into: `void setElementAt(Object obj, int index)`

and the following code should be replaced:

```
JMethod method = new JMethod(„set" +
fieldInfo.getMethodSuffix());
method.addException(SGTypes.
INDEX_OUT_OF_BOUNDS_EXCEPTION, „if the
index given is outside the bounds of the
collection");
method.addParameter(new JParameter(
  JType.INT, „index"));
method.addParameter(new Jparameter(
fieldInfo.getContentType().getJType(),
fieldInfo.getContentName()));
JSourceCode sourceCode = method.
  getSourceCode();
this.addIndexCheck(fieldInfo, sourceCode,
method.getName());
sourceCode.add(„this.");
sourceCode.append(fieldInfo.getName());
sourceCode.append(„.set(index, „);
sourceCode.append(fieldInfo.
  getContentType().
  createToJavaObjectCode(fieldInfo.
    getContentName()));
sourceCode.append(„);");
if (fieldInfo.isBound())
  this.createBoundPropertyCode(fieldInfo,
sourceCode);

jClass.addMethod(method);
```

with:

```
JMethod method = new JMethod(„set" +
fieldInfo.getMethodSuffix(), JType.VOID,
„the element added to the collection");
method.addException(SGTypes.
INDEX_OUT_OF_BOUNDS_EXCEPTION, „if the
index given is outside the bounds of the
collection");
method.addParameter(new JParameter(
  JType.INT, „index"));
method.addParameter(new JParameter(
  fieldInfo.getContentType().getJType(),
  fieldInfo.getContentName()));
JSourceCode sourceCode = method.
  getSourceCode();
this.addIndexCheck(fieldInfo, sourceCode,
method.getName());
sourceCode.add(„this.");
sourceCode.append(fieldInfo.getName());
sourceCode.append(„.setElementAt(„);
sourceCode.append(fieldInfo.
  getContentType().
  createToJavaObjectCode(fieldInfo.
    getContentName()));
sourceCode.append(„, index);");
if (fieldInfo.isBound())
  this.createBoundPropertyCode(fieldInfo,
    sourceCode);
jClass.addMethod(method);
```

3.10 Short explanation

The list of classes which can be used to generate source code can be found here:

http://www.castor.org/1.3/javadoc/org/exolab/javasource/package-summary.html

What we are now looking for is the `org.exolab.castor.builder.factory.SourcefactoryClass` in which there should be `private void initialize(final JClass jClass) { ... }` method. The only thing to do is to comment out the following line

> Before we start testing our library, we need some test data.

in this method:

```
jClass.addInterface(„java.io.Seria-
lizable”);
```

We are doing this since we do not want this interface to be added to each newly created class because, as I have already mentioned this, interface does not exist in Java ME. Instead of it, we will use something different:

```
de.enough.polish.io.Externalizable
```

Huh, this was the last ‘replacement operation’!:)

Now, let us edit `pom.xml` file: change its artifact name to `castor-codegen` and add version element:

```
<version>1.2.1</version>.
```

Then, we should invoke the command in the shell: `mvn clean install`.

The artifact should be installed in our local repository (in the directory `M2_REPO/repository/org/codehaus/castor/castor-codegen/1.2.1/`) and it should be added as a dependency to the `pom.xml`:

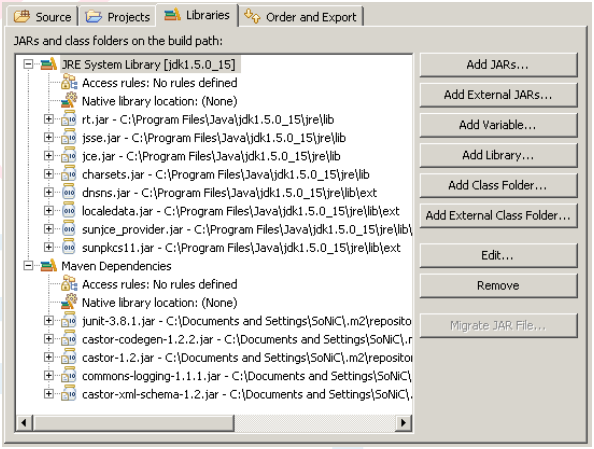
```
<dependency>
  <groupId>org.codehaus.castor
  </groupId>
  <artifactId>castor-codegen
  </artifactId>
  <version>1.2.1</version>
</dependency>
```

If you have M2 plugin installed in Eclipse, simply click the right mouse button on the project and select `Enable dependency management`. In that way all the dependencies from `pom.xml` will appear in our classpath (Picture 2).

The project should now compile successfully and it can be installed in your local repository:)

4. It is high time to implement the method:

```
public void printClass(
final JClass jClass, final String
outputDir, final String lineSepara-
tor, final String header) { ... }
```



Picture 2. Java Build Path

which will add to the already generated class (`JClass` object) the additional code (the method)

```
modifiedClass.addMethod(someMethod)
```

and some comment:

```
modifiedClass.setHeader(someComment)
```

The following code should be generated:

```
public void read(DataInputStream dis)
throws IOException {

this.name = dis.readUTF();
this.myObject2 = (MyObject2)de.enough.
polish.io.Serializer.deserialize(dis);

}
```

Step 1. Let us create a method signature:

```
JMethod readMethod = new JMethod(
  „read”);
JModifiers readMethodModifiers =
  new JModifiers();
```

> Files generation will be started by using Maven

```
readMethodModifiers.makePublic();
readMethod.setModifiers(
  readMethodModifiers);
JParameter readMethoParameter =
  new JParameter(JType.
    DATA_INPUT_STREAM, „dis”);
readMethod.addParameter(
  readMethoParameter);
readMethod.addException(new JClass(
  „java.io.IOException”), „”);
```

...I will leave it without a comment :)

Step 2. Create method body:

```
JField[] fields = modifiedClass.
  getFields();
if( fields.length > 0) {

  for(JField field : fields) {
    readSourceCode.append(
      returnProperReadMethod(field));
    readSourceCode.append(„\n”);
  }

  readMethod.setSourceCode(
    readSourceCode.toString());
} else {
  readMethod.setSourceCode(
    „super.read(dis);”);
}
```

...so we are iterating through all the fields in the class and creating the code :)

Sample implementation of the `returnProperReadMethod(...)` method may look like the one below:

```
private String returnProperReadMethod(JField
field) {
  final String name = field.getName();
  final String type = field.getType().getName();
  if(type.compareTo(„java.lang.String”) == 0)
  {
    return „this.” + name +
      „ = dis.readUTF();”;
  } else if(type.compareTo(„int”) == 0 ||
    type.compareTo(„java.lang.Integer”) == 0)
  {
    return „this.” + name +
      „ = dis.readInt();”;
  } else if(type.compareTo(„boolean”) == 0 ||
    type.compareTo(„java.lang.Boolean”) == 0)
  {
    return „this.” + name +
      „ = dis.readBoolean();”;
  } else if(type.compareTo(„java.util.Date”)
    == 0) {
    return „this.” + name +
    „ = new java.util.Date(dis.readLong());”;
  } else if(type.compareTo(„double”) == 0 ||
    type.compareTo(„java.lang.Double”) == 0) {
      return „this.” + name +
        „ = dis.readDouble();”;
  } else {
    return „this.” + name + „ = („ + type +
      „)Serializer.deserialize( dis );”;
  }
}
```

The code created in this way should be added to our class with the command:

```
modifiedClass.addMethod(readMethod);
```

Please remember to import the necessary class:

`modifiedClass.addImport(„de.enough.polish.io.Serializer”)` and about `write(...)` method implementation!!!

Time for a short quiz... Does the project compile? :) Well... take a closer look at the line:

```
for(JField field : fields)
```

Unfortunately, the compilation fails with the following error message:

```
[INFO] [compiler:compile]
[INFO] Compiling 1 source file to c:\eclipse\workspace\artykul\CastorSourceGener
ator\target\classes
[INFO] ------------------------------------------------------------------------
[ERROR] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Compilation failure
c:\eclipse\workspace\artykul\CastorSourceGenerator\src\main\java\com\sonic\gen\M
ySourceGenerator.java:[67,33] for-each loops are not supported in -source 1.3
(try -source 1.5 to enable for-each loops)
                for(JField field : fields) {


c:\eclipse\workspace\artykul\CastorSourceGenerator\src\main\java\com\sonic\gen\M
ySourceGenerator.java:[67,33] for-each loops are not supported in -source 1.3
(try -source 1.5 to enable for-each loops)
                for(JField field : fields) {


[INFO] ------------------------------------------------------------------------
[INFO] For more information, run Maven with the -e switch
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Sun Feb 15 08:53:34 CET 2009
[INFO] Final Memory: 6M/254M
[INFO] ------------------------------------------------------------------------
```

On the Maven website [7] we can read this:

„...The default source setting is 1.3 and the default target setting is 1.1, independently of the JDK you run Maven with...”

So, let us configure Maven Compile Plugin. In order to do that, we add the following section to `pom.xml`:

> Unfortunately, the compilation fails

Listing 3. Maven Compiler Plugin configuration

```
<build>
  <plugins>
  <plugin>
  <groupId>
    org.apache.maven.plugins
  </groupId>
  <artifactId>
    maven-compiler-plugin
  </artifactId>
  <configuration>
  <source>1.5</source>
  <target>1.5</target>
  </configuration>
  </plugin>
  </plugins>
</build>
```

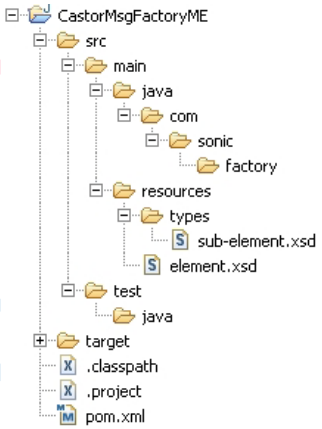
It is also essential to remember about setting the correct parameters in `castorbuilder.properties` file:

```
org.exolab.castor.builder.javaVersion=1.4

org.exolab.castor.builder.jclassPrinterTypes=\

      com.sonic.gen.MySourceGenerator,\

      org.exolab.castor.builder.printing.TemplateJClassPrinter
```



Listing 4

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema attributeFormDefault="unqualified"
    elementFormDefault="qualified"
    version="1.0"
    xmlns="http://com.sonic/element"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"

    xmlns:xmime="http://www.w3.org/2005/05/xmlmime"
    xmlns:es="http://com.sonic/types/complexTypes"
    targetNamespace="http://com.sonic/element">
  <xsd:import
    namespace="http://com.sonic/types/complexTypes"
    schemaLocation="types/sub-element.xsd"
  />
<xsd:element name="MyElement">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded"
        name="MySubElement"
        type="es:SubElementType" />
      </xsd:sequence>
      <xsd:attribute name="attr1" type="xsd:string"   />
      <xsd:attribute name="attr2" type="xsd:int"      />
      <xsd:attribute name="attr3" type="xsd:double"     />
      <xsd:attribute name="attr4" type="xsd:dateTime" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

> It is also essential to remember
> about setting the correct parameters

Picture 4. Project view in Eclipse 3.5M3

Before we start testing our library, we need some test data. We should create a sample schema file and implement an automat which will generate valid code. So, let us do it:

Go to the `CastorMsgFactoryME` and create resources folder in which the `element.xsd` file should be put. Next, create `types` folder and place `sub-element.xsd` file in it (Picture 4). It is a very good idea to use XML Schema Editor in Eclipse [8] for editing schema files. The listings 4 and 5 present sample content of these files.

Now we are going to use `org.castor.anttask.CastorCodeGenTask` class. We create build.xml file. As a matter of fact, we can delete `com/sonic/factory` directories, as we are not going to need them. Files ge-

Listing 5

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema  attributeFormDefault="unqualified"
    elementFormDefault="qualified"
                                                version="1.0"

    xmlns="http://com.sonic/types/complexTypes"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xmime="http://www.w3.org/2005/05/xmlmime"
    targetNamespace="http://com.sonic/types/complexTypes">
  <xsd:complexType name="SubElementType">
    <xsd:attribute name="attr1" type="xsd:int"    />
    <xsd:attribute name="attr2" type="xsd:double" />
    </xsd:complexType>
</xsd:schema>
```

Listing 6. Maven Antrun Plugin configuration

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-antrun-plugin</artifactId>
  <version>1.3</version>
  <executions>
    <execution>
      <id>generate-me-sources</id>
      <phase>generate-sources</phase>
      <configuration>
        <tasks>
          <property
            name="compile_classpath"
            refid="maven.compile.classpath"/>
          <ant antfile="build.xml" dir="${basedir}" />
        </tasks>
      </configuration>
      <goals>
        <goal>run</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

So, let us do it

neration will be started by using Maven. In the `generate-sources` phase, we will run Ant task using `maven-antrun-plugin` plugin **[9]**. Sample configuration is shown in the listing 6.

The `compile_classpath` property is actually the reference to Maven's classpath and it will be used in `build.xml` file. The plugin will execute the default task in `build.xml`:

```
<project    name="CastorMsgfactoryME"
default="castor:gen:src" basedir=".">
```

This behaviour can be changed by defining `<target name="nazwa"/>` in `<ant>` section.

Code generation will be done by the defined task:

```
<taskdef name="castor-srcgen"
  classname="org.castor.anttask.
    CastorCodeGenTask"
  classpathref=
    "castor.class.path" />

... and called in this way:

<castor-srcgen file="src/main/
  resources/element.xsd" todir=
  "${src.dir}"
  package="${package.name}"
  warnings="true" nodesc="true"
  nomarshal="true"/>
```

where:

```
<property name="package.name"
  value="com.sonic.dto"/>
<property name="src.dir"
value="src/main/java"/>
<path id="castor.class.path">
<path path=
  "${compile_classpath}"/>
</path>
```

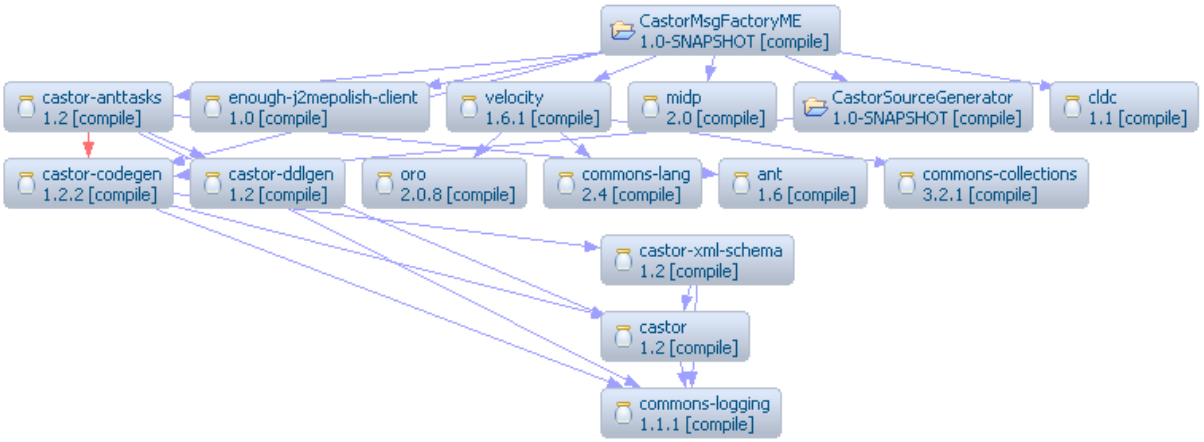
By default, CastorCodeGenTask generates collections compatible with Java 1.1. If for some reason you would like to force compatibility with Java 1.2, then you should add attribute `types` and set its value to `j2`. The `nomarshal` attribute [10] is set to `true`, to inform generator not to generate methods for marshalling/unmarshalling.

Finally, the dependency graphs for both projects are shown in the pictures 5 and 6.

The artifacts `enough-j2mepolish-client.jar`, `midp.jar` and `cldc.jar` are placed in the `/j2me-polish2.0.7/lib/` directory.

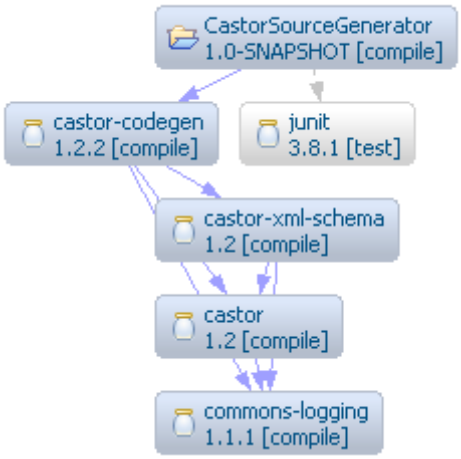
The example of how to install artifacts to the local repository:

```
mvn install:install-file -DgroupId=ja-
vax.midp   -DartifactId=midp   -Dver-
sion=2.0 -Dpackaging=jar -Dfile=/path/
```



Picture 6. From Maven POM XML Editor

Finally...




```
to/file
```

Picture 5. From Maven POM XML Editor

Links

1.http://maven.apache.org/

2.http://ant.apache.org/

3.http://m2eclipse.codehaus.org/

4.http://dist.codehaus.org/castor/1.2/castor-1.2-src.zip

5.http://java.sun.com/javame/reference/apis/jsr139/

6.http://java.sun.com/products/archive/jdk/1.1/index.html

7.http://maven.apache.org/plugins/maven-compiler-plugin/

8.http://wiki.eclipse.org/index.php/Introduction_to_the_XSD_Editor

9.http://maven.apache.org/plugins/maven-antrun-plugin/plugin-info.html

10.http://www.castor.org/srcgen-anttask.html

XML in Java – XStream library

Marek Kapowicki

XML Format

XML (Extensible Markup Language) is an universal formal language used for representing different data in a structural way. XML is platform independent which enables document exchange between various systems and made the language popular. XML became an unchallenged data exchange standard on the Web. XHTML language and documents can efficiently describe the contents of a web page; whereas XML allows us to describe any data which has a well-defined structure. In addition XML enables us to define custom dictionaries of tags for different domains, instead using a predefined set. This article's purpose is not to present a detailed description of XML. The full specification can be found on-line and in books.

Examples of use

XML documents are widely used and have a wide range of applications. I, as a programmer, often receive an XML file with data, which should be stored in a database. There could be information about people, books, institutions etc. The document has to be parsed and objects obtained from that stored in a database. Data from XML documents can be transformed into an HTML document, a PDF file or any other text format using XSLT. So it is good to know how to create XML documents to store your data in order to generate a PDF file for instance. XML is used to create configuration files – this usage is for sure known to Java programmers.

XML in Java

Popular technologies

I think that two most popular API's to access XML from Java are SAX and DOM. They are used for syntax analysis of the documents. The first one processes the document and everytime a tag, comment, a piece of text or any other XML element occurs, it calls a piece of Java code to signal an event. Our code can perform an appropriate action then. Using this API we can access only the currently processed element. SAX interface is particularly useful for processing big files. On the other hand DOM provides a full representation of the document in memory, in a tree form. Even though these API's are not subject of this article, I recommend knowing them.

Alternative approach

A different approach is data mapping (binding). Instead of working with elements and attributes, we would use Java objects, which simplifies working with XML. These approach should sound familiar, if you have used Hibernate (which maps database tables into Java objects).

To mark the difference in using data binding, a different term is used – "marshalling" instead of "parsing" or "serialization" – converting XML elements into Java objects. A reverse process is called "unmarshalling" – turning Java objects into XML. XStream is a library using this approach. It uses reflection to identify the fields that have to be stored. It is very simple (does not

> XML became an unchallenged data exchange standard on the Web

require defining an XML schema) and simplifies greatly using XML in Java. More information can be found at http://xstream.codehaus.org/.

Using XStream library

Adding the library to a project

If we want to use XStream, we have to add it to our project. In order to do this we have to download the newest jar file with the library – during the writing of this article it was 1.3.1 version. If we are not going to develop XStream, only use it, I recommend downloading the binary version. After extracting the downloaded file, we can find the jar file in the lib subdirectory (xstream-1.3.1.jar file). Next the path to the extracted jar file should be added to the Java Build Path of our project. If Maven is used, the following dependency should be added to pom.xml:

```
<dependency>
  <groupId>
    com.thoughtworks.xstream
  </groupId>
  <artifactId>
    xstream
  </artifactId>
  <version>1.3.1</version>
</dependency>
```

XStream library uses annotations therefore Java 1.5 or higher is needed.

First use

To do a simple conversion, let us create a Java Bean:

```
public class Person  {
  private String name;
  private String surname;
  private Date birthday;
  //gettery i settery
}
```

Now we will use the library to create XML:

```
public String person2Xml(
    Person person) {
  XStream mapping =
    new XStream(new DomDriver());
  String xml =
    mapping.toXML(person);
  return xml;
}
```

Let us create a Person object, populate its fields with custom data and call that method. The following result will be displayed on the screen:

```
<pl.marek.Person>
  <name>Marek</name>
  <surname>Kapowicki</surname>
  <birthday>
    1983-09-28 00:00:00.0 CET
  </birthday>
</pl.marek.Person>
```

As you can see the mapping is done according to the names. XStream maps the fully qualified class name (together with the package name), which is not always needed. The date format is a little bit “strange” as well. Fortunately we can modify the mapping e.g. by defining aliases (using annotations). To convert the resulting XML back into a Java object we use:

```
public Person xml2Person(
    String xml){
  XStream mapping =
    new XStream(new DomDriver());
  return (Person) mapping.
    fromXML(xml);
}
```

> As you can see the mapping is done according to the names.

Advanced library use

In order to show the capabilities of the library I have created a program which stores information about films. The following data is stored: title, description, genre, actors starring, date of production, director, cover and a link to a website. Application allows us to create an XML file from data provided in Java as well as to perform the reverse process – creating Java objects from a file.

Application is created using Maven. To run it, you have to type mvn install from the command line. It consists of a few packages, which will be described briefly (more information in Javadoc):

- pl.marek.beans – package containing Java beans which will be mapped to XML:
  - Film – represents a film,
  - Person – represents a person i.e. a director, an actor,
  - Films – this class contains the full information about films, owner etc. Objects of this class are converted into xml files – which are the result of running this program,
  - All the beans inherit from an abstract class BaseXML, which makes it possible to map all files the same way (if we use annotations to configure mapping, not library methods)
- pl.marek enums – enum with the movie genres
- pl.marek.xstream – the main part of the application responsible for data mapping:
  - XMLMappingInterface – interface used to map beans inheriting from BaseXML. It has four methods:
    - public String java2xml(BaseXML base) – converts an object into a string with XML content
    - public void java2xmlFile(BaseXML base, String fileName) – converts an object into an XML file

    Methods creating XML are general and can be used to convert all classes descending from BaseXML. During conversion the type of the object is detected and annotations are loaded.

    - public BaseXML xml2java(String xml, Class<? extends BaseXML> className) – converts a string (XML content) into an object
    - public BaseXML xmlFile2java(String fileName, Class<? extends BaseXML> className) – converts an XML file into an object

    Calling methods converting XML you have to pass the type of the object.

  - XMLMapping – a class implementing the above interface
  - PersonXMLMapping – used for mapping Person objects. Created to show that using XStream library methods instead of annotation we will have problems and cannot use one general mapping.
  - PersonConverter – converter used to manually map classes to XML files.

> It is best to use annotations

The code that calls these methods is in the JUnit tests directory. After running the tests we should see two new files: author.xml and films_list.xml.

Looking at the bean code we can see that we can map Java primitive types, custom class objects e.g. director field in Film class and collections – list of films in Films class.

Mapping management

Using the XStream library we can change the mapping. It is best to use annotations, adding them to fields in the beans. We have to inform the converter (XStream class object) about that fact. In order to do this we call the method: `xStream.processAnnotations(Class className)` – annotations from className class will be loaded by the converter. If we use annotations when creating an XML file, we have to use annotations when converting the file into objects. Otherwise XStream will try to map XML according to names and the conversion will fail.

Useful annotations:

- @XStreamAlias("name") can be used both with classes and fields. It describes the name to which the field should be mapped. An example for that can be found in the Film class.
- @XStreamImplicit(itemFieldName = "nazwa") – used with collections mapping. Describes the name to which collections elements should be mapped. Example in Films class.
- @XStreamOmitField – fields annotated with this will not be mapped.

If we do not want to use annotations, we can use library methods made available by the converter. While creating author.xml I have used xStream.alias("director", Person.class) in order to define an alias (replacing Person class name with string "director"). I think that this approach does not simplify using the library and makes the code unreadable.

Converters

If we do not want to use the standard conversion from the XStream library, we have to write a custom converter. In order to do so, we extend the com.thoughtworks.xstream.converters.Converter class, which implements two methods:

- public Boolean canConvert(Class clazz) – checks which classes can be converted
- public void marshall(Object value, HierarchicalStreamWriter writer, MarshallingContext context) – method which is called when an objects is converted to XML:
  - value – the object which is converted
  - writer – object used to write data
  - context – current context

This method is used in the PersonConverter – used for manuall conversion of Person objects.

```
DateFormat df = DateFormat.
  getDateInstance(DateFormat.
  MEDIUM);
```

> It is a simple tool which makes using XML files from Java quite easy.

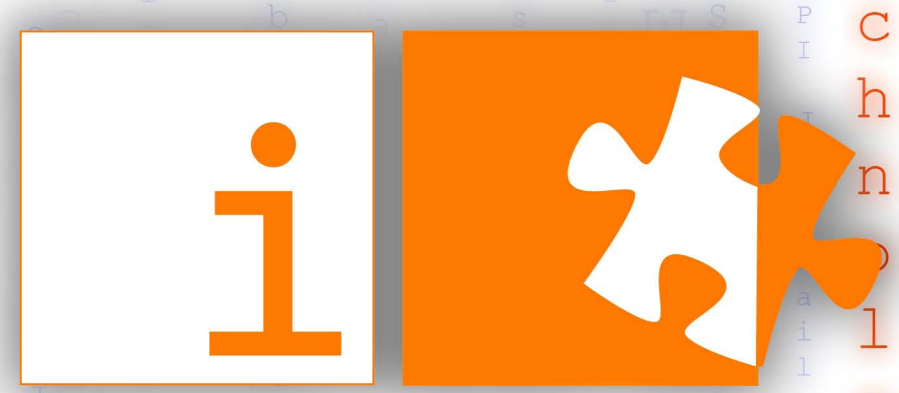
```
if(author.getBirthday()!=null)
{
   writer.startNode("birthday");
   writer.setValue(df.format(
      author.getBirthday()));
   writer.endNode();
}
```

In the method we check what field is currently processed and define the action to perform while converting that field. In the example we convert the date to the yyyy-mm-dd format.

- public Object unmarshall(HierarchicalStreamReader reader, UnmarshallingContext context) – method called when converting an xml file into Java objects.

Summary

I hope I managed to encourage anyone to familiarize oneself with XStream library. It is a simple tool which makes using XML files from Java quite easy. The library should be a problem to use for anyone. Annotations and defining custom converters allow us to manage the mapping.



Java Team

dołącz do nas www.isolution.pl

Express Killers, part III

Damian Szczepanik

First example

In this part we will consider how JVM takes care of reference printing, that are pointing to nowhere. I mean null. There is short code sample that prints class name of the object received from collection.

What will be printed?

- null

- <null>

- different value every time

- none of above

```
import java.util.ArrayList;
import java.util.List;

public class PrintNull {
import java.util.ArrayList;
import java.util.List;

public class PrintNull {

	private static PrintNull o;

	public static void main(String[] args) {
		List<PrintNull> list = new ArrayList<PrintNull>();
		list.add(o);
		for (PrintNull i : list) {
			System.out.println(i.toString());
		}
	}

	public String toString() {
		return (this == null) ? "<null>" : super.toString();
	}
}
```

Second example

Very short sample and question: what is it doing? Anything at all? Is it result of „refactoring by removing”?

```
synchronized(obj)
{}
```

TeamCity: pre-tested commit.

Paweł Zubkiewicz

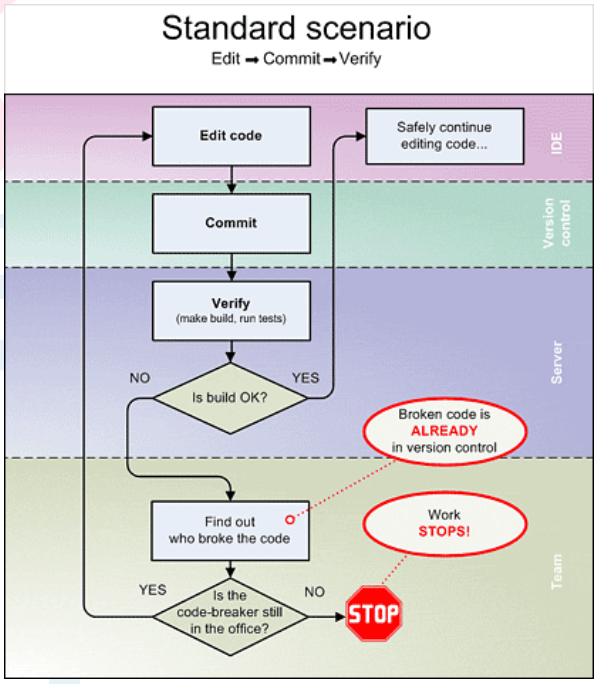
While developing enterprise solutions nowadays, It is difficult to imagine projects which do not follow a set of practices by Fowler commonly known as Continuous Integration. The most important goal of these practices is the reduction of time (and therefore cost) of introducing changes to the project by integrating these changes early and frequently. The changes are the result of many programmers' work in a team. The idea of Continuous Integration is 10 year old. During that time many tools were created which helps us – the programmers – to use this idea in everyday's work (in our own projects).

Among many applications supporting the idea of Continuous Integration a very special place is taken by integration servers, called CI servers. They enable automatic project building, test running and informing about encountered errors and failures. These are only basic capabilities of CI servers. The most advanced systems offer many more interesting features and useful functions.

One of those advanced tools is TeamCity, which gathered many users during last year. The main reason for that is a unique feature; pre-tested commit.

To fully appreciate its advantages, I remind a standard scenario of everyday programmers work:

1. Update code from the repository.
2. Changes in several classes (the actual programming).
3. [optional] Changes' verification – running the unit tests.
4. Committing changes to the repository (preceded with an update – this is taken care of by the repository itself).
5. Automatic download of the newest source code by the CI server and starting to build the project.
6. [optional] Checking whether the CI server successfully built the project with the introduced changes.



We all work that way. One could ask: what is wrong with this way of work, and furthermore, it is so commonly used? The main problem is the fact, that even when the programmer ran the unit tests before commit, there are no guarantees that the same code would compile on the CI server and pass all the tests. All of us were in such a situation some time. There are many reasons for that situation. The most popular one is the difference between the development environment and the CI server environment. These could be a hard-coded file path, a different

> The main reason for that is a unique feature; pre-tested commit.

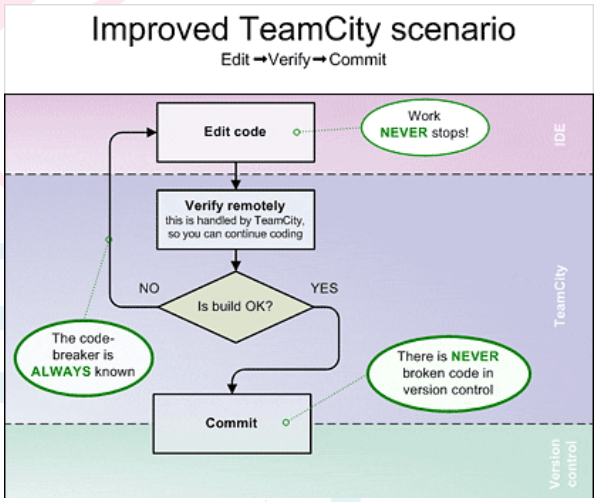
version of a software library or other unknown assumptions i.e. privileges to access remote disk. It is possible that one build could contain many changes which worked separately but do not work together. Sometimes the cause is very simple; not all modified classes were sent to the repository, which makes the code impossible to compile. This is the time when the search for the guilty one in the team begins. Finding the guilty one is not very difficult, but fixing the error might by. In the worst scenario the work of the whole team is halted and it is caused by only one bad commit.

There are many solutions to this problem; introducing a high discipline of work among programmers or highly complicated SCM solutions like stable trunk (which require high discipline as well as a higher work effort).

Engineers from JetBrains invented a unique solution of this problem. They modified the above mentioned scenario in such a way that a faulty code could not get into the repository and at the same time is not a burden for the users. This is the pre-tested commit feature which is supported by TeamCity among others.

The work scenario of a programmer with TeamCity is as follows:

1. Update the code from the repository.
2. Changes in several classes (the actual programming).
3. Sending the modified files to TeamCity (using an IDE plugin).
4. Start of the build process using the files send by the programmer and the files from the repository:
   a. Adding the modified files in the repository if the build was successful.
   b. No changes in the repository if the build failed.
5. Send information to the programmer about the build result.



As it can be seen, the repository contains only checked code in this scenario. The amount of checks depends only on our tests. We have the guarantee that an error of one programmer does not block the entire team and work is not stopped.

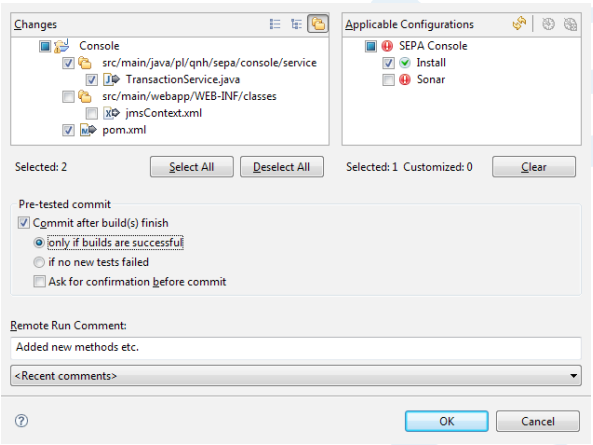
Using pre-tested commit is very simple in practice and resembles using plugins for code repositories. A working instance of TeamCity server and an IDE plugin is required. IntelliJ IDEA and Eclipse environments were supported at the time of writing this article. Plugin for Eclipse adds four new tabs and Remote Run is the most important one. It allows to use the pre-tested commit feature. It is possible to start a build with “private” changes, which is called “Personal Build” (private build).

> Using the pre-tested commit feature is easy

You can choose on the Remote Run tab:

- files with the outbound changes
- build configuration (created earlier on the TeamCity server) in which the changed files are to be used.

Checking the option Commit after build(s) finish, changed files are added to the repository only if the build completed successfully. And this is the core of pre-tested commit idea – the repository only contains the files which were tested on the CI server.



It is worth mentioning that Personal Build can be used as a tool to increase the productivity of a programmer (or even as an alternative to provide him a more powerful workstation). Using this feature a programmer can run his tests (build project) remotely and, at the same time, working on another part of his solution. His workflow is not interrupted by waiting for the local build to finish, which often is time and resource consuming.

The company behind TeamCity is JetBrains, known mainly from the IntelliJ IDEA environment. Even though TeamCity is a proprietary solution, it is available free of charge as a Professional edition. This edition has a restricted number of users, project configurations (builds – 20) and Build Agents (3). The number of Build Agents sets the maximal number of concurrently running build processes. These restrictions however could not prevent deploying this solution in small and medium projects free of charge.

To sum it up TeamCity is an innovative product in the CI servers' world. This solution can reduce the number of problems with continuous integration which occur in everyday work of programmers' teams causing loss of time and money. Using the pre-tested commit feature is easy and does not require the programmer to seriously change his work scenarios, which makes it cheap and feasible to implement (the team would not raise any objections).

Express Killers, part III - Solution

Damian Szczepanik

First example

Null was added to collection, so it will be taken in loop. Try to call method on the null object will result in throwing exception. It will be safer to use System.err.println(i) instead, what will print „null". Overwritting toString() method is useless. Also check if this is equal to null is pointless (when is it true?).

Second example:

I found that code once in repository. We were discussing together if it is done by purpose or maybe it is side effect of some refactoring and code removal. At first sight this code is useless, because there is nothing inside. However, if you look closer, we might find potential usage. There is sample usage. Check what will be printed with and without this block in printTrue() method.

Calling wait(2000) is done only for purpose of this example, and execution may be different on different hardware and VMs (however 2 secs should be enough to see difference). To sum up, code is weird, however removing it may cause with some new bugs to fix.

```
public class Sync implements Runnable {
	private static final Object obj = new Object();
	private final boolean id;
	public Sync(boolean id) {
		this.id = id;
	}
	public static void main(String[] args) {
		new Thread(new Sync(false)).start();
	}
	public void run() {
		System.err.println("start:" + id);
		if (id) {
			printTrue();
		} else {
			printFalse();
		}
		System.err.println("end:" + id);
	}
	public void printTrue() {
		// remove that lines and check results
		synchronized (obj) {
		}
	}
	public synchronized void printFalse() {
		synchronized (obj) {
			try {
				new Thread(new Sync(true)).start();
				wait(2000);
				System.err.println("print");
			} catch (InterruptedException ex) {
				ex.printStackTrace();
			}
		}
	}
}
```

BLENDED FOR THE JAVA COMMUNITY

DEVOXX™

the java™ community conference

Layering

Mariusz Sieraczkiewicz

Why layers (tiers)?

Almost every programmer has heard of multi-tier (two-tier, three-tier or n-tier) architecture. However, in many conversations concerning software development I often had a feeling that this element has little impact on everyday life of programmers' work. Although this subject is strictly related to system architecture. No matter what your role in the project is, it can always affect the things that you are working on. The purpose of this article is to explain how to use the idea of multi-tier architecture in practice and how its understanding may improve the quality of your code. It also attempts to show some pragmatic approaches in this matter.

What is it all about?

The notion of layers, similarly to many other human inventions, was introduced in order to make our life easier. In this case, the main goal was to simplify computer systems development. To organize system structure it is advisable worth to determine logical parts of a system that are related to each other and share some common responsibilities. Therefore, in many systems we can distinguish, for instance:

- user interface that is responsible for user interaction, in most cases by appropriate views or windows.
- domain which could be defined by main system data, processing, algorithms, computations and system lifecycle.
- communication with the outside world, which can be understood as ways of accessing data, writing and reading it persistently and cooperating with external systems.

That was only an example of splitting system features into layers. There might be more of them and they may be defined in various other ways.

The first determinant that arises from using layers is the division of a system into logical parts with clearly defined responsibilities. These parts are orthogonal to system functions.

The second determinant is defined by relations between layers. Similarly to onions, computer systems are built out of layers. The most outer layers are closer to the users of an application. Therefore, layers are ordered and serve functions between themselves. This dependency is shown in the picture below.

User Interface
Domain
Data Access

As you can see, the most outer layer is a user interface, which is responsible for communication – it deals with querying and showing data and with a logical views organisation. The actual processing in a system is delegated to the domain classes, since it is responsible for main application functions (separately from the UI). On the other hand, all write, read or external systems communication operations in the domain layer are delegated to the data access layer. Only the adjacent layers can communicate with each other directly; however, it is always the upper layer that calls the one below.

> Is it worth to apply layers pattern?

What are the main advantages of using layers (tiers)?

- Layers are a way of dividing a system into high-level logical components – it is easier to manage and understand them because each of them has got clearly defined responsibility.
- Each layer is uniquely constructed and provides a set of interfaces that should be implemented.
- Tiers should be treated as autonomous entities which are to a large extent independent of each other.
- Components in a layer may be reused in other applications with the same layered structure and this supports the idea of creating application frameworks.
- Independent teams may independently work on the development of each system layer.
- Components from different layers may be independently implemented, installed, maintained and upgraded.

Of course, there are also some drawbacks:

- The existence of several layers may cause some serious changes in functions of a system and they often enforce cascade modifications in numerous layers.
- Layers may affect performance of a system.

If you create your application working on your own or if you have influence on a system's architecture, then layers will certainly help you to take control of a project and make its creation simpler – as the responsibilities in a system are clearly determined. If you come across situations when, after a couple of development days, your application becomes inconsistent – there are many iterations and you do not know how to separate the code responsible for database queries from the rest of the system, then you can overcome this obstacle by dividing the system into layers.

If you are a member of a larger team, then it is very likely that the system's architecture has already been chosen and someone has decided that your system will follow the layers architecture schema. It may be defined differently in different technologies, but the basic idea always remains the same. Knowledge of layers allows you to understand easily the system you are developing and makes it significantly easier to get adjusted to its boundaries. You should then be aware what your classes are responsible for and what is beyond their scope. Layers are like continents on the map of the world – you know what can be found and where it is located.

Simple example with no layers

We will take a closer look at a straightforward console program. Even in such a simple application, it is possible to isolate layers. Of course, the important question that should be asked is „Is it worth to apply layers pattern?". For the needs of this article and to keep it simple we will deal

> layers will certainly help you
> to take control of a project
> and make its creation simpler

with a console application.

Our simple application will provide users with a simple dictionary that would be able to translate words from English to Polish. It should make possible to:

- add new words and their translations
- remove a word and its translation
- find a word and its equivalent
- show all the words and their translations:
  - in random order
  - alphabetically sorted
  - sorted according to dates when the words were added
- the application should persistently store its data.

One of the possible implementations of such a system may look like the one below:

```
package bnsit.layers.wordbook;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Date;
import java.util.List;
import java.util.Scanner;

public class Wordbook {

  private static String FILENAME = "wordbook.dat";

  public static void main(String[] args)
    throws FileNotFoundException, IOException, ClassNotFoundException
{

    List<DictionaryWord> words = loadData();

    boolean ok = true;
    Scanner s = new Scanner(System.in);

    System.out.println("Welcome to Wordbook.");

    while (ok) {
      System.out.print("dictionary> ");
      String line = s.nextLine();
      String [] arguments = line.split(" ");

      if ( line.startsWith( "search" ) ) {
        if ( arguments.length != 2 ) {
```

```
            System.out.println( "Command syntax: search <english_word>"
);
          } else {
            String englishWord = arguments[1];
            for (DictionaryWord word : words) {
              if ( word.getEnglishWord().equals(englishWord) ) {
                System.out.println( word );
              }
            }
          }
        } else if ( line.startsWith( "add" ) ) {
          if ( arguments.length != 3 ) {
            System.out.println(
            "Command syntax: add <english_word> <polish_word>" );
          } else {
            String englishWord = arguments[1];
            String polishWord = arguments[2];
            DictionaryWord dictionaryWord
              = new DictionaryWord(
                englishWord, polishWord, new Date());
            words.add( dictionaryWord );
            System.out.println( "Word added: " + dictionaryWord );
            writeData(words);
          }
        } else if ( line.startsWith( "delete" ) ) {
          if ( arguments.length != 2 ) {
            System.out.println(
              "Command syntax:delete <word_natural_number>");
          } else {
            int wordNumber = Integer.valueOf( arguments[1] );
            words.remove( wordNumber - 1 );
            writeData(words);
          }
        } else if ( line.equals( "show" ) ) {
          showList(words);
        } else if ( line.equals( "show sorted by name" ) ) {
          showList(words, new Comparator<DictionaryWord>() {
            @Override
            public int compare(DictionaryWord o1, DictionaryWord o2) {
              return o1.getEnglishWord()
                .compareTo(o2.getEnglishWord());
            }
          });
        } else if ( line.equals( "show sorted by date" ) ) {
          showList(words, new Comparator<DictionaryWord>() {
            @Override
            public int compare(DictionaryWord o1, DictionaryWord o2) {
              return o1.getDate().compareTo(o2.getDate());
            }
          });
        } else if ( line.equals( "exit" ) ) {
          ok = false;
        } else {
          System.out.println( "Command not found: '" + line + "'" );
        }
        }
        s.close();
    }

    private static void writeData(List<DictionaryWord> words)
```

```
        throws IOException, FileNotFoundException {
        ObjectOutputStream objectOutputStream
          = new ObjectOutputStream( new FileOutputStream( FILENAME ) );
        objectOutputStream.writeObject( words );
    }

    private static List<DictionaryWord> loadData()
      throws FileNotFoundException, IOException, ClassNotFoundException
{

        List<DictionaryWord> result = new ArrayList<DictionaryWord>();
        File file = new File( FILENAME );
        if ( file.exists() ) {
          ObjectInputStream objectInputStream
            = new ObjectInputStream( new FileInputStream( FILENAME ) );
          result   =   (List<DictionaryWord>)   objectInputStream.readO-
bject();
        }
        return result;
    }

    private static void showList(List<DictionaryWord> words) {
        int counter = 0;
        for (DictionaryWord word : words) {
          System.out.println( ++counter + " " + word );
        }
    }

    private static void showList(List<DictionaryWord> words,
        Comparator<DictionaryWord> comparator) {

        List<DictionaryWord> wordsCopy = new ArrayList<DictionaryWor-
d>(words);
        Collections.sort(wordsCopy, comparator);
        showList(wordsCopy);
    }
}
```

This is a typical application with so-called *flat architecture*. Of course, because the system is so simple, this solution has got a lot of advantages – it is clear, concise and easy to navigate when you browse its source code. However, when the system is developed, this type of solution will be getting more and more in maintenance. There will be more repetitions, the structure of the source code will be getting more and more complicated and the elements of the user interface and the data access part will get mingled.

Let us introduce layers

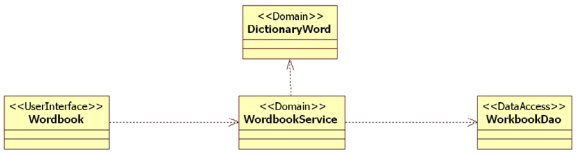
Is it possible to distinguish layers in such a simple system? Of course, it is! By looking at the application, we can separate elements that are responsible for:

- user interface (such as a user data input or showing information on the screen) – *Wordbook* class
- data processing (sorting, adding new words) – *WordbookService* class
- data access (saving to and loading from a file) – *WordbookDao* class.

All of these are shown in the following pic-

> when the system is developed, this type of solution will be getting more and more in maintenance

ture:



Let us take a look at the classes in order to determine their main characteristics from each layer. We will start with the user interface class – *Wordbook* (the source code below). This class, compared to the code from the previous version, has got specifically identified responsibility which is the interaction with the user. Only functions related to handling console and delegating specific tasks to *WordbookService* has been left there, which in this case represents domain layer. *Wordbook* class:

- retrieves data from the console
- validates and analyzes user input data
- shows appropriate messages
- delegates concrete operations.

Please note that there is no single line of code related to the internal logic of the system, there is only the user interface. Summing up, the class was reduced so that it plays a single role in the system.

```
public class Wordbook {
  private WordbookService wordbookService = new WordbookService();

  public static void main(String[] args) {
    Wordbook wordbook = new Wordbook();
    wordbook.run();
  }

  public void run() {

    boolean ok = true;
    Scanner s = new Scanner(System.in);

    System.out.println("Welcome to Wordbook.");

    while (ok) {
      System.out.print("dictionary> ");
      String line = s.nextLine();
      String [] arguments = line.split(" ");

      if ( line.startsWith( "search" ) ) {
        if ( arguments.length != 2 ) {
          System.out.println(
              "Command syntax: search <english_word>" );
        } else {
          String englishWord = arguments[1];

          List<DictionaryWord> words
            = wordbookService.find( englishWord );
          for (DictionaryWord word : words) {
              System.out.println( word );
          }
```

```
            }
        } else if ( line.startsWith( "add" ) ) {
            if ( arguments.length != 3 ) {
                System.out.println( "Command syntax: "
                    + "add <english_word> <polish_word>" );
            } else {
                String englishWord = arguments[1];
                String polishWord = arguments[2];
                DictionaryWord dictionaryWord
                  = wordbookService.createNewWord(
                      englishWord, polishWord);
                System.out.println( "Word added: " + dictionaryWord );
            }
        } else if ( line.startsWith( "delete" ) ) {
            if ( arguments.length != 2 ) {
                System.out.println( "Command syntax: "
                    + "delete <word_natural_number>" );
            } else {
                int wordNumber = Integer.valueOf( arguments[1] );
                wordbookService.remove( wordNumber );
            }
        } else if ( line.equals( "show" ) ) {
            List<DictionaryWord> words = wordbookService.findAll();
            showList(words);
        } else if ( line.equals( "show sorted by name" ) ) {
            List<DictionaryWord> words
              = wordbookService.findAllSortedByName();
            showList(words);
        } else if ( line.equals( "show sorted by date" ) ) {
            List<DictionaryWord> words
              = wordbookService.findAllSortedByDate();
            showList(words);
        } else if ( line.equals( "exit" ) ) {
            ok = false;
        } else {
            System.out.println( "Command not found: '" + line + "'" );
        }
    }
    s.close();
  }

  private void showList(List<DictionaryWord> words) {
    int counter = 0;
    for (DictionaryWord word : words) {
      System.out.println( ++counter + " " + word );
    }
  }
}
```

Specific operations are delegated to WordbookService class which deals with the main tasks related to system functions. On the other hand, operations of persistent storing or data searches are delegated to another object – *WordbookDao*.

Let us now have a look at *WordbookService* class. What is important to notice?

1.The methods in this class respond to the system's functions, e.g. find, delete, find all.

2.All the methods are short and concise.

3.The methods do not depend on the user interface, so they can cooperate with any user interface!

```
public class WordbookService {

  private WordbookDao wordbookDao = new WordbookDao();

  public List<DictionaryWord> find(String englishWord) {
    return wordbookDao.find(englishWord);
  }

  public DictionaryWord createNewWord(String englishWord, String po-
lishWord) {
    DictionaryWord dictionaryWord
      = new DictionaryWord( englishWord, polishWord, new Date());
    wordbookDao.save(dictionaryWord);
    return dictionaryWord;
  }

  public void remove(int wordNumber) {
    DictionaryWord dictionaryWord
      = wordbookDao.findByIndex( wordNumber - 1  );
    wordbookDao.remove(dictionaryWord);
  }

  public List<DictionaryWord> findAll() {
    return wordbookDao.findAll();
  }

  public List<DictionaryWord> findAllSortedByName() {
    List<DictionaryWord> words = wordbookDao.findAll();
    Collections.sort(words, new Comparator<DictionaryWord>() {
      @Override
      public int compare(DictionaryWord o1, DictionaryWord o2) {
        return o1.getEnglishWord().compareTo(o2.getEnglishWord());
      }
    });
    return words;
  }

  public List<DictionaryWord> findAllSortedByDate() {
    List<DictionaryWord> words = wordbookDao.findAll();
    Collections.sort(words, new Comparator<DictionaryWord>() {
      @Override
      public int compare(DictionaryWord o1, DictionaryWord o2) {
        return o1.getDate().compareTo(o2.getDate());
      }
    });
    return words;
  }
}
```

4.Operations that depend on a specific data source are delegated to *WordbookDao* class.

Let us look at the *Wordbook* class. There are a few elements which require your attention:

1.This class is responsible for cooperation with data and the data source (in this case it is a file that contains serialized data).

2.Methods in this class represent basic operations related to the system data.

3.Methods are short, precise and their responsibilities are clearly defined.

```
public class WordbookDao {

  final private String FILENAME = "wordbook.dat";
  private List<DictionaryWord> words = null;

  public WordbookDao() {
    words = loadData();
  }

  public List<DictionaryWord> find(String englishWord) {
    List<DictionaryWord> result = new ArrayList<DictionaryWord>();
    for (DictionaryWord word : words) {
      if ( englishWord.equals(word.getEnglishWord()) ) {
        result.add(word);
      }
    }
    return result;
  }

  public DictionaryWord findByIndex(int i) {
    return words.get( i );
  }

  public List<DictionaryWord> findAll() {
    return new ArrayList<DictionaryWord>(words);
  }

  public void save(DictionaryWord dictionaryWord) {
    words.add(dictionaryWord);
    writeData(words);
  }

  public void remove(DictionaryWord dictionaryWord) {
    words.remove( dictionaryWord );
    writeData(words);
  }

  private void writeData(List<DictionaryWord> words) {
    ObjectOutputStream objectOutputStream;
    try {
      objectOutputStream = new ObjectOutputStream(
          new FileOutputStream(FILENAME));
      objectOutputStream.writeObject(words);
    } catch (Exception e) {
      throw new WordbookDaoException(e);
    }
  }

  private List<DictionaryWord> loadData() {
    List<DictionaryWord> result = new ArrayList<DictionaryWord>();
    File file = new File(FILENAME);
    if (file.exists()) {
      ObjectInputStream objectInputStream;
      try {
        objectInputStream = new ObjectInputStream(
            new FileInputStream(FILENAME));
        result =
          (List<DictionaryWord>) objectInputStream.readObject();
      } catch (Exception e) {
        throw new WordbookDaoException(e);
      }
    }
    return result;
  }
}
```

> This class, compared to the code from the previous version, has got specifically identified responsibility which is the interaction with the user.

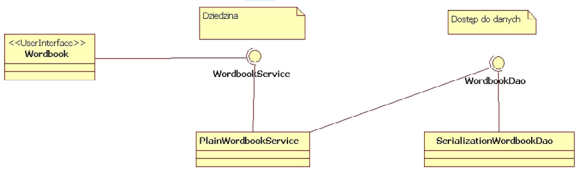
4.Thanks to the fact that the data access has been encapsulated, it is easy to change the way the data is written (e.g. to a XML file) and it will not affect the rest of the application.

That is how we have managed to split the application into layers. What are the consequences of this step? Well, responsibilities are now clearly defined, the application is ready for further changes, the code has become easier to manage and better organized – it is clear where to look for the specific elements. On the other hand, the application structure is now more complex – there are three classes instead of one. Also, the application performance may be now worse. Well, there is no rose without a thorn. In simple systems, which are composed from less than twenty classes, such an approach would probably consume too much work. In bigger systems, however, it would make their structure clarified and would make navigation much more easy.

Layers switching

One of the main features of the layers concept is the possibility of switching layers almost in the real-time. This allows you to dynamically adjust your solution wrapped in one of your layers with a very little impact on the other parts of a system. And this is the magic of the layers concept.

In order to do this, we should make our system more flexible. At the moment the classes are strictly tied to each other. We will use two techniques to make them loosely coupled – these are: interfaces for the classes that are in a layer and Dependency Injection pattern. Both of these will simplify changing of the application dependencies. The system will look like this:




```
public class Wordbook {
  private WordbookService wordbookService = null;

  public static void main(String[] args) {
    Wordbook wordbook = new Wordbook();

    PlainWordbookService plainWordbookService =
      new PlainWordbookService();
    plainWordbookService.setWordbookDao(
      new SerializationWordbookDao());
    wordbook.setWordbookService(plainWordbookService);

    wordbook.run();
  }
  // ...
  public WordbookService getWordbookService() {
    return wordbookService;
  }

  public void setWordbookService(WordbookService wordbookService) {
    this.wordbookService = wordbookService;
  }
}
```

> Thanks to interfaces, system elements are now loosely coupled

Thanks to interfaces, system elements are now loosely coupled and we can change their concrete implementations.

As you can see in the picture, *Wordbook* class implements *WordbookService* interface, which means that it is possible to insert any implementation in its place (that is based on POJO or EJB). To enable dependency injection, we have added getters and setters and put a code that builds related classes in the method *main*.

```
public class Wordbook {
  private WordbookService wordbookService = null;

  public static void main(String[] args) {
    Wordbook wordbook = new Wordbook();

    PlainWordbookService plainWordbookService =
      new PlainWordbookService();
    plainWordbookService.setWordbookDao(
      new SerializationWordbookDao());
    wordbook.setWordbookService(plainWordbookService);

    wordbook.run();
  }
  // ...
  public WordbookService getWordbookService() {
    return wordbookService;
  }

  public void setWordbookService(WordbookService wordbookService) {
    this.wordbookService = wordbookService;
  }
}
```

By analogy, we have corrected *PlainWordbookService* class which implements *WordbookService* interface.

```
public interface WordbookService {
  public abstract List<DictionaryWord> find(String englishWord);
  public abstract DictionaryWord createNewWord(String englishWord,
      String polishWord);
  public abstract void remove(int wordNumber);
  public abstract List<DictionaryWord> findAll();
  public abstract List<DictionaryWord> findAllSortedByName();
  public abstract List<DictionaryWord> findAllSortedByDate();
}

public class PlainWordbookService implements WordbookService {
  private WordbookDao wordbookDao = null;
  public WordbookDao getWordbookDao() {
    return wordbookDao;
  }

  public void setWordbookDao(WordbookDao wordbookDao) {
    this.wordbookDao = wordbookDao;
  }
}
```

```
public interface WordbookDao {

  public abstract List<DictionaryWord> find(String englishWord);

  public abstract DictionaryWord findByIndex(int i);

  public abstract List<DictionaryWord> findAll();

  public abstract void save(DictionaryWord dictionaryWord);

  public abstract void remove(DictionaryWord dictionaryWord);

}

public class SerializationWordbookDao implements WordbookDao {

  final private String FILENAME = "wordbook.dat";
  private List<DictionaryWord> words = null;

  public SerializationWordbookDao() {
    words = loadData();
  }

  public List<DictionaryWord> find(String englishWord) {
    List<DictionaryWord> result = new ArrayList<DictionaryWord>();
    for (DictionaryWord word : words) {
      if ( englishWord.equals(word.getEnglishWord()) ) {
        result.add(word);
      }
    }
    return result;
  }

  // ...

  private List<DictionaryWord> loadData() {
    List<DictionaryWord> result = new ArrayList<DictionaryWord>();
    File file = new File(FILENAME);
    if (file.exists()) {
      ObjectInputStream objectInputStream;
      try {
        objectInputStream = new ObjectInputStream(
            new FileInputStream(FILENAME));
        result = (List<DictionaryWord>) objectInputStream.readO-
bject();
      } catch (Exception e) {
        throw new WordbookDaoException(e);
      }
    }
    return result;
  }
}
```

In a similar way, *WordbookDao* class has been modified. You can see its fragment below. The whole source code for this article can be downloaded here: http://www.bnsit.pl/rozwarstwienie/.

Now the data access layer in our sample application works with a file with serialized data. It is very easy to develop some other solution by implementing *WordbookDao* interface, for example, based on JDBC.

> Layers are not a cure-all drug,
> that will overcome all obstacles in software design.

Then, the same application, with no major changes in the user interface and domain layers, will work with database! You can treat it as an exercise, the correct answer can be found on the following website: http://www.bnsit.pl/rozwarstwienie/.

The approach described in this article may be applied to more complex systems.

Testing

The next profit from using layers can be noticed when we are testing our system. To spot that it is enough to compare the initial and the final version of the sample program. Which is easier to test? Perhaps the monolithic code that contains several alternative paths mixed together with user interface, domain and data access functions? Or maybe classes that contain small methods with clearly defined responsibilities? The unit testing becomes a pleasure.

Summary

Layers are not a cure-all drug, that will overcome all obstacles in software design. In complex systems they are indispensable if they are to be effectively developed. However, each layer constitutes an additional level of complexity. In the case of minor applications it is a decision of an individual, which is worth-taking if you want to develop your application in a structuralized way. At this stage it is up to you how many and what kind of layers you want to apply. Good luck with your experimentation!

About me

Mariusz Sieraczkiewicz

Trener, consultant, IT project manager, coach. Founder of Equilibrium team of developers. Co-founder of JUG Łódź. Author of articles about software engineering. Co-owner of training company BNS IT. Happy husband :-) http://www.linkedin.com/pub/2/a24/812

GroovyMag Review - June 2009

Krzysztof Konwisarz

GroovyMag is a magazine created by Groovy and Grails users and enthusiasts. It has been publishing every month since November 2008 as a pdf.

Batch processing with Spring Batch

Author: Bob Brown

Spring Batch is one of the youngest part of Spring family. This Framework allows batch processing of huge amount of data providing tools for read, validating and write data, logging, tasks management and resource management as well as parallel computing. Presented example is Groovy application that converts data from hard to read text format to XML.

Groovy under the Hood – Groovy Scripts

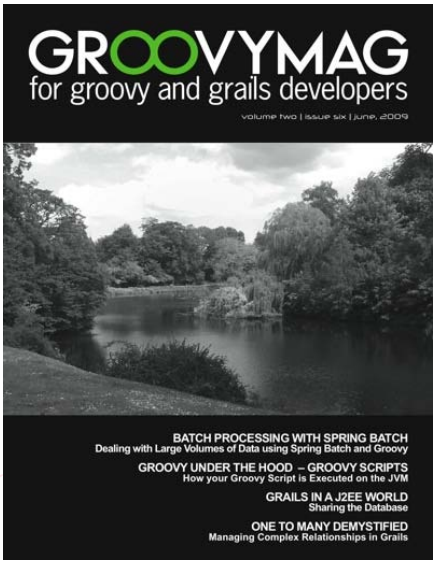
Author: Kirsten Schwark

This time, in this recurrent section, we can learn how scipts in Groovy are implemented. We can read how they are executed and how creators treated local variables and properties in the scripts.

Grails in a J2EE World - Sharing the Database

Author: Shawn Hartsock

If the database can be used by many users, it can be used also by many applications. Making this sentence happen, author shows us that tough task that is migration to new platform can be easy with Grails. He starts with optimistic locking on old system (in this example it is JBoss), moving forward to mapping DB to domain classes in GORM.



One-to-Many Demystified – Managing Complex Relationships in Grails

Author: Tyler Williams

Default GSP pages generated by scaffolding in Grails are very rarely good enough for production systems. It is even worse for views that need to work on both ends of one-to-many relations. Author shows us how this problem can be solved using modal windows and table component. He also propose his own solution that is the best for his example, based on HTML and AJAX.

Plugin Corner – Grails Functional Test Plugin

Author: Dave Klein

Grails Functional Test is an alternative to functional tests with Canoo Webtest and Selenium. Its advantages are great integration with Grails testing framework and clear code created. Similar to Canoo Webtest it is using HtmlUnit to emulate web browser.

About Author

Student of Comuter Science at Physics Faculty, Adam Mickiewicz University in Poznan.

A new website will be launching in September at the address: http://vriltek.com
together with an English version.
Prepare for extremely good offers and a variety of payment options.

VRILTEK.COM

Logos

Posters

Cards

Websites

DTP

E-stores

Leaflets

Banners