FAKULTÄT FÜR !NFORMATIK

Design and prototyping of a flexible value-at-risk measurement system

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Philipp Guggenberger
Matrikelnummer 0303703

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Ao.Univ.-Prof. Mag. DDr. Thomas Dangl

Wien, 20.06.2008

(Unterschrift Verfasser) (Unterschrift Betreuer)

Contents

1 Introduction

Finance is probably one of the most dynamic business sectors. In short time intervals more and more complex financial products are developed. A mature risk management requires the risk quantification and analysis of all investments in a portfolio. Risk managers are supported by software systems, which evaluate the risk (in different ways) for the single products and portfolios. Risk evaluation systems are very dynamic applications and it's a challenge for all involved persons to keep it up to date. All new types of investments have to be integrated in the system. Furthermore it should be possible to improve existing evaluation methods on the fly without jeopardize the consistency of the system. The objective of this work is the design and partly implementation of a flexible risk measurement system. The term "flexible" has different meanings. First, it should be possible to execute the system on multiple operating systems. There should also be a maximum flexibility in the variety of used programming languages. The extensibility and improvement of the system functionality should be very easy, and shouldn't have any negative impact on already existing parts of the system. Finally it should be possible to distribute the single modules on different servers to handle workload peaks.

Two very interesting technologies, which can handle the requirements, are SOA (service oriented architecture) and EDA (event driven architecture). The basic elements in both approaches are services. A service is an independent module with a clearly defined interface that provides a specific functionality. The independence of a service allows an easy reusability in any other project and the use of several different programming languages to implement it. The task of the orchestration process is to form a complete application by combining the single services. Here is the biggest difference between SOA and EDA. Whereas SOA has a clearly defined and in the most cases synchronous workflow, EDA reacts more dynamically, triggered by events. This topic will be discussed in Chapter 2.

Chapter 4 covers the market risk evaluation of financial products. In this work the very popular risk evaluation approach value-at-risk (VAR) will be discussed. Value-at-risk can be defined as the expected maximum lost over a predefined period within a specified confidence interval. First, the most common approaches to calculate VAR will be described. After that a more complex topic will be discussed, the risk evaluation of non-linear investments, and here especially the evaluation of European stock options.

Chapter 3 describes the basic architecture of the system. The application consists of different modules, which cover a specific functionality and contain a couple of web services (for example: database access components, risk evaluation modules (evaluating the risk of the different types of investments), etc.). Another important part of this chapter is the database design. The challenge is to model the database in a way that allows an easy extensibility and change in future iterations. Last but not least the process and workflow management will be described. In general, every process (depending on the grade of granularity) can be imple-

mented as service. The workflow of the single processes is the basis for the orchestration of the application and is shown in three examples in Section 3.3.2 and 3.5.

SOA and especially EDA are rather new technologies. Fowler wrote about EDA ”the architect’s dream, the developer’s nightmare” [HOH06]. There cannot be found any reliable field reports about EDA, yet. That’s one of the reasons why it’s interesting to implement a prototype. On the one hand it’s a good opportunity to prove the design of the system, on the other hand it’s possible to test this type of development in a practical context and find out if Fowlers statement is true. The prototype consists of several components. The objective is to implement the workflow shown in Figure 3.5 and here is the focus on the evaluation of stocks, stock options and portfolios with help of the delta-gamma approach as well as with the Monte-Carlo simulation approach. Chapter 5 describes some special features of the prototype, the used infrastructure and presents the test results of the different approaches.

The last chapter summarizes the development process and discusses the advantages and problems of the used technologies.

2 Service oriented architecture

2.1 Concept and idea

In the history of software-development you will find a continuous evolution of programming paradigms. From the first algorithm written in machine language, to state of the art like service oriented programming and aspect oriented programming, it was a long way with a continual improvement of simplicity and reusability.

In the last few years service oriented architecture (SOA) becomes more and more popular. SOA offers a new way of software development. SOA itself is no new technology, it's more a new type of architecture. Gannod, Burge und Urban have the following definition: "design of SOA systems involves the description of atomic and composite processes (services) and the orchestration of those processes to form a distributed application". [GBU07] The core components of this architecture are services: "a service provides a logically coherent piece of functionality to its clients." [BGL07] The main task of a SOA infrastructure is to coordinate these independent parts and build a complete solution. A very popular type of service implementations are web services. It's not the only way to implement a service, but the most common. On the next pages I want to take a closer look to services in general and web services in detail. The following questions should be answered:

- What are the advantages of services?
- How do they work?
- What are the problems?

2.1.1 Web services

A service is an independent software component, which provides a defined functionality. To do this, the required input and output parameters have to be specified. With the use of services, code reusability has a completely new dimension. The object-oriented programming paradigm gives us the chance to write a class or more general a package once and use it in many other projects. This works fine if the same development platform and the same operating system are used. But if I wrote my package for example in Java and the new project is developed with Microsoft .net it's not easy to use the functionally of the Java-class in the new project. Another example: a software application will be developed by an inter-institutional development-team, which has different programming skills. [PS06] So it's not practical and reasonable to retrain part of the team. With the service oriented approach everybody can use his familiar development environment, which saves costs and time. As aforementioned the

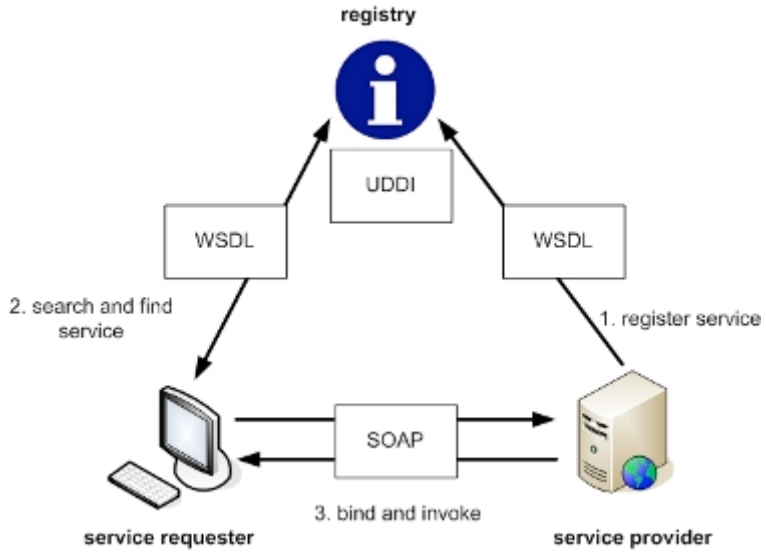

Figure 2.1: Use of web services in SOA

most popular way to do this, is the use of web services. Modern programming languages like Java, Microsoft .net, etc. already support this technology. One of the reasons of the success of web services is the standardization. A web service can be split into three parts:

- the communication protocol SOAP (simple object access protocol)
- the interface description language WSDL (web service description language)
- and the web service registration component UDDI (universal discovery, description and integration)

The relationship between these three parts is illustrated in Figure 2.1.

SOAP describes the structure of a message to exchange information between two parties, but it doesn't handle the transport of the message. For this task several protocols are practical like HTTP, SMTP, FTP, etc. whereas HTTP is mostly used. The message is written in XML. The advantages of this format are the standardization and the possibility of easy reading (e.g. with XPath [W3C99]). The disadvantages are the message overhead (which implies a larger message size in comparison with binary data) and the long parsing time. In literature you will find different interpretations of pro and contras, so I will discuss these facts again in Chapter 2.2. Now it's important to know, what is a message? Hohpe and Woolf give the following definition: "A message is an atomic packet of data that can be transmitted on a channel. (...) It can be interpreted simply as data, as the description of a command to be invoked on the receiver, or as the description of an event that occurred in the sender" [HW03]. A message can be separated into a header and a body. The header contains meta-information about the sender, the receiver, which kind of message it is, etc. The body itself contains the actual data, it could be a whole object, a string, or other fundamental variables. The most common development frameworks already provide message-objects in their core libraries, which can be used for the own data transfer.

The world wide web consortium describes WSDL as follows: ”Web Services Description Language Version 2.0 (WSDL 2.0) provides a model and an XML format for describing Web services. WSDL 2.0 enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as ”how” and ”where” that functionality is offered.” [W3C07]

In other words WSDL describes the interface of a web service. If I want to use an unknown service it's recommended to take a closer look to the corresponding WSDL document. In detail a WSDL document (version 2.0) [1] consists of four parts:

- types element
- inteface element
- binding element
- service element

The types element defines the data types of the in- and output messages of a service. It could contain simple types like string, double, etc. or complex types. In the latter case it's possible to implement an own structure. For example if the result of a web service consists of three variables, the type element will look like this:

```
<s:complexType name="Riskresult">
 <s:sequence>
    <s:element name="Currency" type="s:string"/>
    <s:element name="Duration" type="s:float"/>
    <s:element name="Value" type="s:double"/>
 </s:sequence>
</s:complexType>
```

The interface element includes all provided methods of a service and assigns the defined types to the corresponding in- and output message. With the binding element the used transport protocol (in the most cases it will be HTTP) can be defined. Last but not least the address where the service can be accessed must be specified within the service element. For a more detailed description, take a closer look to the W3C WSDL 2.0 specification [W3C07]. In the most cases software development tools like Microsoft Visual Studio already generate the WSDL document automatically. But it's recommended to control these settings to guarantee a well-formed and ”standard-based” document which could be handled by other platforms.

The basic idea of UDDI was to build a kind of dictionary for web services. It should be possible to register provided web services or search for already existing services. For example a bank wants to open its currency converter service to public. So it can register this service in a UDDI registry. On the other hand a software developer wants to develop a web shop with international support and needs the up-to-date spot rates. So he can search in the UDDI for a matched service and integrate it in his application. The idea of a self-administrated directory where everybody can post his services was not enforceable. The biggest problems were the

[1] here are some serious changes in comparison with the previous version 1.1

missing quality criteria and spam. As a consequence of these problems IBM, Microsoft and SAP took their public UDDI-directories offline[2]. Another situation is to use UDDI within an organisation. In a service oriented infrastructure there exist a multitude of web services. As mentioned several times, one of the reasons to use services is the possibility of code reusability. Due to this a central dictionary is necessary, where UDDI is a good way to realize this requirement.

There are two different mechanisms to invoke a web service, synchronous and asynchronous. SOAP was originally designed for synchronous message exchanges [PS06]. This means that the service requester sends its message to the receiver and waits until the result is replied. This approach works fine if the result could be transferred or calculated in a short time. In all other cases the client is locked for the whole length of time. In a service oriented infrastructure with several services where the most business processes will be invoked parallel this way is not practical. A better solution will be to call the web services asynchronous. Hohpe and Woolf [HW03] describe this mechanism as ”send-and-forget” approach. The requester sends a message to the service provider and gets (optionally) a notification if the result is available. This functionally is not defined in the web service specification of W3c but is implemented in some frameworks or middleware. This missing specification could be a problem if you use services which run on different frameworks (actually one of the basic ideas from web services).

2.1.2 Orchestration

Now I want to come back to the main topic of this chapter, the service oriented architecture. If you implement an application with the service oriented paradigm, you have to invoke many different services which are normally provided on different sources. If we imagine an application as a business process workflow, where a service represents one process, we need a method to coordinate these workflows. The two buzzwords are orchestration and choreography s. Fig. 2.2. The orchestration describes the workflow of web services from the calling view. The choreography has a more global approach, and shows the message interactions between two or more participants. One of the most established orchestration languages is BPEL (Business Process Execution Language), founded by OASIS with the support from Microsoft, IBM, SAP, Siebel, etc. OASIS specifies BPEL as follows: ”BPEL Business Processes offer the possibility to aggregate web services and define the business logic between each of these service interactions.” [OAS07] BPEL also uses XML for the process definition. It’s a powerful language so I only want to show the most important activities of BPEL:

- invoke: provides the functionality to invoke a web service synchronous or asynchronous
- receive: in the most cases it’s the entry point of a service oriented implementation. It receives a request from another service.
- reply: returns a message (synchronous) back to the calling instance (see receive).

[2] http://webservices.sys-con.com/read/164624.htm (last access 10/19/2007)

Figure 2.2: Hierarchy of SOA-technologies

- assign: manipulate a variable, for example to map the result from one web service to the input parameter of the next invoked service.
- empty: doing nothing, it is used as placeholder for further processes or to represent empty branches, e.g. in an if-else activity
- wait: waits for a specified time that a process comes to an end, e.g. a marketplace service wants to find the cheapest price of a commodity, so the market participant have time for ten minutes to transmit their prices. The service will wait for ten minutes and continue with the next process to find the cheapest vendor.
- sequence: a sequence is a container, where you can insert your activities (invokes, replies, receives, etc.) which are executed in a sequence order.
- flow: could also be seen as container, it provides the functionality of parallel process execution. All included activities will be started at the same time (when the flow process is initialized).
- if-else: is equal to the if/else construct from other programming languages.
- while/repeatUntil/forEach: allows to execute a collection of activities, more than one times
- partnerLinks: define the communication channel to the responsible web services.

For a more detailed description of the available activities including the syntax and semantic specification I refer to the Web Services Business Process Execution Language Version 2.0 primer document from OASIS [OAS07]. By now there exist a lot of BPEL-design tools and BPEL-runtimes, which are also integrated in popular middleware like IBM WebSphere, Oracle Business Process Analysis Suite and many more. With these design-tools it's possible to design the workflow in a graphical way, so it's not necessary to write the activities with XML, it's generated automatically. Another concept, even if it's quite similar to BPEL, pursues Microsoft with the Windows Workflow Foundation (WWF) which is included in Microsoft .net 3.0. Microsoft uses its own format called XAML (Extensible Application Markup Language)[3] instead of BPEL, but it offers a BPEL-add-on to use the BPEL syntax inside a .net project.

[3] formerly also called XOML

In the next subchapter I will show a short SOA example which is implemented with the MS WWF and the BPEL plug-in and want to discuss the advantages and disadvantages of this tool combination.

2.1.3 Example

For a better understanding I want to give a short example. The objective is to write an application which calculates the price and the risk of a forward rate agreement (FRA). To do this there are five input parameters required:

- currency
- termination date
- end date
- is it a short or a long position
- nominal amount

First of all, the required services have to be defined. A forward rate agreement can be split into two zero-coupon bonds. For a better illustration I want to show a short example. I will receive € 100000 in three month. This money will be used for a payment in 6 month, but I want to fix the interest rate already now. Therefore are three dates significant for calculation, called $t0$, $t1$ and $t2$. $t0$ is the conclusion date of the agreement. $t1$ is the termination date of the forward-rate agreement (in the example it's $t0$ + 3 month), $t2$ is the end date ($t0$ + 6 month). A FRA long position is equivalent to a zero-coupon bond short with duration from $t0$ to $t1$ and a zero-coupon bond long with duration from $t0$ to $t2$. That's the first needed service, named calculateBond, with the input parameters: currency, start date, end date, short/long, the nominal amount and a Boolean variable if it's a progressive or regressive calculation (progressive: the value at $t0$ is known and the value at $t1$ is the result; regressive: the value at $t1$ is required, I want to know which amount of money I have to invest at $t0$). The output has the following structure:

```
<calculateBondResult>
    <marketValue>double</marketValue>
    <riskfactors>
      <riskfactor>
        <currency>string</currency>
        <maturity>double</maturity>
        <value>double</value>
      </riskfactor>
      <riskfactor>
        <currency>string</currency>
        <maturity>double</maturity>
        <value>double</value>
```

```
      </riskfactor>
    </riskfactors>
</calculateBondResult>
```

The result is the fair value of the zero-coupon bond, and an array of risk factors, where the value of the bond will be mapped to the relevant risk factors (for example the maturity of a zero-coupon bond is 2.5 years, so it's common to split the value to the next fitting risk factors e.g. the 2 and the 3 year interest rate). Now the price of the forward rate agreement is known, the next step will be to calculate the risk of the investment. To do this another service is required which is called calculateVaR. The input parameters are a collection of risk-factor-items (see output of calculateBond), the output is the value-at-risk (see Chapter 4.1, calculated with the analytic approach) of the forward rate agreement. After defining the services, it's necessary to orchestrate these independent modules, to form a complete application. In this example I want to use the Microsoft .net framework (version 2.0) to implement the web services and the Windows Workflow Foundation (part of the .net Framework 3.0), to orchestrate the activities. For a more general approach I tried to use the BPEL add-on[4] instead of the unstandardized Microsoft format. The workflow is defined as follows (see also Figure 2.3): At first the user has to enter the required input parameters. After that a zero-coupon bond will be valued with the start date $t0$ and the end date $t1$, with a regressive calculation approach (how much money will I get now, if I have to pay back the nominal amount at $t1$). Afterwards the result of this calculation will be used as input parameter for the next invoke. For this, it's necessary to assign the previous result to the input variable of the next web service, which calculates a zero-coupon bond in a long position from $t0$ to $t2$. The result is the forward rate. Now the risk factors of both calculations will be merged and are the input parameter for the next step, the value at risk calculation. At the end the results will be displayed. The application itself works fine, but there are several points of criticism. If you are using the BPEL plug-in, MS Visual Studio doesn't store the workflow with the standardize BPEL syntax, furthermore it uses its own XAML format. Microsoft offers an import/export tool, but when I tried to convert the XAML file into a BPEL-file, other tools like Active BPEL [5] weren't able to read the file (there was a syntax error in the exported file. After fixing this bug manually, I still wasn't able to open it). The current version of the BPEL-plugin doesn't allow an active use of this framework. If a standardized technique is preferred, another platform should be chosen.

2.2 Advantages and problems

In this chapter I want to discuss the advantages and the problems of a service oriented architecture. The main advantage of SOA is the possibility of the reuse of already implemented functionality. All services have a predefined interface and a clearly defined functionality. Due to these properties the services have to be independent, or to use another popular term, they have to be loosely-coupled. The services themselves could be seen as a black-box. The requestor doesn't know what framework is used, or in which programming language the service

[4] BPEL for Windows Workflow Foundation - March 2007 Community Technology Preview
[5] www.active-endpoints.com

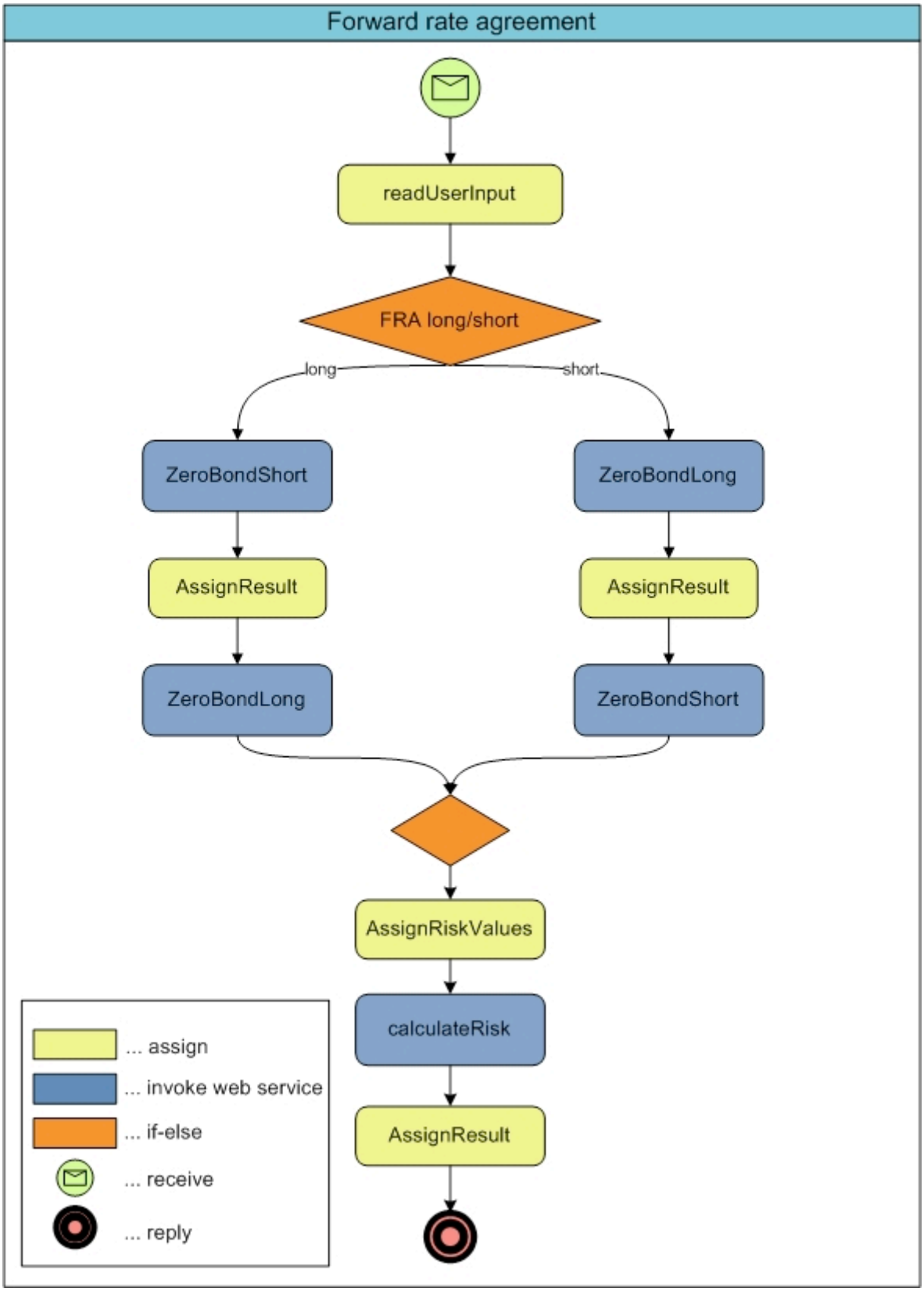

Figure 2.3: SOA workflow for a forward rate agreement calculation

is implemented. He just calls the service with the required parameters and gets the results back. Due to the platform and programming language independence, it is possible to include any existing services in a software solution. For example it's no problem to provide a method, maybe as part of a statistical toolbox web service, which calculates the statistical moments of a time series. Every software developer (it doesn't care which framework is in use), could use this toolbox. We could go a step further, the described statistical moment problem isn't really complex, but it has to be implemented and tested. Any simple mathematical software suite could solve this problem with few commands. So there are no reasons why we shouldn't use the existing functions. For example with Mathworks MatLab[6] the following approach is feasible. First of all it is necessary to implement an own function, which uses the existing methods from MatLab and solve our problem. The next step could be to compile the function (e.g. with the MATLAB Compiler associated with the MS Visual Studio 2005 C++ compiler), the result will be a dll-file. This library file can be included in most windows based frameworks, but it is not always easy to use it on other operating systems. As already discussed in Section 2.1.1, a web service allows a platform-independent use. Building on that, a solution would be to implement a web service (using the methods of the dll) hosted on a windows web server. From now on every application has access to the methods written in MatLab.

Next, I will take a closer look to the granularity of a service. For a software developer it is fine if he could choose from a pool of existing services. But the call of web service will take a longer time than the call of a local method. This has several reasons. At first, services are normally hosted on a remote computer. So you have a longer transmission time and the possibility of transmission errors. Bierhoff, Grechanik and Liongosari describe several other problems in their paper "Architectural mismatch in service-oriented architectures" [BGL07]. Due to the characteristic of a loosely coupled service on the one hand, and the requirement of holding the state of an application on the other hand, it's necessary to transmit a huge amount of data to the single services. A service is not just written for one project, furthermore it should be implemented as general as possible to cover a large problem domain. But this implies that the service has to provide a huge amount of input and output parameters, which are not needed by each client and, thus, the transmitted messages are larger than necessary. Due to these problems it's inevitable to find a trade-off between the runtime of an application and the reusability of existing software components. The finer the granularity of the used services, the more invokes have to be executed and the longer is the runtime of the software. On time-critical components it is better to decrease the number of services and choose services which provide a larger functionality. Otherwise are these large packages maybe useless for any other applications, the advantages of services don't exist anymore. [GBU07] have also analysed the overhead of SOAP and XML on their physics-simulation project and found out that it was less than 5%. However, they used services with a larger functionality where the execution of each call takes a long time, so the message overhead is negligible. Another "inefficiency" can be found at the message model used by SOA. Each service-interface defines its own of input and output parameters and in the most cases its own complex data types too. Therefore it's required to convert the input and output parameters in a uniform format, which is a time expensive procedure. The same problem exists with ambiguous data type formats

[6] www.mathworks.com

like date/time data types or float numbers. In this case the variables have to be converted into the right format.

In literature you can find the argument that the XML-messaging format is one of the bottlenecks in a service oriented architecture. [BGL07] wrote [7]: "Studies have shown that XML messages are typically 10 to 50 times larger than their binary counterparts and that XML-related tasks such as parsing, transformation and serialization consumed over 93 % of total processing of typical XML documents". In my opinion it's true that XML is (on performance view) not the best choice for the message handling. On the other hand it is an established data format with a standardized syntax, which is very simple to handle. In a market with so many players it's not easy to establish a standard format. The advantages of readable data (e.g. it's no problem to create the WSDL-file manual) and the fact that it can be used on all service oriented based frameworks outweigh the disadvantages. This is all the more so, when the used services have a coarser granularity, this affects less traffic and thus less influence of the XML bottleneck.

The use of third party services is one of the key elements of SOA. Therefore quality of service is a very important point. Normally more than one application uses the provided services and the requests will be executed one by one. If every requestor has the same priority and the request is synchronous, it could happen that the whole application has to wait for one result. Another problem can be found on static linked services. All used services have to be defined at compilation time. But what happens if one of these services is out of order? In general this situation would also mean a break down of the own application. The situation will be even worse, if the service is not in responsibility of the own organization, in other words there exists a large dependency. To avoid these scenarios it's a good way to cover the maximum allowed response time and the availability of a third party service by service level agreements (SLA).

Not really a problem, but a challenge for software architects and developer is the asynchronous transaction approach. Requests will be sent to the service provider, but it's not clear when the response will be received. During this time period the application can continue the procedure. This means, first, that the requestor has to remember the state of the time point of calling the service. Second, he cannot specify the order of the callbacks. If he sends three service requests, it's possible that the result of the last request will be received at first. Due to this nondeterministic behavior a different style of programming is needed unlike the sequential programming methods. [HW03] wrote also that it's very difficult to debug such applications.

As aforementioned one of the biggest problems are static linked web services. Also if the design enables a dynamical switch of the service at runtime, the services have to be known, and administrated by human beings. A more flexible way will be discussed in the next chapter, the event driven architecture approach.

[7] they refer to www.zapthink.com/report.html?id=WP-0137

2.3 Orchestration - An event-driven approach

After discussing the service oriented architecture, we want to go one step further and take a look to the event driven architecture, which can be seen as an extension of SOA. The workflow design of a service oriented architecture doesn't differ much from regular development paradigms like the procedural or object oriented programming. The general principle is always the same and bases on a sequential approach. For example after finishing task A, call task B. If the result of B is greater 0, call procedure E otherwise procedure D. This approach requires the knowledge of every used function and service, respectively, already at the time of development.

EDA has a different philosophy that requires a completely different thinking for all involved persons in the software development process. Dheap and Ward illustrated the power of EDA in a very good example in their paper "Event-Driven Response Architecture for Event-Based Computing" [DW05]. It demonstrates how an event-driven environment can solve complex tasks. At the beginning of this section, the summarization of the example should help to understand the basic principles of EDA.

The scenario starts with a woman who wants to pick up her mother from the airport. She arrives 20 minutes before the planned arrival time at the airport and finds out that the flight has been delayed by an hour. If she had this information earlier, the time could be used for more useful purposes than waiting. What would be the options to avoid situations like this?

1. call the airport or take a look to the webpage before leaving. But there are three weak points. If she forgets to check the arrival time before leaving, in case of a delayed she will still have to wait. Statistics of the US House of Representatives show that 75% of all flights arrive on time [DW05]. So why should someone invest time for a scenario that only occurs in one-fourth of all cases and in any other circumstances he/she will just find out that everything is normal. And third, there is still the possibility that the call was made before the delayed was known.

2. the airline or the airport could offer a notification service, where travelers and their relatives can register for the case of a delay. But everybody has to do it by him or herself, so if he/she forgets, there will be no notification.

3. a third party (in this scenario for instance a travel agency) can register all interested persons for the notification service after booking the flight and takes care for possible changes.

In this example four very important elements of EDA can be identified:

1. events: the flight has been delayed

2. message: the delay is one hour

3. notification service: do something (in this example, inform someone) if an event occurs

4. subscribe mechanism: who should be informed if an event occurs

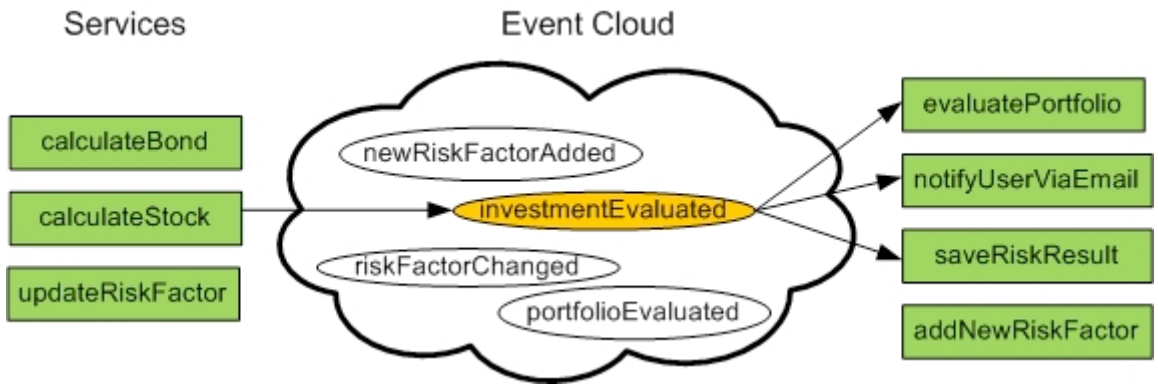

Figure 2.4: Subscribe mechanism EDA

We can extend the previous example; let's assume that someone else in the plane has no chance to get his connection flight now. Normally it would be stressful for individual travelers to find the next available flight. If there are no more flights available today, it's necessary to find a hotel room and maybe rent a car to get there. A software system could support the travelers. The event "*flight has been delayed*" could trigger the procedure "*search next flight*". If there are no flights available, the event "*no more flights available*" will be thrown, which could trigger several other services, for example "*search (or book) a hotel room*". This event triggers another service "*rent a car*", and so on.

An event can be described as "an object that is a record of an activity in a system. The event signifies the activity." [LUC02]. In other words an event represents the state of a procedure, service or system. An event can also include data components; for instance the result of a service. In general the data is transmitted in a message. "A message is an atomic packet of data that can be transmitted on a channel[8]" [HW03]. An essential part of every event-driven system is a transport media for the messages, also called Message Bus. "A Message Bus is a combination of a Canonical Data Model[9], a common command set, and a messaging interface to allow different systems to communicate through a shared set of interfaces" [HW03] There exist several other components in an EDA. A very detailed description can be found in [HW03].

The occurrence of an event in an event-driven system is not predictable. The services are called asynchronous in the most cases. The asynchronous service invocation is easier to realize in an event-driven environment than in SOA. The reason is that the services haven't to wait for the results of any other part of the system. The single elements react only on predefined (subscribed) events. Therefore the Message Bus is one of the most important components in an event-driven system. Every message is transmitted over the Message Bus. In literature you can often find the term "Event Cloud" (see for example [LUC02]) as a metaphor for Message Bus. Every thrown event can be found in this cloud. Most of these events are not relevant for the majority of the services. They can register all events of interest and will get exactly these event-messages transmitted. Figure 2.4 illustrates this process. The system consists of several

[8] "a virtual pipe that connects a sender to a receiver" [HW03]

[9] converts the messages of different systems to a standardized format, which simplifies the data exchange

services (”*calculateStock*”, ”*evaluatePortfolio*”, etc.). The cloud represents the Message Bus, which collects all events in the system. The event ”*riskFactorChanged*” is for instance not relevant for the service ”*notifyUserViaEmail*”, but could trigger the service ”*calculateBond*” and ”*calculateStock*”. Because the change of the underlying risk factor requires the reevaluation of all depending investments. Figure 2.4 shows a similar example. The service ”*calculateStock*” throws the event ”*investmentEvaluated*” after it has finished its calculation. Three services have subscribed this event. So if this event occurs, all subscribed services will get the message, including the result of the calculation, forwarded and can start their procedures (save the result, send the result to the users, evaluate the whole portfolio).

That is the basic principle of EDA. The implementation differs depending on the used messaging system. A more project specific explanation of EDA can be found in Section 3.5 and Chapter 6.

One of the biggest problems of this technology is the error handling. In a dynamic system, it’s very difficult to trace the route of events. The invocation of the single services doesn’t happen always in the same order. It is possible that a workflow delivers the right result in one run, but fails the next time. The reason could be that the services were invoked in a different order. Therefore it’s essential that a powerful debugging and logging framework is available. It has to be considered that conventional test procedures don’t work in the same way like in an object-oriented framework and have to be modified for an event-driven system.

3 Design and architecture of a flexible risk measurement system

3.1 Basic idea - requirements

In this section the functionality and goals of the designed risk-measurement system will be defined. The following properties should be taken into account at the design and implementation phase of this application:

- expandability
- scaling
- easy administration
- reliability
- traceability
- reusability
- implementation of open standards

As already mentioned in the previous chapter the main elements of a service-oriented or event-oriented architecture, respectively, are business processes. Therefore the first task will be to identify these processes and to model the relationships between them. The first assumption is that there don't exist any legacy systems, so it's possible to design the system top-down without considering any existing modules. There is one exception; the major goal of the designed system is to evaluate the risk of investments and whole portfolios. So it would not be reasonable to include a powerful portfolio- and investment-management component (it would also lead to an additional administration work and redundant datasets). Furthermore this kind of system should be separated and the interaction is handled over predefined event-based interfaces.

All elements of the application should be implemented as services to improve the reusability and scalability. For a better organization all services are grouped into packages. For instance the package risk-evaluation includes all services, calculating the risk of an investment product or portfolio. But it doesn't mean that all services have to be provided at one server, the packages show only a logical correlation and help to administrate the system.

The orchestration of all involved services is managed with help of the event-driven architecture approach. The management of the events, which also includes invoking the web services

and receiving and delegating the results, is handled by a central component called ESB (Enterprise Service Bus). The architecture and processes will be discussed in detail in the next section. But to illustrate the basic principles we want to take a closer look to the major functionality of the application. The main task is to evaluate the market-risk of a wide range of investment-products. Due to the usage of web services it's possible to implement the evaluation logic in different programming languages and on various platforms. In chapter 5, one module is presented which calculates the value-at-risk of stock options with help of the Monte-Carlo simulation approach (see also 4.3.2), whereas the complete simulation and evaluation process is implemented with Mathworks MatLab. The usage of already existing functions allows a very fast development. The objective of this system is to base on an iterative software development process, so that a lot of basic financial instruments already exist in the first version. The used models can be improved anytime. But this can also be a problem comparing historical results, which are calculated with different evaluation approaches. Therefore the application has to be designed to handle these kinds of inconsistencies.

A very important part is the management of the risk factors including their historical time series. One package is created which includes all interfaces to the common data providers (e.g. Bloomberg, Thomson, Reuters, etc.) and manages the master data of the various risk factors too. All datasets will be stored in a shared database, whereas the access is covered over an individual web service. This allows the use of loose-coupled services. For example one process updates all risk factors daily at 11 p.m., with the following workflow: first, the time-scheduler throws an event at 11 p.m. "update risk factors". This event leads to several service invocations. The first service loads all available risk factors in the database; for every dataset a new event is thrown called "update single risk factor". These events activate another service, which downloads the price time series from the corresponding data provider. If the prices are downloaded another service will be activated, storing the data into the database. The services know nothing about each other. The whole orchestration of events and invocations is handled by the ESB. But this allows a maximum flexibility extending the application. For instance, the quality of the risk factor data is very important. Wrong values or outlier in data would lead to incorrect evaluation results. Therefore, maybe it's expedient to check all downloaded data (with help of statistical functions) to avoid this kind of problems. Such extensions are quite simple to realize in this system. The event "data download finished" will not only lead to an invocation of the database store service, furthermore it will also call a controlling service which checks the new data. If there are any inconsistencies, a new event will be fired which can lead to a notification of the responsible persons by email, text message, etc. and/or to a generation of an error report.

While this system is very easy to extend and administrate, it's not so trivial to understand and control the processes. All service invocations occur asynchronous. In case of an error it's very difficult to find the state where the failure happens. There is also the possibility that other events influence the input parameter of a service. Therefore it's necessary to find the right level of abstraction and write a good documentation of all influence processes and events. Hohpe quotes Fowler: EDA: "the architect's dream, the developer's nightmare" [HOH06]. This scenario has to be avoided. Another objective of the application design is to modeling the events and service invocations as universal as possible. But that won't be possible at all times. For example a rule is specified, which calculates all depending investments after

updating a risk factor. This implicates, if an investment depends on ten risk factors, after the daily update the investment will be recalculated ten times. Furthermore, if the calculation of an investment leads to an update of the portfolio evaluation, a portfolio with 100 assets would be recalculated 1000 times, instead of only one needed validation. A better way to improve the runtime behavior will be to use group as well as single events. Daily routines will be combined to one event, so only one iteration is needed. Another situation is given if a "non-routine" event happened, for example if the nominal amount of an investment has changed, the risk evaluations of the investment and the depending portfolio have to be updated.

One of the most complex tasks is the data storage. As already mentioned before, the investment and portfolio administration is located outside of this system. But it has to be considered that the results of the evaluation can be assigned to the right investment. Anyway, a change of the underlying management system shouldn't lead to a high administration work. Therefore all relevant investment data are synchronized with the own system-database. But what happens if a new type of investment (with new parameters) is added to the system? It's not possible to cover all possible parameters already at the design phase. The aim is to create an open database, where additional arguments can be added very quickly and safely. The database access component (web service) should be implemented with help of an object-relational mapping software (e.g. Hibernate[1]), so that the code fragments are generated automatically. This topic will be discussed in detail in chapter 5.

The risk evaluation system has no specifications about the used user-interface technology. Due to the service based business logic, it's possible to implement standard windows-applications as well as web applications, mobile applications, etc. For the standard functionality there is no user interaction required. So it would be possible to send all results in kind of risk reports via email. But in many cases it's more interesting to build a whole information system. In the prototype all interfaces will be implemented as web application.

In the next section the architecture of the system is described in detail. After that, all relevant processes should be identified and their relationship will be shown in a graphical way.

3.2 Architecture

Figure 3.1 shows all basic elements of the system. The EDA-component is used as gateway and orchestration module. All functionalities of the system are grouped into nine independent packages. Each of them contains a number of services, which fit thematically to the topic and can be extended anytime. Also different versions or implementations of one service can be grouped in one of these containers. The following itemization shows all components and their covered topics.

- risk-evaluation: in this module all services to evaluate the risk of an investment are grouped. It's also possible to implement different risk evaluation approaches for one financial instrument. In chapter 5 one of these cases can be found, where the evaluation of a stock option is implemented with help of the delta-gamma approach and the Monte-Carlo simulation approach. Due to the event-based control of the system, it's very easy

[1] http://www.hibernate.org

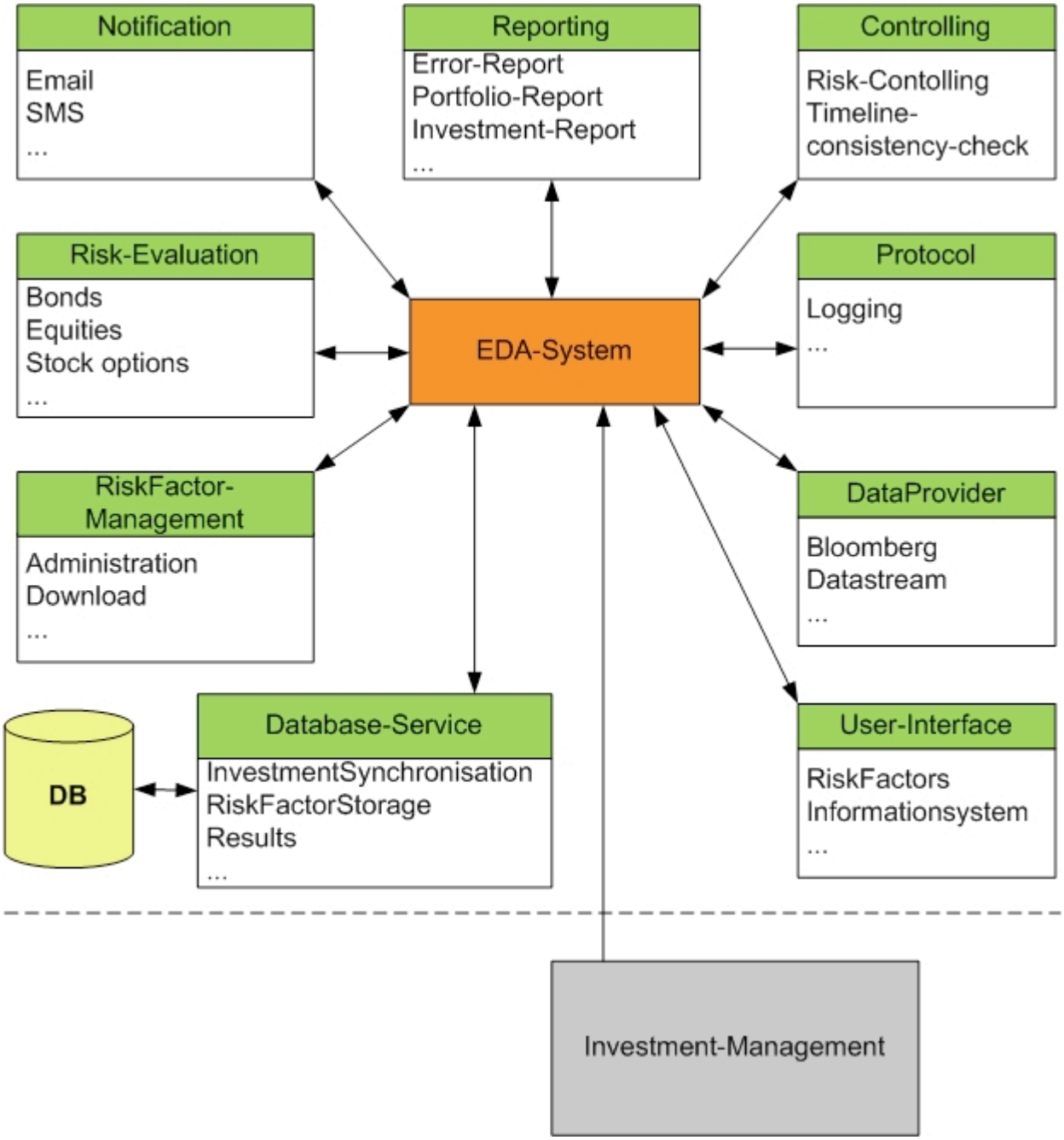

Figure 3.1: Core architecture of the risk measurement system

to calculate both approaches at the same time and compare the different results. In this category the evaluation of the whole portfolio can be found, too. Therefore it has to be considered, that if an investment is valuated with different methods, it should be only calculated once in the portfolio.

- risk factor-management: the risk factor timelines are one of the most important parts of the risk evaluation. This module provides all needed functions to administrate the risk factors, for example the methods to update a time series or to export data in other formats (e.g. XML, CSV, MS Excel, etc.).

- data provider: the historical time series of the risk factors have to be downloaded from a data provider. This package includes all needed interfaces to the most common data provider services, for instance: Bloomberg, Thomson Datafeeds[2], Reuters. Another possibility is to parse data from existing web pages like Yahoo finance[3] or Google finance[4]. The following input parameters are required for every service: ticker of the time series (or any other unique identification, e.g. ISIN), data type of the time series (price data, dividend yields, etc.), time period (daily, weekly, mothly data, ...).

- controlling: this module bundles all controlling mechanisms in the system. This covers different areas. One function could be to test the downloaded risk factor time series for wrong or unordinary data points. Another scenario is to check the evaluated investments for exceeded ranges, e.g. a rule in the enterprise lays down that the daily value-at-risk of a portfolio has to be less than 8%.

- notification: the notification package allows to send information and attachments over different channels. These could be email, SMS, ICQ, MSN, Skype, etc. One application would be to send system error notifications via text messaging to the responsible administrator, or distribute risk reports with help of the email interface.

- reporting: a very important application is to generate reports to summarize all results in a compact format. These reports can have different contents and it should be possible to customize them in a simple way.

- protocol: as already mentioned before, due to the event-based behavior, it's not easy to reconstruct all executed processes. That's one of the reasons why it's maybe more important to implement an extensive protocol system than in other applications. The goal is to log all events in a special database. One of the extensions could be to view all depending processes (including their runtime, parent/child processes, etc.) in a graphical way.

- database-service: The package contains all functions to access the database of the risk measurement system. This includes the result management as well as the synchronized investment and portfolio administration database and the risk factor storage. The single

[2] http://www.thomsonreuters.com/products_services/financial/datafeeds
[3] http://finance.yahoo.com
[4] http://finance.google.com

services have to be customized to the specific database and therefore for the whole system. So it's not possible to implement loosely-coupled services in this case.

- user interface: the module doesn't represent fundamental components of the risk measurement system, but it groups all parts of the user interface. This could be, for instance, an information system where all results can be explored or some applications to administrate and control the whole system.

3.3 Workflow Management

The basic elements of an event-driven system are independent processes. One very important task is to identify the single processes and choose the right granularity. As already mentioned before, this system is completely new designed. This means, no legacy-systems have to be taken into account. Therefore the top-down approach to find the processes can be used. This means starting at the basic functionality of the system, the level of abstraction gets more and more detailed. For instance, one part of the system updates the risk factors, which affects the following processes: get all available risk factor in the system (process: getAllRiskFactors). For all of these risk factors download the time series from a data provider (downlDataBloomberg), etc.

3.3.1 Process description

In the following pages the most important processes of the system are shown in a tabular form (Table 3.1), where the name of the process, the description and the thrown events will be described. This list represents only the basic functionality of the system, so it would be no problem to add new processes and workflows to improve the application (see also Chapter 3.5).

Name	Description	Event
Risk factor management		
addRiskFactor	a new risk factor is created	RFMcreateRF
chgRiskFactor	an existing risk factor should be changed	RFMchangeRF
delRiskFactor	a risk factor should be deleted	RFMrmRF
regularUpdate	all risk factor timelines are updated (e.g. as part of the daily routine)	RFMregUpdate
singleUpdate	only one single risk factor is updated (e.g. if a new risk factor was added)	RFMsglUpdate
chgTimelineItem	an user initialized update of a time series entry	RFMchgTiml
downloadTimeSeriesCVS	delivers a CSV-File including the time serie of a specific risk factor	RFMdownlCSV
Data provider		
downlDataBloomberg	downloads a time series from Bloomberg	DPdownlBl

downlDataDatastream	downloads a time series from Thomson Datastream	DPdownlDS
Risk evaluation		
evaluateStockDelta	evaluates the risk of a stock with help of the delta approach	REstockDelta
evaluateStockSim	evaluates VaR of a stock with the Monte-Carlo simulation approach	REstockSim
evaluateBondDelta	evaluates a bond with the delta approach	REbondDelta
evaluateStockOptionDelta	evaluates the risk of a stock option with help of the delta-gamma approach	REsoDelta
evaluateStockOptionSim	evaluates a stock option with the Monte-Carlo simulation approach	REsoSim
evaluatePortfolioDelta	a whole portfolio is evaluated with the delta-gamma approach	REportDelta
evaluatePortfolioSim	a whole portfolio is evaluated with the Monte-Carlo simulation approach	REportSim
calculateAllInvestments	calculates all active investments as part of a routine	RECalcAllInv
calculateSingleInvestment	calculates one specific investment (e.g. after a new transaction)	RECalcInv
calculateAllPortfolios	calculates the risk of all portfolios in the system	RECalcAllPort
Database services		
buyInvestment	a new investment (bond, stock option, ...) is added to the system	DSbuyInv
sellInvestment	an investment (or part of it) was sold	DSsellInv
addPortfolio	a new portfolio is created	DSaddPort
delPortfolio	an existing portfolio will be removed	DSdelPort
getInvestmentData	delivers details of an investment (nominal value, etc.)	DSgetInv
getAllPortfolios	returns all existing portfolios	DSgetAPort
getAllInvestments	returns all active investments	DSgetAInv
addRiskFactor	adds a new risk factor to the database	DSaddRf
addRiskFactorItem	stores one data set entry	DSaddRFI
getAllRiskFactors	delivers all available risk factors	DSgetARFs
getRiskFactorTimeSeries	delivers the time series of a risk factor	DSgetRFTs
saveResultInv	saves the risk evaluation result of an investment	DSsavRInv
saveResultPort	saves the risk evaluation result of a portfolio	DSsavRPort
getAllResults	delivers all risk evaluation results	DSgetARes

getResult	delivers the detailed result of an investment	DSgetRes
Controlling		
checkTimlineCons	controls if there are any inconsistencies in a time series	ckTimeCons
checkRiskRange	controls if the calculated VaR is higher than a predefined range	ckRiskRange
Reporting		
createPortfolioReport	creates a report which summarizes the calculated results	RcrPortRep
createErrorReport	creates an error report (e.g. with all problems of a daily update)	RcrErrRep
createInvReport	creates a report which shows a specific investment	RcrInvRep
Notification		
sendEmail	allows to send a message via email	NsendEmail
sendSMS	sends a text message via SMS (e.g. if a service break down the administrator will be informed)	NsendSMS
Protocol		
logEvent	logs all thrown events	PlogEv

Table 3.1: Process description

3.3.2 Workflow scenarios

After describing the single processes of the system, we want to go one step further and coordinate these processes to perform several tasks. The invocations of the services are completely event-based. Therefore three kinds of events have to be distinguished:

1. Events initialized by users (e.g. manual changes of a time series)
2. calendar time based events (e.g. regular routines, the update should be started every day at 11 p.m.)
3. system based events (e.g. a calculation has finished)

In the following illustrations, two scenarios should be described. First, the daily update routine (see Figure 3.2): the goal of this workflow is to calculate the risk of all investments and portfolios once a day and generate a risk report which summarizes the results. The initializing event is calendar time based and will be, for instance, thrown every day at 11 p.m. At the beginning, the risk factor time series will be updated (download all time series from the data provider Bloomberg, save the entries to the database and check if there are any inconsistencies

in the data). After that the risk of all investments for the actual evaluation date will be evaluated. Then the portfolio evaluation will be calculated, followed by generating the risk report, which will be send to all responsible persons.

The second scenario shows the workflow after adding a new investment (see Figure 3.3). The initializing event in this case is system based. After inserting a new dataset into the investment management database, an event is thrown which will be fetched from the EDA-system. The investment will be added to the risk database, and all necessary risk evaluations are calculated. The predefined risk rules are controlled for each evaluation result. If there are any violations the controlling department will be notified.

3.4 Database

The database design is a very important part of the whole architecture process. Due to the flexible property of the system, it's necessary to design the database in a way, which allows a fast and efficient expandability. Figure 3.4 shows the core architecture of the database. In general there exist three kinds of fields:

- risk factor management
- investment and portfolio administration
- management of the results

It's possible to distinguish between different kinds of risk factors, for instance: exchange rates, interest rates, price data of indices, etc. All risk factors have to be bound to a data provider where the time series can be obtained (e.g. Bloomberg, Reuters, or also any other web source). These data are stored in an own database table called "RiskFactorTimeline". The next block of functionality is the portfolio administration. As already mentioned before, the portfolios and investments are managed outside of the risk evaluation system. But it's necessary to allocate the results and the related risk factors to the investments. Therefore all needed investment data are stored in the database. Updates are handled event-based, for instance if a change occurs in the portfolio-management system an event is thrown. This event is caught from the risk evaluation system and the changes are transacted in this database too. The main table is called "Investment". All types of investments are deduced from this table. This approach allows to extend the system in a very simple way. For example if an index option, as a new type of investment, was implemented, just a new table named "IndexOption" has to be attached to the table "Derivative". All other existing entries will remain unaffected. The "Investment"-table and all of its children include only the master data fields. The specific transactions are stored in the table "Transaction". For all active transactions the value-at-risk is calculated in a predefined interval. The big advantage of this system is the possibility of using different approaches to evaluate the risk at the same time, which allows an easy comparison. But therefore the results have to be marked with the used approach. Another possible scenario is the existence of different versions of an evaluated risk result. This can be occurred if one entry in a risk factor time series has changed or a new investment is added to the portfolio.

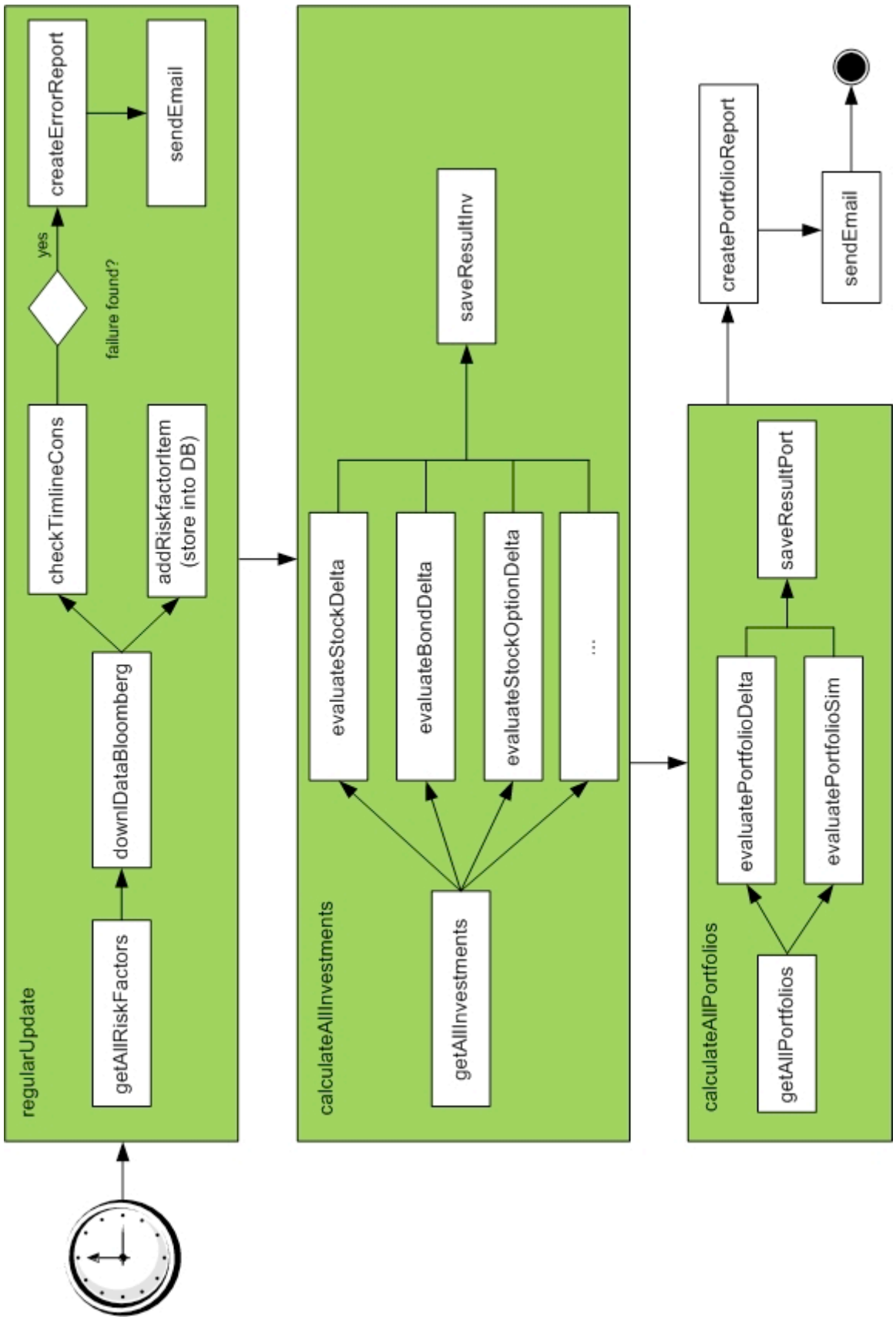

Figure 3.2: Workflow - daily update routine

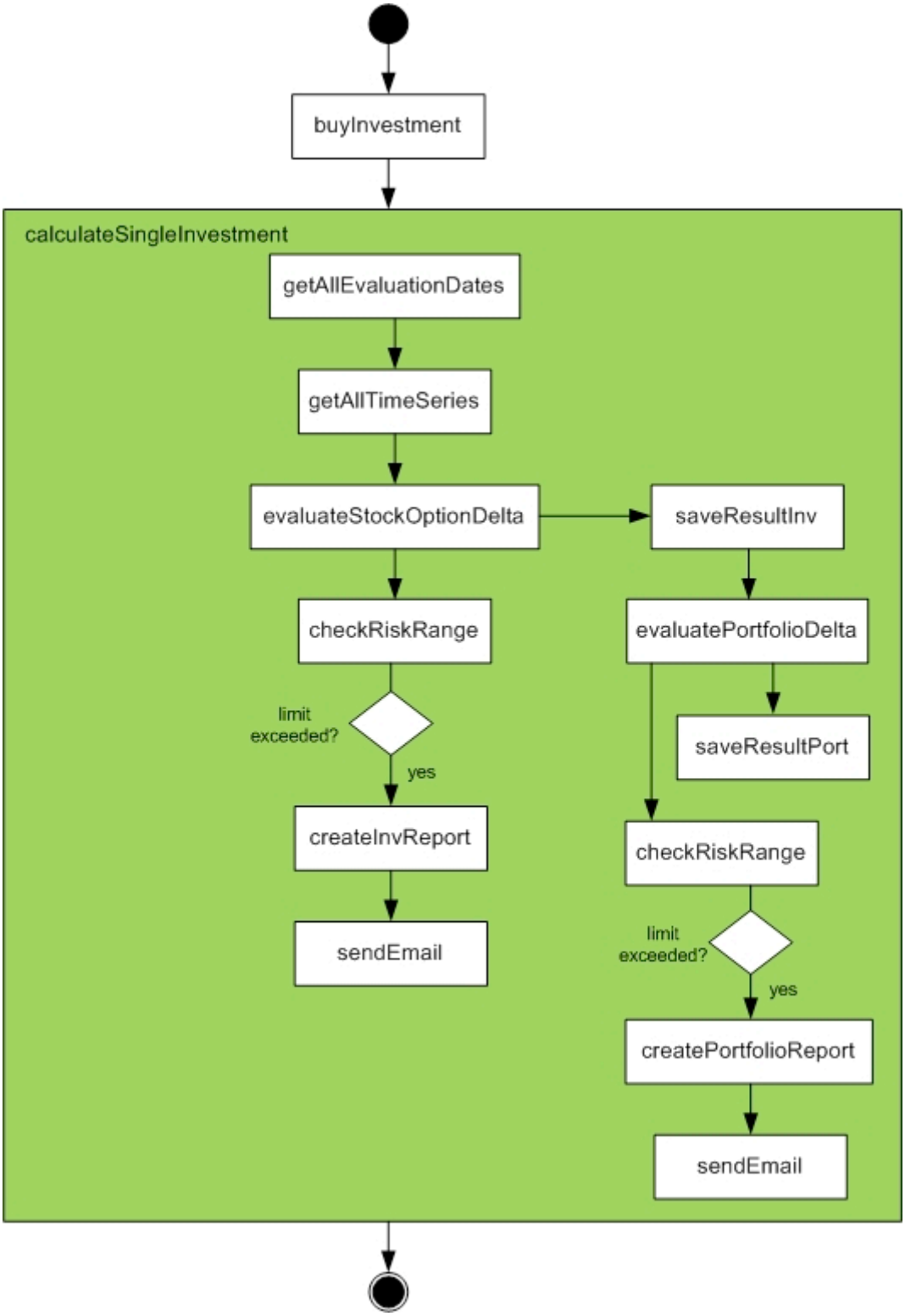

Figure 3.3: Workflow - new investment added

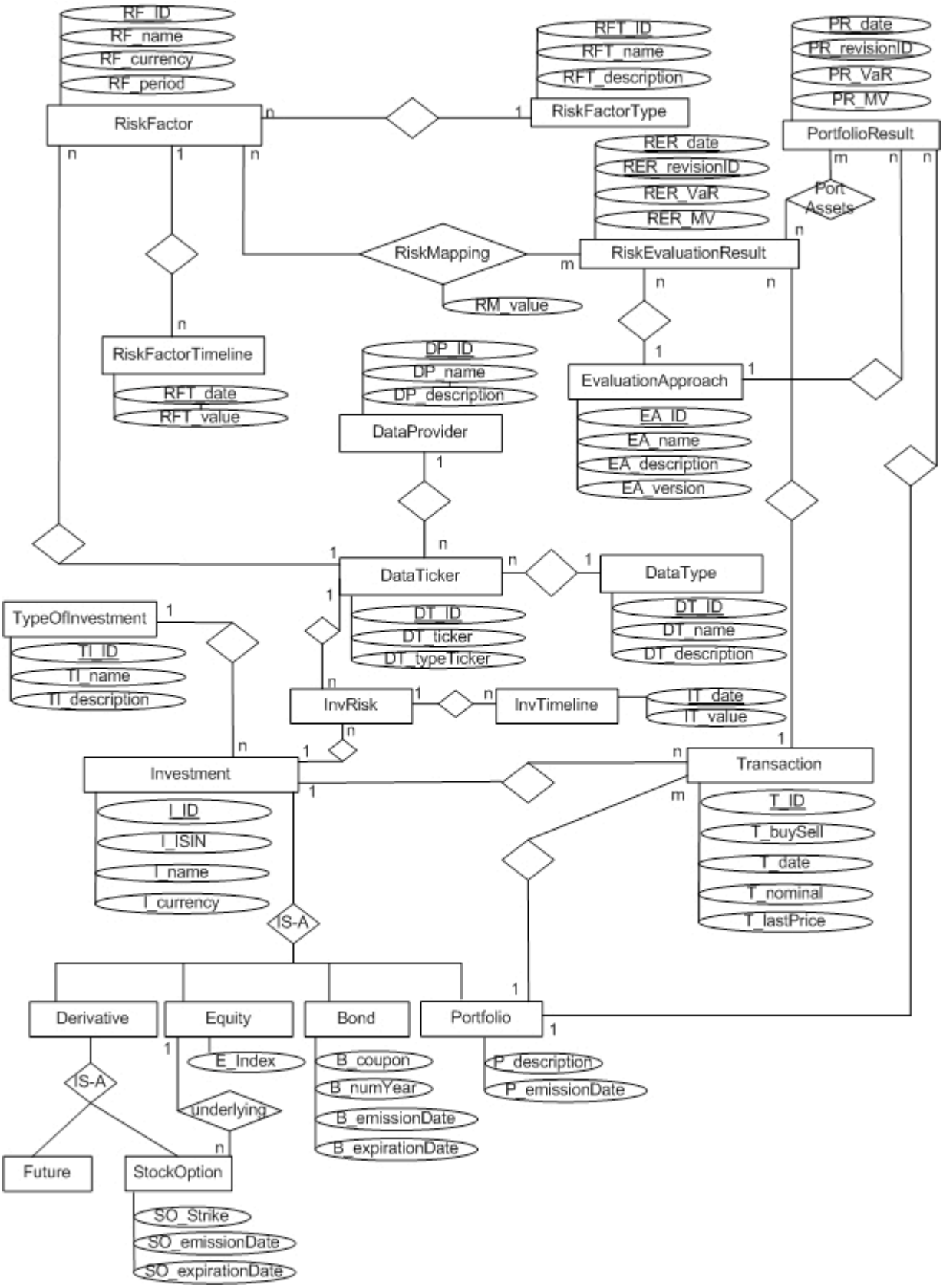

Figure 3.4: Database design

In this cases the old results shouldn't be overwritten, furthermore a versioning system (in addition to the log-system) is established. For further statistics the weights on the related risk factors for each investment and portfolio are also a very important information to check the risk allocation of the portfolios. This information is stored in table "RiskMapping"

3.5 Expandability

The goal of this project is the design and prototyping of a flexible value-at-risk system. One of the most important aspects of the term flexibility is a fast and simple expandability of the existing system. In practice there are several fields of possible extensions. The most common scenario is the implementation of a new kind of investment. For instance a new product was added to the portfolio, which should be evaluated. Another possibility could be found in the change of the used data provider to download the risk factor time series, or just improve the existing evaluation approaches.

The solution to handle this kind of problems, builds on several aspects. The first critical component is the database. Most changes can be found at the investment administration, therefore all relevant sub-categories are derived from the table "Investment". This allows a fast adding of new financial instruments because the main attributes are always the same and new tables include only the new required fields. But it's not enough that only the database can be changed quickly, furthermore the whole database access components have to be adopted in a reasonable time. For this O/R (object-relational) mapping frameworks like Hibernate see Chapter 5.2 provide a very elegant way to extend the data-access layer.

The most powerful tool to guarantee flexible applications is the use of services and the event-based paradigm. As already mentioned in the previous chapter, a service acts completely autonomous, therefore a new-implemented service doesn't affect any existing modules. This behavior allows an exchange of different versions of a module on the fly. Every service has its own URL (unique resource locator). A new implemented version of a module can be integrated in the system just changing the address in the configuration file.

Next I want to show one possible procedure to extend an existing workflow with the help of services and events. The goal is to extend the workflow of Figure 3.3 (a new investment is added) with the following scenario: in the current version the risk of a stock option is calculated with help of the delta-gamma approach (see Chapter 4.3.1). The long-term objective is to increase the precision of the evaluation results. Therefore the option should be calculated with the Monte-Carlo simulation approach additionally. First, the service with the new calculation method has to be implemented as web service and published on a web server. The integration of the new service in the existing workflow is very simple. After getting all needed risk factor time series from the system, a new event is thrown which activates the process "evaluateStock-OptionDelta". The new process "evaluateStockOptionSim" has just to subscribe this thrown event. This leads to a parallel calculation of both approaches and an easy way to compare the different methods. The new workflow is shown in Figure 3.5.

The subscribe mechanism allows a really quick extension of functionality. For example if a risk report should be generated after calculating the stock option with the Monte-Carlo simulation approach, the process "createInvReport" has only to subscribe the event "REsoSim".

There are no additional changes required in the code.

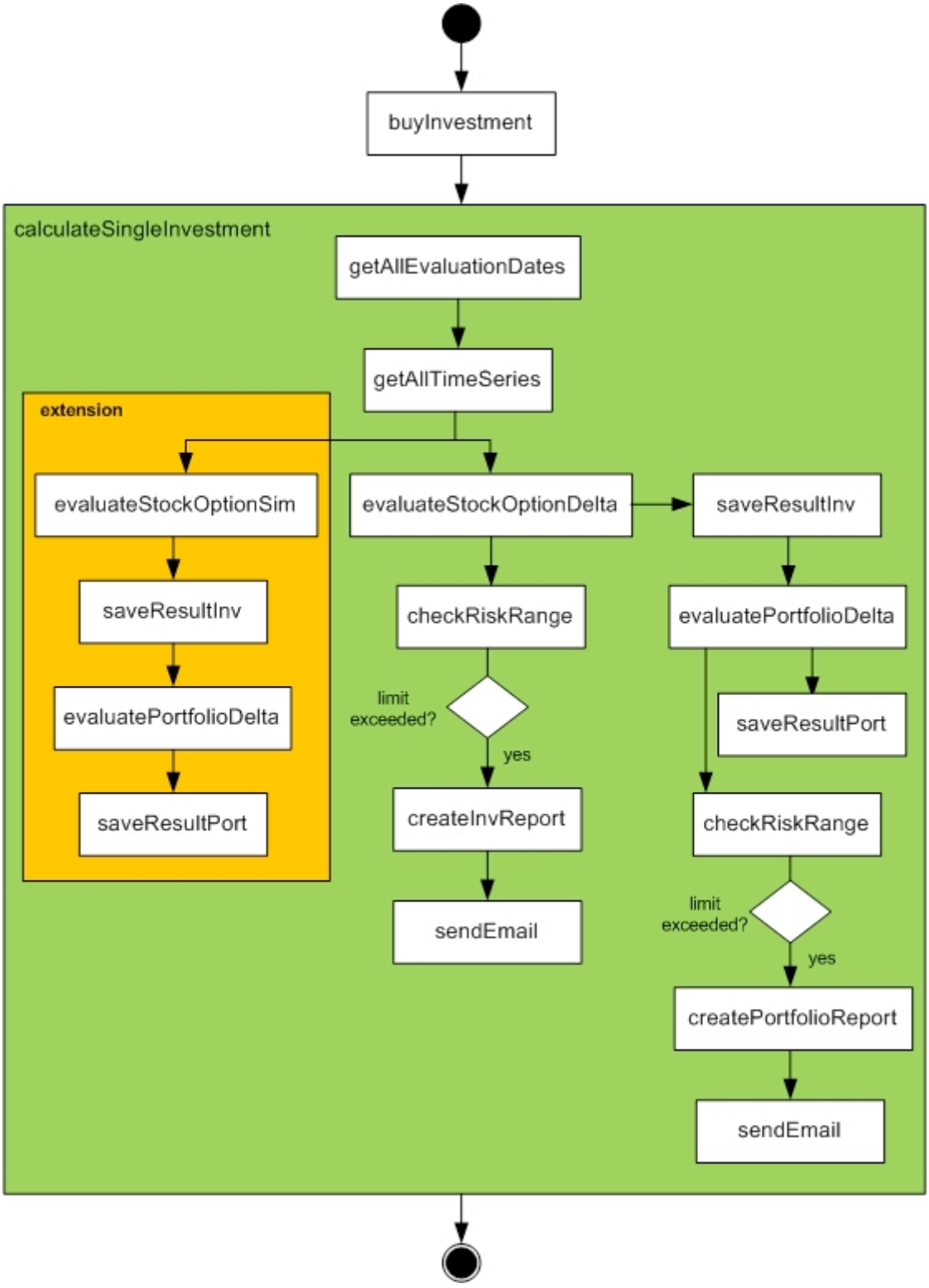

Figure 3.5: Expandability - new evaluation approach added

4 Pricing and risk measurement of stock options

4.1 Value-at-risk

In financial institutions risk measurement got more and more important. Jorion [JOR97] shows in several case-studies the effects of neglected risk regulation. One of the most impressive examples is the bankrupt of the Barings bank in 1995. A single futures trader lost more than 1.3 billion US-dollars in just two month, with an investment on the Japanese stock index Nikkei 225. The incredible sum of 7 billion US-dollars was allocated on only one risk factor (for comparison, the total profit of Barings bank was approximately $100 million in 1994).

On the other hand investment products get more and more complex. Especially for "structured products" it's not easy, or in many cases impossible, to see intuitively all affected risk factors. The situation gets even worse in a portfolio with many of these investments, where correlation effects have to be considered. That's one of the reasons, why such an approach is needed, which allows a simple but powerful risk evaluation on various investment products. A very popular approach today, is the value-at-risk (VAR) calculation, presented by JP Morgan[1] in 1994. Jorion gives the following definition: "VAR summarizes the expected maximum loss (or worst loss) over a target horizon within a given confidence interval" [JOR97]. In general there exist three kinds of VAR calculations:

- analytic method
- historical method
- simulation method

but all of them have the same idea, based on the expected return-distribution of the investment. A simple example should demonstrate the basic principals of VAR calculation, evaluating the risk of an equity investment. For that, the following assumptions are made: The last price of a stock is 100€ and the expected daily returns are normally distributed with a mean value of 0 and a standard deviation (volatility) of 1.5%. At value-at-risk especially the left-tail of the distribution is of interest to see the potential loss. Now the following question can be answered: "What is the maximum daily loss of the equity investment with a 99% probability?" In this example it is 3.49€ per share as shown in Figure 4.1.

Next, I want to discuss different ways to find the distribution of asset returns. A common example is used to show the differences between the various approaches. The objective is to

[1] Riskmetrics [JPM96]

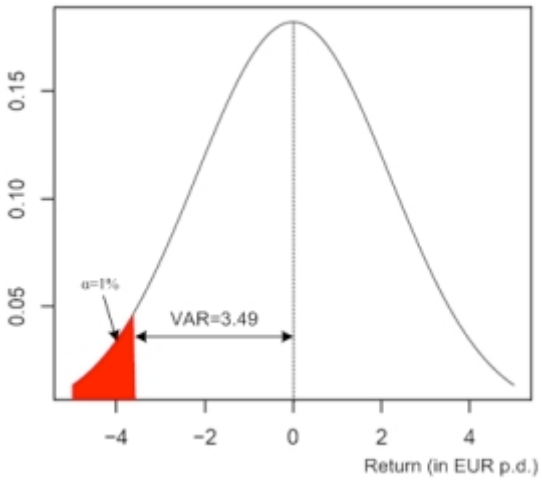

Figure 4.1: 99% VAR for an equity investment with $\mu = 0$ and $\sigma = 0.015$

evaluate value-at-risk of a DAX index listed BMW[2] share for one trading day with a 95% confidence level. The evaluation date is August 1st, 2007, for that, the calculation is based on historical prices[3] of the last three years (August 2nd, 2004 - August 1st, 2007). The last available price on August 1st 2007 is 43.8 €.

4.1.1 Analytic approach

The analytic approach can be divided into two categories, the delta-normal method and the delta-gamma approximation. The delta-gamma approximation is an extension of the delta-normal method, which allows the risk evaluation of non-linear investments. In this chapter only the linear investments will be discussed, the VAR calculation of stock options (as representative of non-linear investments) is presented in Chapter 4.3. The analytic approach has the assumption that the returns of the underlying are normally distributed. Three factors are necessary to measure the market risk of a stock investment, the mean μ, the volatility σ and the last price S_t of the risk factor. With these parameters VAR can be calculated as follows:

$$VAR(\alpha) = -S_t(\mu + \sigma\Phi(\alpha))$$

The operation is quite simple, $\Phi(\alpha)$ represented the α quantile of a standard normal distribution. The standard normal distribution is, as known, a special case of the normal distribution with $\mu = 0$ and $\sigma^2 = 1$. Therefore with the given mean value and volatility of the risk factor the quantile of the corresponding normal distribution can be calculated, which is equivalent to the maximum lost return with a probability of $1 - \alpha\%$. After multiply this value by the last price of the underlying, the maximum loss per share is the result. To simplify the calculation, in literature the drift factor μ is sometimes set to 0 (for example in JP Morgan's Riskmetrics [JPM96]).

[2]ISIN: DE0005190003
[3]Thomson Datastream Symbol: D:BMW data type: P

Next, the required parameters have to be set. There are different ways to estimate the volatility of asset returns. On the one hand historical data can be used to calculate the volatility, or on the other hand, it's possible to use derivative instruments to deduce the standard deviation. For instance if the underlying is a stock investment, the implied volatility could be calculated with help of the Black-Scholes formula (see Chapter 4.2) by a given option price. Jorion writes: "Historical data, however, may not provide the best available forecasts of future risks. Situations involving changes in regimes, for instance, are simply not reflected in recent historical data. This is why it is useful to turn to implied forecasts contained in the latest market data." [JOR97] The question which kind of volatility estimation should be chosen will be a discussed topic in the next chapter, pricing a stock option. However, there are many situations where the use of historical time series is inevitable. As mentioned before, for calculating VAR not the price of the underlying is relevant, but rather the returns of the underlying prices. To calculate the return of a price time series, two methods can be used, the arithmetic (also called discrete) rate of return and the geometric rate of return.

$$rt_{t\_arit} = \frac{P_t + D_t + P_{t-1}}{P_{t-1}} \qquad rt_{t\_geom} = ln\Big(\frac{P_t + D_t}{P_{t-1}}\Big)$$

Whereas P is the price of the asset at time point t and D is a dividend payment. Which method should be chosen depends on the time series. For example there exist differences between long term and short term time series or also on kind of investment. Jorion describes two advantages of using geometric returns. First, due to the normal distribution of the returns, it's not possible to reach a negative price of the underlying. Second, the returns of foreign exchange rates are more consistent. For example the return of two FX(EUR_USD) prices is $rt = ln\Big(\frac{P_t}{P_{t-1}}\Big)$. If we take a look on the opposite view FX(USD_EUR) it's $-rt = ln\Big(\frac{\frac{1}{P_t}}{\frac{1}{P_{t-1}}}\Big)$. This reverse isn't possible with the arithmetic approach.

The next step is to evaluate VAR for the BMW share. At first the return series will be calculated with the geometric approach. The mean value of the daily returns is 2.189E-4, the volatility (with equally weighted returns[4] of the last three years) is 0.012. With this two moments a normal distribution can be generated (see Figure 4.2). After calculating the 95% quantile of this distribution, a return value of -0.01946 is the result. The value-at-risk is 0.85€ (multiplying the 95% quantile of the distribution by the last asset-price at evaluation date (in this example 43.8€)). Which means, with a probability of 95% I cannot lose more than 0.85€ per share within one trading day. This example shows the simplest way to evaluate VAR with only one risk factor, the underlying itself. A little bit more complex is the calculation of the market risk of an investment with more than one risk factor (for instance if an equity noted in none Euro markets, the foreign exchange risk has to be considered) or the evaluation of the risk for a whole portfolio. If not all risk factors are perfectly correlated, the diversification effect has to be taken into account. A very simple method to calculate VAR of the portfolio can be done with the following formula:

$$VAR(\alpha) = \sqrt{\Delta^T \cdot \Sigma \cdot \Delta} \cdot \Phi(\alpha) \cdot \sqrt{T}$$

[4] $\sigma = \sqrt{\frac{1}{(T-1)} \sum_{i=1}^{T} (rt_i - \hat{\mu})^2}$

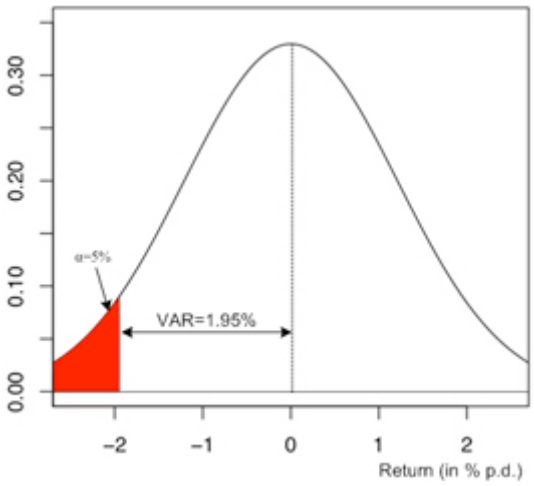

Figure 4.2: 95% VAR for one day of a BMW share on August 1st, 2007

Δ ... is a vector of all delta equivalents $\delta_1 \dots \delta_n$
δ ... symbolize the exposure of a risk factor. If there are more than one delta equivalent per risk factor the values add up
Σ ... is the variance-covariance matrix for all corresponding delta equivalents
$\Phi(\alpha)$ ... the α quantile of a standard normal distribution
T ... is the target horizon

Example: A portfolio (quoted in Euros) consists of two investments:

1. 10,000 General Motors shares[5]
2. an USD zero-bond with a duration of 2 years and a nominal amount of 100,000 USD

The objective is to calculate the 95% VAR for a target horizon of 1 day (what is the maximum loss of the portfolio over one trading day with a given probability of 95%), date of evaluation is August 1st 2007. All historical time series based on daily data from 2/8/2004 - 1/8/2007.
Last price GM share on 08/01/2007[6]: 32.73 USD
Discount factor USD2Y on 08/01/2007[7]: 0.9066
Foreign exchange rate EUR/USD on 08/01/2007[8]: 1.3668

For the GM share risk evaluation, two risk factors are necessary: the underlying itself and the foreign exchange risk.
$\delta_{GM} = \frac{GM_0}{FX(EUR/USD)_0} * n = \frac{32.73}{1.3661} * 10,000 = 239,587.15€$
$\delta_{EUR\_USD} = \delta_{GM} = 239,587.15€$
where n is the number of shares. The bond depends also on two risk factors, the USD2Y

[5]ISIN: US3704421052
[6]Thomson Datastream symbol: U:GM data type: P
[7]Thomson Datastream symbol: US02Y00 data type: P
[8]Thomson Datastream symbol: EMUSDSP data type: P

	GM	USD2Y	FX(EUR/USD)
GM	5.5082E-04	1.1170E-06	-3.2297E-07
USD2Y	-1.1170E-06	6.2481E-07	4.8205E-07
FX(EUR/USD)	-3.2297E-07	4.8205E-07	4.8957E-03

Table 4.1: Variance-covariance matrix

interest rate and also the foreign exchange rate.
$\delta_{USD2Y} = \frac{USD2Y_0}{FX(EUR/USD)_0} * n = \frac{0.9066}{1.3661} * 100,000 = 66364.10€$
$\delta_{EUR\_USD} = \delta_{USD2Y} = 66,364.10€$
$\delta_{EUR\_USDGes} = 239,587.15 + 66,364.10 = 305,951.25€$
Now the VAR of the portfolio can be evaluated with the following calculation:

$$VAR(95\%)=\sqrt{\begin{pmatrix}239587.15 & 66364.10 & 305951.25\end{pmatrix}\begin{pmatrix}5.51e^{-4} & -1.12e^{-6} & -3.23e^{-7}\\ -1.12e^{-6} & 6.25e^{-7} & 4.82e^{-7}\\ -3.23e^{-7} & 4.82e^{-7} & 4.90e^{-3}\end{pmatrix}\begin{pmatrix}239587.15\\ 66364.10\\ 305951.25\end{pmatrix}}\cdot\Phi(0.95)=-36403{,}94$$

4.1.2 Historical simulation approach

Another possibility to calculate value-at-risk is the historical simulation approach. Instead of using the normal distribution assumption for the asset returns, a different way is chosen based on the historical timeline of a risk factor. The basic idea is that the return distribution in history will be "similar" to the return distribution in the future. At first all relevant risk factors have to be identified. The price-data time series for these risk factors have to be loaded for a reasonable time period (for instance for the last three years). The problem, choosing the right time interval will be discussed in Chapter 4.1.4. After that, the returns (in this case using the arithmetic rate of return) of all available data points are calculated and a potential market value change distribution can be generated ($S_0 \ldots$ market value of the underlying at the date of evaluation, $S_n = S_0 * (1 + rt_n) - S_0$)). The α quantile of the distribution is the value-at-risk.

Let's calculate the market risk for the BMW example with the historical simulation approach. The used historical BMW price time series is taken from August 2nd, 2004 - July 31st, 2007) which are 782 data points. The calculation in extracts is shown in Table 4.2. The VAR for the 95% quantile is 0.87€ per share.

There is also a very easy way to calculate the market risk for different time horizons. For daily VAR, the historical time series should also be based on daily data. If the VAR for ten trading days is required, the returns for every two weeks have to be chosen, etc.

One big advantage of the historical simulation approach is that the modelling of correlations between two or more risk factors is very simple. The following assumption is made: all combinations of price changes in history for one time point are also taken in this combination in future. If we take another look to the General Motors example, on July 19th 2007 the price for one GM share was 35.38$ and the exchange rate EUR/USD was 1.3814; on the next day the prices were 34.92$ and 1.3788. Therefore the returns are -0.0131 for the GM share and -0.0019 for the exchange rate, on July 20th 2007. One value in the expected market value change distribution for VAR on August 1st 2007 (with the base values 32.73$ and 1.3661) is

			S_0 = **43.80**
Date	**Price**	**Return**	**expected market price change**
08/02/2004	36.9		
08/03/2004	36.9	0.000	0.00
08/04/2004	35.65	-0.034	-1.51
08/05/2004	35.01	-0.018	-0.79
⋮	⋮	⋮	⋮
07/26/2007	44.64	-0.035	-1.55
07/27/2007	45.06	0.009	0.41
07/30/2007	44.96	-0.002	-0.10
07/31/2007	45.94	0.022	0,94

Table 4.2: Example Historical Simulation - BMW

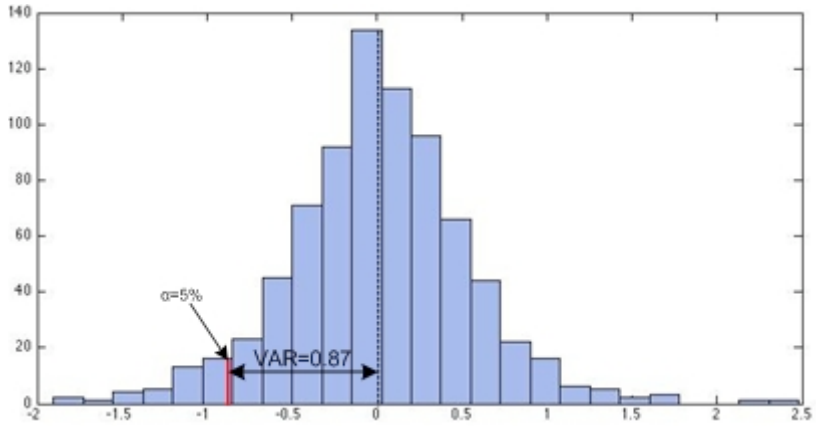

Figure 4.3: Example Historical Simulation - BMW - Histogram of the expected value changes

$\frac{32.73(1+(-0.0131))}{1.3661(1+(-0.0019))} - \frac{32.73}{1.3661} = 0.27$€. The correlation between the GM share and the foreign exchange rate has already taken into account.

4.1.3 Monte Carlo simulation approach

The last presented method to calculate value-at-risk is the most powerful but also the most complex approach, the Monte Carlo simulation. Jorion defines Monte Carlo Simulation in finance as follows: "The Monte Carlo method approximates the behaviour of financial prices by using computer simulations to generate random price paths." [JOR97]. Therefore the basic elements are random numbers, which allow building these paths. The larger the numbers of simulations the more precise are the estimated results. "The law of large numbers ensures that this estimate converges to the correct value as the number of draws increase" [GLA03]. In general the standard error of the expected values is normally distributed with the statistical moments $\mu = 0$ and $\sigma = \frac{\sigma_f}{\sqrt{n}}$. This implies that the quality of the simulation improves with increased numbers of simulated paths n. So if you want to improve the quality with factor 10,

you'll need 100 times more simulation paths.

In the following lines I want to show one possible way to calculate a stochastic path for stock prices, based on a geometric Brownian motion (compare [GLA03]).

$$S(t_{i+1}) = S(t_i)exp([\mu - \frac{\sigma^2}{2}](t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i}Z_{i+1})$$

Whereas $Z = [z_1, z_2, \ldots, z_n]$ are independent standard normally distributed random numbers. The term $[-\frac{\sigma^2}{2}]$ is a correction if the arithmetic rate of return is used in the model. For geometric returns (like in the following example) this correction has not to be applied. Another assumption in this formula is that the stock prices are log normally distributed. If this formula is used for n paths, a distribution of asset prices is the result. This distribution can be used for value-at-risk in the same way as in the previous presented methods. For the common BMW example the VAR is evaluated as follows: At first the number of simulations has to be defined. In this case $n = 100,000$. Because of the short time interval (of just one day) the result is not really a path, it's a distribution of single data points (compare with Figure 4.4, which shows a path simulation for ten days). After that, n normally distributed random numbers (with $\mu = 0$ and $\sigma = 1$) are generated (vector Z with the values z_1 to z_n). Using the formula above, the result are 100,000 estimated prices for one BMW share for the next day.

$$S_1 = 43.8 * exp([2.189 * 10^{-4}] + 0.012 * Z$$

After calculating the 95% quantile of the distribution, the result is a maximum loss of 0.84€.

Another extension is needed if an investment depends on more than one risk factor (see for instance the GM example in Chapter 4.1.1). In this case the correlations of these risk factors have to be considered. One popular approach to do this is to generate correlated random variables, for instance with help of the Cholesky factorization. The variance-covariance matrix (Σ) is separated into a lower triangular matrix A and its transposed of matrix A^t.

$$\Sigma = A \cdot A^t = \begin{pmatrix} a_{11} & 0 & \ldots & 0 \\ a_{11} & a_{22} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \ldots & a_{nn} \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ 0 & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & a_{nn} \end{pmatrix}$$

this implies

$$\Sigma = \begin{pmatrix} a_{11}^2 & a_{11}a_{12} & \ldots & a_{11}a_{1n} \\ a_{11}a_{12} & a_{12}^2 + a_{22}^2 & \ldots & a_{12}a_{1n} + a_{22}a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{11}a_{1n} & a_{12}a_{1n} + a_{22}a_{2n} & \ldots & a_{1n}^2 + a_{2n}^2 \ldots a_{nn}^2 \end{pmatrix}$$

After multiplying the matrix of random numbers Z from the left by A^t, a random numbers matrix which reflects the correlations between each simulation factor is the result. There exist different ways to detect the Cholesky factor. Glasserman [GLA03] shows as one possible approach the iterative algorithm from Golub and Van Loan to find the factor. But there already exist a lot of implementations in the most popular mathematics software packages, which can be used for the Monte Carlo Simulaton.

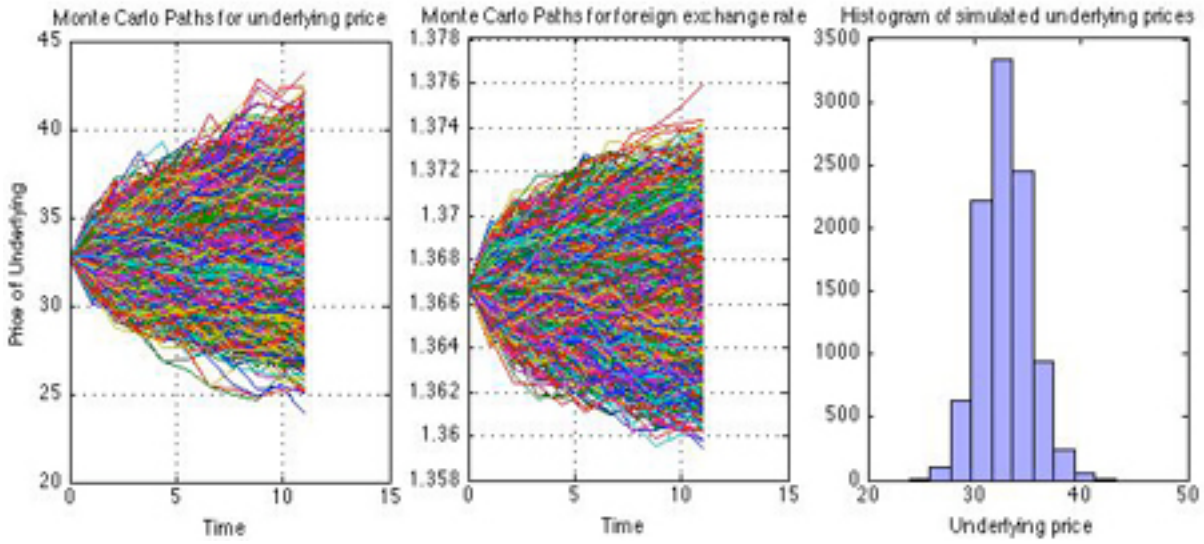

Figure 4.4: Example Monte Carlo Simulation - General Motors share

Haugh shows in [HAU04] a simple code fragment how to generate random numbers for three correlated variables in MatLab. This code is customized to the GM example now.

```
Sigma = [5.5082E-04 -3.2297E-07;  %1=GM  2=FX(EUR/USD)
        -3.2297E-07  4.8957E-03];
A=chol(Sigma);
Z=randn(2,100000); %generate a 2x100000 matrix with
                    random normally distributed numbers
X=A'*Z; % building correlated random numbers
```

With the random numbers matrix X it's possible to simulate the price path of the underlying and the foreign exchange rate (s. formula above). The simulation path for these two risk factors is shown in Figure 4.4. The result is a VAR of 1.11€ per share for a time horizon of ten trading days and a confidence level of 95%.

4.1.4 Method comparison

After discussing three possible value-at-risk evaluation methods, we want to take a closer look to the pros and cons of each approach.

The analytic delta-normal method is the simplest way to calculate VAR. There are only few historical data needed for the risk evaluation, the two statistical moments standard deviation and (depending on implementation) mean value of the risk factor returns and the actual price of the risk factor. To evaluate a whole portfolio or an investment depends on more than one risk factor, the variance-covariance matrix including all relevant risk factor is also needed. But there is a trade-off between the fast and simple calculation and the precision of the result. Due to the normal distribution assumption, it is possible to underestimate the risk of an investment. The reasons are unconsidered fat-tails or extreme values at the tail of a distribution.

A very important aspect for all approaches is the chosen time horizon of the historical time series. The BMW examples in previous chapters used the daily price data of the last three years. If we take for comparison reasons the historical data of the last ten years, the standard deviation would be 2.2% p.d. (compare 1.2% p.d. based on the three year time horizon). The outcome of this is a value-at-risk (calculated with the delta-normal approach) of 1.57€ per share, which is an 85% higher value at risk than the result using the three year time horizon. One of the reasons is the characteristic of time varying volatilities. The challenge is to choose the right length of the historical time series data. One possible solution is to use alternative volatility estimation approaches like EWMA, GARCH, etc. (see Chapter 4.3.2), where the influence of older data points decrease exponentially with a predefined factor.

The historical simulation is also a very simple method to calculate the risk for a wide range of investment products. Due to the abolition of the normal distribution assumption, the problem, underestimating the distribution tails (fat tails, etc.), doesn't exist anymore. However, there are also a number of criticism. The assumption, that the returns of the past reflect the returns in future could be a misinterpretation if some environmental parameters get changed. Furthermore in this model only one possible simulation path will be considered, unlike the Monte Carlo simulation approach. Another problem is to choose the right length of historical time periods (see above), because all data points have the same weight at the historical simulation (for instance, if historical data of the last 5 years are taken, an event 4 years ago has the same influence than an event happened yesterday). Another challenge is to handle the need of a huge amount of historical data for each risk factor. Especially for large portfolios the calculation is more complex and time intensive.

The biggest advantage of the Monte Carlo simulation is the power of the method, which allows evaluating a wide range of investment products. One of the weaknesses is the complexity of the underlying stochastic model. Jorion writes: "Another potential weakness of the method is that it relies on a specific stochastic model for the underlying risk factors as well as pricing models for securities such as options or mortgages. Therefore, it is subject to the risk that the models are wrong" [JOR97]. In literature you often find the argument that Monte Carlo simulations are due to the high resource requirements not useable for the daily business. Nowadays these arguments are not valid anymore. Due to cheap and fast computers (and the increasing use of distributed systems) the expenditure of time and costs are negligible. For example the option price simulation with MatLab, including four risk factors (price of the underlying, dividend yield, foreign exchange rate and interest rate) and simulating 100,000 paths where each path includes ten time points, takes 3 seconds on a customer computer[9] (which implies 4 million data point simulations).

4.2 Stock options

4.2.1 Properties

After discussing the risk evaluation of linear investments, we want to go one step further to calculate non-linear investments. As one example of these kinds of financial products the

[9] CPU: 2 Ghz Intel Core 2 Duo, 2 GB 667 Mhz DDR2 SDRAM, OS: Mac OS X 10.4.10

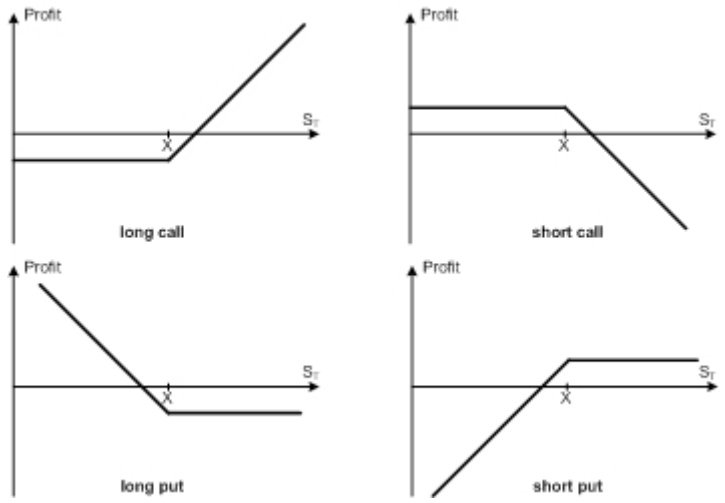

Figure 4.5: Payoffs from positions in European options [HUL05]

following chapter will discuss stock options in detail.

In general, options as one kind of derivative financial instruments, can be divided into two types:

1. Call options (of European style): the holder of a call option has the right but not the obligation to buy an asset for a predefined price (called strike price X) on a certain date (expiration date).

2. Put options: is the opposite case, the holder has the right to sell an asset, for a fixed price.

Due to the properties of an option, that the holder purchases the right to buy or sell an asset, he will only exercise the option if the price of the underlying is greater than the strike price for a call option or lower for a put option. Otherwise it would be possible to buy the underlying cheaper on the market or sell it for more money. With this behaviour the payoff of an option at the expiration date can be defined as follows:

$$C_L = max(S_t - X; 0) \qquad P_L = max(X - S_t; 0)$$

Whereas S_t is the price of the asset at the expiration date. This implies that the holder of an option (long position) cannot lose more than the paid option price. On the other side is the seller of the option (short position) who has the initial option price as fixed income and will lose if the counterpart wins.

$$C_S = min(X - S_t; 0) \qquad P_S = min(S_t - X; 0)$$

Figure 4.5 shows the payoff of each of these four possibilities in a graphical way.

There also exist different types of exercising an option. The two most important types are:

- European options: give the holder the right to exercise an option at the expiration date.

- American options: allow the holder to exercise an option at any time up to the expiration date. There are some situations where the holder should make use of this right, which will be discussed later in this chapter.

On the other hand can be also find a lot of "exotic" options, for example:

- Asian options: the payoff of the option doesn't depend on the price of the asset at expiration date; instead of that, the average price of a predefined period will be taken.
- Bermudian options: A Bermudian option is a special case of an American option where the exercise of the option before the expiration date can be limited to certain dates.

Most options traded at the option market are American style. As already mentioned before, American options can be exercised at any time up to the expiration date. But when does such early exercise make sense? There exist differences between call and put options. "It is never optimal to exercise an American call option on a non-dividend-paying stock before the expiration date." [HUL05] With the assumption that the holder of the option plans to keep the asset for the remaining life of the option, there are two reasons the holder doesn't exercise the option earlier. First, the strike price has to be paid at the expiration date. Therefore for this amount of money, interest is earned for the remaining time period. Second, if the option is in-the-money (call option: $S > X$; put option $S < X$) there is also the chance that the underlying price falls under the strike price during the holding period, where an early exercise would be negative for the option holder. If the investor identifies the trend of falling underlying prices, it will be better to sell the option (or choose another short-strategy) instead of exercising earlier. Another situation can be found in the case of options on dividend-paying stocks. A dividend payment (D) implies a drop of the stock price to $S_t - D$. In this case it could be "optimal to exercise an American call immediately prior to an ex-dividend date." [HUL05] Unlike an American call option, the early exercise of an American put option (on a non-dividend-paying stock) can be more practical. Hull writes: "a put option should always be exercised early if it is sufficiently deep in the money." [HUL05] If a put option is seen as insurance against falling stock prices (the investor holds a long position of the underlying stock), it is better to realize the strike price earlier, because it isn't possible to increase the profit in situations with continuing falling stock prices. For holders of put options, dividend payments have a positive effect. Therefore an early exercise is not expedient.

4.2.2 Pricing

This chapter discusses two methods to evaluate the price of a stock option:

1. Binomial tree approach
2. Black-Scholes model

A general principle for both methods is the assumption of a risk-neutral world. In this scenario the expected return on all assets is equal the risk-free interest rate

$$E(S_T) = S_0 e^{rT}$$

Hull writes: "... The principle states that we can with complete impunity assume the world is risk neutral when pricing options. The resulting prices are correct not just in a risk-neutral world, but in other worlds as well." [HUL05]. For both approaches a common example is used. The evaluation date is August 1st 2007 again. The objective is to calculate the price of the following stock option:

- Kind of option: Call option (long), European style
- Underlying: BMW share (for details see chapter 4.1)
- Price of the underlying at 8/1/2007 (S_0): 43.8€ σ=18.97%
- Expiration date: 3/19/2008 $\rightarrow$ maturity (T): 0.633 years
- Strike price (X): 46€
- Risk-free interest rate (EURIBOR 8M[10]): 4.43%

4.2.2.1 Binomial tree

The basic idea of constructing a binomial tree is to build paths of simulated underlying stock prices by dividing the maturity in n sub periods. At every "time-point" there are two possibilities, the stock price will increase with a probability of u or will fall with a probability of d. The number of simulated underlying prices grows exponential with the number of chosen sub periods (2^n). In the example, if we would choose n with the number of trading days up to the expiration date (162 days), 2^{162} potential underlying prices would be calculate, which are $5.45E + 48$. This huge number is also for a (up-to-date) customer computer no problem to calculate. "When binomial trees are used in practise, the life of the option is typically divided into 30 or more time steps." [HUL05]. Anyway, for a better illustration 4 sub-periods are chosen for this example.

Let's come back to the calculation algorithm. At first all possible price paths are calculated from the evaluation date to the expiration date of the option.

$$S_{t+1} = S_t * u^i * d^j$$

where

$$u = e^{\sigma\sqrt{\frac{T}{n}}}$$
$$d = e^{-\sigma\sqrt{\frac{T}{n}}} = \frac{1}{u}$$

where i= number of previous up moves und j = number of previous down moves = $t - i$, σ is the volatility of the underlying.

After that, the option price is calculated. Starting at the expiration date, the option price at the time-point T can be evaluated with

[10] www.euribor.org/html/content/euribor_data.html [Last Access: 12/10/2007]

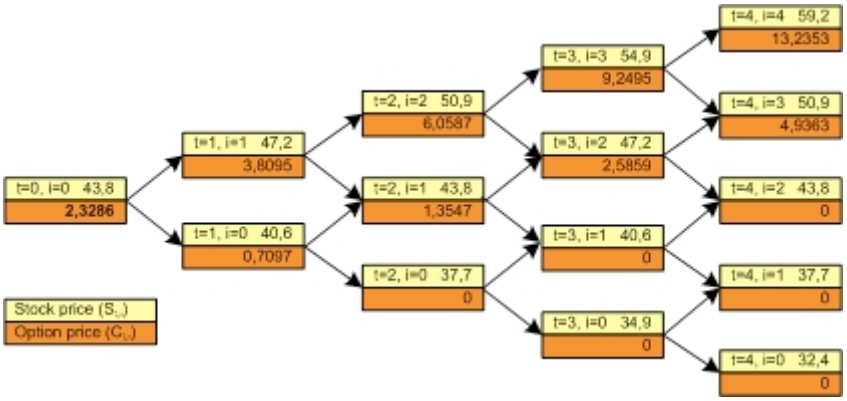

Figure 4.6: Example binomial tree - BMW stock option

$$C_{T,i} = max[(S_{T,i} - X), 0] \qquad P_{T,i} = max(X - S_{T,i}; 0)$$

Now for every data point from period T to period 0 the option price is calculated iterative.

$$C_{t,i} = e^{-r\frac{T}{n}}[pC_{t+1,i+1} + (1-p)C_{t+1,i}] \qquad P_{t,i} = e^{-r\frac{T}{n}}[pP_{t+1,i+1} + (1-p)P_{t+1,i}]$$

where

$$p = \frac{e^{r\frac{T}{n}} - d}{u - d}$$

The variable p is the risk neutral probability and can be interpreted as the probability of an increasing stock price.

Figure 4.6 shows the binomial tree of the BMW example. The option price at August 1st 2007 is 2.3286€. In the model shown above dividend payments are not taken into account, to do this, it has to be extended with the dividend yield (q) of the underlying stock share. As mentioned before in a risk-neutral world the expected total return of an asset is equal to the risk-free interest rate. If the dividend yield is greater zero, the expected return from capital gains must be $r - q$. Therefore the following formula has to be adapted:

$$p = \frac{e^{(r-q)\frac{T}{n}} - d}{u - d}$$

4.2.2.2 Black-Scholes model

The most popular way to calculate the price of an option is the use of the Black-Scholes model. The basic idea is to build a risk-free portfolio including the asset and a depending derivative. Therefore as already mentioned before the return of the portfolio must be equal the risk-free interest rate. But unlike the binomial tree approach the risk-less assumption of the portfolio applies only for a short time period and has to be adjusted frequently. This adjustment has to be considered in formula.

The price of the underlying follows a stochastic process, called Wiener process or Brownian motion. I will discuss this topic in the next chapter also if the concept is part of the Black-Scholes formula. The following input parameters are needed:

- S_0: price of the underlying at the evaluation date
- r: risk-free interest rate
- X: strike price
- T: maturity (in years)
- σ: the volatility

Assumptions:

- the volatility is constant during the whole life of the option
- there are no dividend payments
- the risk-free interest rate is also constant
- the formula covers only European-style options (except American call-options without dividends)

$$C = S_0 N(d_1) - X e^{-rT} N(d_2)$$

or

$$P = X e^{-rT} N(-d_2) - S_0 N(-d_1)$$

where

$$d_1 = \frac{ln(\frac{S_0}{X}) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{ln(\frac{S_0}{X}) + (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$$

The function $N(d)$ denotes the cumulative normal distribution with $\mu = 0$ and $\sigma = 1$. The result is the probability that an observation will fall in the interval $[-\infty, d]$. This function can be found in all common mathematic software packages (see e.g. normcdf in MatLab, pnorm in R or in Microsoft Excel the function normsdist).

There exist a lot of extensions of the basic Black-Scholes formula to eliminate some of the assumptions. For instance it's possible to establish a stochastic model for the interest rates, so that r is not constant for the whole life of the derivative. Another model allows the approximate evaluation (e.g. the quadratic approximation of Barone-Adesi and Whaley [BAW87]) of American call-options (in consideration of dividend payments, which can lead to an early exercise). Anyway, another extension which is needed in further chapters is the

consideration of dividend payments. Therefore the extension of Merton (1973) is taken. As already defined in the previous section, q is the dividend yield of the underlying stock.

$$C = S_0 e^{-qT} N(d_1) - X e^{-rT} N(d_2)$$

or

$$P = X e^{-rT} N(-d_2) - S_0 e^{-qT} N(-d_1)$$

where

$$d_1 = \frac{ln(\frac{S_0}{X}) + (r - q + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{ln(\frac{S_0}{X}) + (r - q - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$$

If we take a look to our example, the evaluated option price is 2.2172€. In comparison with the binomial tree calculation it's a quite large difference of the prices of approximately 5%. This effect can be explained with the chosen number of sub periods. For demonstration reasons for 162 trading days only 4 time points were simulated at the binomial tree. If we would simulate every trading day, the result is exactly equal to the option price calculated with the Black-Scholes formula.

4.3 Risk measurement

In this chapter, two methods to calculate value-at-risk for stock options will be discussed. In the prototype of this work, both approaches will be implemented and it should be possible to compare the advantages and problems of the different methods.

4.3.1 Analytic approach

In the previous chapter the risk evaluation for linear investments was discussed. But these methods will not lead to satisfying results in view of the non-linear behaviour of options. Therefore the delta-normal method is extended with the gamma moment, building the delta-gamma approximation (see Figure 4.7). At first, the option sensitivities are discussed in detail:

- Delta: for a call option: $\Delta_c = \frac{\partial f}{\partial S} = e^{-qT} N(d_1)$ and for a put option: $\Delta_p = \frac{\partial f}{\partial S} = e^{-qT}[N(d_1) - 1]$ "The delta of a stock option is the ratio of the change in the price of the stock option (f) to the change in the price of the underlying" [HUL05]

- Gamma: $\Gamma = \frac{\partial^2 f}{\partial S^2} = \frac{e^{-qT}\Phi(d_1)}{S\sigma\sqrt{T}}$ Gamma is the second partial derivative of the option price with respect to the underlying price and measures the sensitivity of delta in view of the underlying price. For instance a high Γ implies that Δ changes fast as S changes. Gamma can have positive as well as negative values. "Positive gamma is beneficial, as it implies that the value of the asset drops more slowly and increases more quickly than otherwise. In contras, negative gamma can be dangerous because it implies faster price falls and slower price increases" [JOR05]

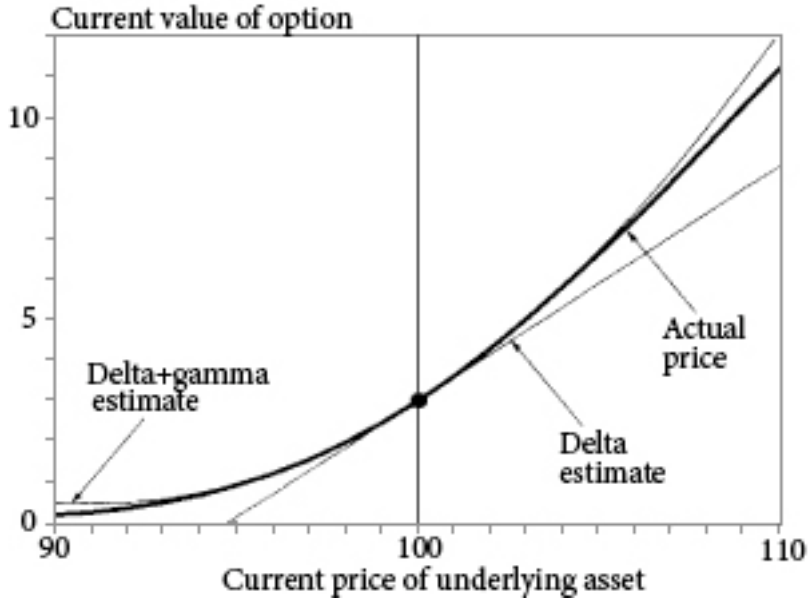

Figure 4.7: delta-gamma vs. delta approximation for a long call [JOR05]

- Vega: $\Lambda = \frac{\partial f}{\partial \sigma} = Se^{-qT}\sqrt{T}\Phi(d_1)$ The volatility is a very important input parameter to evalute the option price with help of the Black-Scholes formula. Vega measures the sensitivity of σ to the option price.
- Rho: Call: $\rho_c = \frac{\partial f}{\partial r} = Xe^{-qT}TN(d_2)$ Put: $\rho_p = \frac{\partial f}{\partial r} = Xe^{-qT}TN(-d_2)$ shows the sensitivity of the interest rate to the option price. A higher interest rate increases the probability that $S > X$ and the option will be exercised (and opposite for put options).
- Theta: for a call option: $\Theta_c = \frac{\partial f}{\partial T} = -\frac{Se^{-qT}\sigma\Phi(d_1)}{2\sqrt{T}} + qSe^{-qT}N(d_1) - rXe^{-rT}N(d_2)$ For a put option: $\Theta_p = \frac{\partial f}{\partial T} = -\frac{Se^{-qT}\sigma\Phi(d_1)}{2\sqrt{T}} - qSe^{-qT}N(-d_1) + rXe^{-rT}N(-d_2)$ Theta has usually a negative value. This is because; as time goes by the option loses value, ceteris paribus.

Let's come back to the delta-gamma risk evaluation approach. Based on the sensitivity variables above, the change of the option value can be written as:

$$dc = \frac{\partial f}{\partial S}dS + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}dS^2 + \frac{\partial f}{\partial \sigma}d\sigma + \frac{\partial f}{\partial r}dr + \frac{\partial f}{\partial q}dq + \frac{\partial f}{\partial T}dT$$

Jorion calculates the value-at-risk of an option in [JOR05] as follows:

$$VAR(df) = \Delta \times VAR(dS) - \frac{1}{2}\Gamma \times VAR(dS)^2$$

where

$$VAR(dS) = \Phi(\alpha)\sigma S_0\sqrt{T}$$

This formula has one weak point. The assumption is that the returns of the underlying are normally distributed. But the quadratic term of the VAR are not normally distributed anymore, rather it's distributed as a non-central chi-squared. "The problem is therefore that of characterising the distribution of a sum of a normal variate and a non-central chi-squared variate" [BJS99]. Britten-Jones and Schaefer have described a better solution to solve this problem in their paper "Non-Linear Value-at-Risk" ([BJS99]).

$$df = \mu + \Delta dS + \frac{1}{2}\Gamma(dS)^2$$

where

$$\mu = \Theta t$$

t is the target horizon of the value-at-risk calculation. Θ considers the value change of the option due to t. The changes of the underlying price have the assumption of a normal distribution with $N(\mu_S, \sigma_S^2)$ with the mean and the variance of the underlying returns. As already mentioned before, $(dS)^2$ are not longer normally distributed, but it's distributed as a noncentral chi-squared. Therefore to avoid this "mix" of distributions, the formula is extended to get only a chi-squared distribution.

$$df = \mu^* + \frac{1}{2}\Gamma(e + dS)^2$$

where $e = \frac{\Delta}{\Gamma}$ and $\mu^* = \mu - \frac{1}{2}\frac{\Delta^2}{\Gamma}$ Now it's possible to define the non-central chi-squared distribution $\omega \sim \chi^2 : \upsilon, d$ where $\upsilon = 1$ and $d = (\frac{e+\mu_S}{\sigma_S})^2$. The value at risk of the option can be calculated with:

$$VAR(\alpha) = \mu^* + \frac{1}{2}\Gamma\sigma_S^2 w(\alpha)$$

This approach shows only the delta-gamma approach for one option, to evaluate the risk of a whole portfolio with a mix of linear and non-linear investments I refer to [BJS99], where the multivariate case is described in detail.

The next section shows the risk evaluation of derivative instruments with help of the Monte Carlo simulation. The calculation of big portfolio implies the simulation of many risk factors. Glasserman (in [GHS01]) presented therefore a method to decrease the number of needed scenarios, combining the advantages of the simulation and the delta-gamma approaches. This modification will not be discussed further but could be an extension in the risk measurement system.

4.3.2 Monte Carlo simulation approach

Also, in this case, the maximum loss of the investment during a defined time period (T) should be calculated. There exist several ways to find the distribution of option prices at the time point t_T. In this project a very simple approach is taken, where the following assumptions have to be specified:

1. the option price at t_0 is (or can be) calculated with the Black-Scholes model

2. the option price at t_T is calculated with the Black-Scholes formula, too.

Therefore the following workflow can be used to calculate VAR:

1. simulate the price paths of all risk factors up to the time point T
2. calculate the option price for all scenarios with the Black-Scholes formula. The result is a distribution of option prices.
3. choose the α-quantile of the distribution
4. $VAR(\alpha) = c_0 - c_T$ where c_0 is the option price at the evaluation date and c_T is the α-quantile of the simulated option price distribution at time point T

The simulation of the price paths of the risk factors is equivalent to the previous described method (see chapter 4.1.1). To evaluate VAR of a stock option, four, respectively five (for stock options quoted in foreign exchange) risk factors have to be simulated (price of the underlying, risk-free interest rate, volatility of the underlying, dividend yield, foreign-exchange rate).

The correlations between the risk factors are taken into account with help of correlated random numbers (using the Cholesky-factorization). There exist one special case of simulating a price path, which concerns the interest rates. Interest rates have the interesting property of mean reversion, which means that the interest rate always tends to the long-term average level. Hull [HULL05] demonstrates three models to simulate an interest rate. The first one is the Rendleman and Bartter model:

$$dr = \mu r dt + \sigma r dz$$

It's similar to the already discussed geometric Brownian motion formula to calculate random walk processes for stock prices. Therefore it is easy to implement, but it doesn't take care of the mean reversion effect and produces negative interest rates with positive probability. Another approach is the Vasicek model that captures mean reversion.

$$dr = a(b - r)dt + \sigma dz$$

In Vasicek's model it's also possible that the interest rate becomes negative. An improvement of the model, which solves this problem, is the Cox, Ingersoll, and Ross model:

$$dr = a(b - r)dt + \sigma\sqrt{r}dz$$

In both methods it's necessary to determine the constants a and b. Whereas "b can be interpreted as long-run interest rate level and a as the speed at which r is pulled toward b" [GLAS03]. The chosen time period for value-at-risk is typical between 1 and 10 days. It has to be analysed if the additional implementation and runtime costs effect an improvement of the simulation results.

The standard deviation plays a very important role in the stochastic simulation process as well as in the formula of Black-Scholes. Therefore different methods to evaluate the volatility of a risk factor should be discussed in detail.

1. "standard"-method: $\sigma = \sqrt{\frac{1}{(T-1)} \sum_{i=1}^{T} (rt_i - \hat{\mu})^2}$ One of the disadvantages of this method is, that all time series entries are equally weighted. This implies that extreme events in the past (which are not relevant today) influence the volatility.

2. EWMA (exponentially weighted moving average) approach: $\sigma_t^2 = \lambda \sigma_{t-1}^2 + (1-\lambda) rt_{t-1}^2$ where rt is the return of two data points at time $t-1$. λ is also called decay-factor and specifies the constant weight of the historical data. It's usually set between 0.94 and 0.97 (compare [ME01], [JPM96] set λ equal 0.94). This method eliminates the problem of historical "outlier" and allows a quick iterative implementation with good runtime behaviour.

3. GARCH(1,1) (generalized autoregressive conditional heteroskedasticity): $\sigma_n^2 = \omega + \alpha u_{n-1}^2 + \beta \sigma_{n-1}^2$ EWMA can be seen as special case of GARCH(1,1) where $\omega = 0$. The difference between this two approaches is, that GARCH considers the mean reversion effect (similar to the stochastic process for interest rates) where the volatility tends to its long-term average level over time (called V_L, $V_L = \frac{\omega}{1-\alpha-\beta}$).

4. implied volatility: this is very common approach to determine the volatility of the underlying based on the Black-Scholes formula. If all required parameters (except σ) of the Black-Scholes option price formula are given and the option price is also known, the volatility can be calculated.

Furthermore, even if the volatility is estimated, at the simulation of the option price the volatility smile effect has to be considered. Which means that the (implied) volatility of a stock option increases if a call is in-the-money ($S > X$) or a put is out-of-the-money ($S < X$), and vice versa.

This value-at-risk simulation approach is a very simple method but it provides a good basis for further extensions and improvements. Therefore it fits perfectly to the designed risk measurement system, where one of the major goals is a fast development of the basic financial instruments, but it should allow continuous enhancements. For instance it's possible to adjust the stochastic processes to the particular risk factors (see [GLA03]) to provide a better prediction.

5 Prototyping

5.1 Task definition

After discussing the design of the risk evaluation system and the theoretical background of the value-at-risk calculation, with the main topic risk evaluation of stock options, this chapter covers the last part of this work, the implementation of the prototype. The objective is to show the possibilities of an open and flexible system. There are not enough personal resources to implement a complete risk evaluation system with a wide range of investment product types in a practical time. Therefore the prototype consists of only two products, stocks and stock options. The specific tasks are:

- set up the required infrastructure (see next session)
- implement the database of the risk measurement system (see Section 3.4)
- implement the data-access web services
- develop web services to evaluate investments with the following approaches:
  - stocks: Monte-Carlo simulation approach
  - stocks: delta approach
  - stock options: Monte-Carlo simulation approach
  - stock options: delta-gamma approach
  - portfolios: Monte-Carlo simulation approach
  - portfolios: delta-gamma approach
- configure the EDA-system and orchestrate the services
- develop a simple web application for user interactions

Due to the limited resources, there are also some simplifications. The system architecture (see Section 3.2) shows the investment administration database outside of the risk evaluation system. In the prototype this expedient assumption would require an enormous additional effort. Instead, the investment administration is managed over a web interface, where the transactions are written directly to the database of the risk evaluation system. The second limitation affects the access of the required historical market data. The available infrastructure has no access to one of the common data providers (Bloomberg, Thomson Datastream, etc.). That's the reason why all data had to be inserted into the database with help of CSV-files. Therefore not all processes of the designed workflows can be implemented. The daily update routine (see

Figure 3.2), for instance, downloads all new data points from Bloomberg at the beginning of the workflow. Due to the missing data providers an implementation of this workflow as part of the prototype doesn't make sense. The same problem can be found adding a new investment. To solve this kind of problems a wide number of investments (equities and stock options), including all relevant price data, were added to the system. In the prototype only transactions of already existing investments are taken into account. Note: the limitations are only necessary in this prototype and are no general weakness in the system design.

5.2 Infrastructure

The prototype needs a comprehensive infrastructure, which will be described in this section. The used hardware is an Apple MacBook with the following specifications:

- CPU: Intel Core 2 Duo 2GHz
- RAM: 2 GB 667 MHz DDR2 SDRAM
- Operating system: Mac OS X 10.5.2

The prototype should illustrate the advantages and chances of a distributed system. In general the required services are installed on two different operating systems. The host operating system is Mac OS X. Additionally a virtual machine (VMWare Fusion 1.2) has been installed with a second operating system (Microsoft Windows XP SP 2). The guest system is in the "bridge" network mode, this means that the Windows system gets its own IP address and acts as an independent PC. Both systems have to share the hardware, whereas every operating system has access to one CPU core. The maximum available RAM for the guest system is 600MB (which slows up the performance a little bit).

The following software components were used to implement the prototype:

- MySQL 5.0.45 database server (Mac OS)
- Microsoft Visual Studio 2005 professional edition (Windows)
- Microsoft Internet Information Server 5.1 web server (Windows)
- Mathworks MatLab 2007a (Windows)
- Sun Java System Application Server 9.1.01 (Mac OS)
- Sun Java JDK 5 (Mac OS)
- Eclipse Platform 3.3.0 (Mac OS)
- Adobe Flex Builder 2.0 (Mac OS)

A very important part of the prototype is the event-driven architecture supporting framework. This system orchestrates and coordinates the single processes to form a complete application. The market for SOA-systems is quite large with many different vendors and product solutions. But EDA is a relative young field with less literature, best-practice papers, etc. In general the existing systems can be split into three categories:

1. prototypes from a couple of universities which cover a specific domain
2. open-source community projects
3. commercial products

It wasn't possible to find any papers, which discuss the advantages and problems of the existing systems. There are some web-blogs (for instance the blogs from Marco Seiriö[1], Mårten Gustafson[2] and Jack van Hoof[3]) where researchers and software developers try to collect the news in this field and categorize the software products. But there hasn't been published any summarization yet. Due to these problems it wasn't easy to find the "right" framework for the prototype. After several experimental small prototypes I decided to choose the Mule-framework[4], due to the following characteristics:

- the framework is implemented in Java, therefore it can be used on several platforms. In this project Mule is installed on Mac OS X.
- Mule is an open-source project, there are no software costs
- there exist a lot of interfaces to other systems. For example: SOAP-provider for accessing web services, email provider, but also interfaces to legacy systems (e.g. for the IBM AS-400 mainframe)
- many implemented examples
- a lot of possible configuration settings
- integration into the Eclipse-Framework

5.3 Implementation - risk evaluation of stock options

The whole application consists of many independent modules, which build together the risk evaluation system. The required processes and workflows were identified in Chapter 3. In this section the implementation process is described in detail. Every process should be an independent part of the system. So it's possible to use the implemented modules in different projects. Therefore every process is implemented as loosely-coupled web service with clearly defined

[1] http://rulecore.com/CEPblog/
[2] http://marten.gustafson.pp.se/blog/
[3] http://soa-eda.blogspot.com/
[4] http://mule.mulesource.org/

interfaces. The services can be distributed on different servers to improve the performance behavior. It's also possible to use different kind of programming languages to implement the web services. To show the flexibility of such system, the prototype uses two different servers (with two different operating systems). The database, the database access services (implemented with Java web services) and the EDA-framework (which represents the final application) run on the Mac OS system. The investment evaluation services are implemented with MS .net 2.0 and Mathworks MatLab (see next section) and use MS Windows XP as host system.

The Mule-framework allows to orchestrate the single processes to form the entire application. The basic elements of the orchestration are messages and events. The transmission of the in- and output parameters is handled with help of messages. The transformation of these messages is task of the EDA-system (e.g. the result of one service is the input parameter of another service). Mule provides predefined transformers, for example to generate an instance of an object from XML (the result of a web service). But it's also possible to write its own transformers in Java and integrate it into the project. Every process in this project throws an event after finishing its task. Any other process can subscribe these events. With this kind of subscription mechanism, the complete workflow can be modeled.

In the following pages I want to describe some interesting special aspects of the prototype. The first section covers the integration of MatLab in MS .net. The second part describes the advantages of using the O/R mapping framework Hibernate to improve the flexibility of the software. The last section introduces a technology to develop web applications called Adobe Flex.

5.3.1 Integration of Mathworks MatLab in MS .net

The goal of the application is to allow a fast and efficient development. In many cases the responsible risk managers or financial engineers design the risk evaluation models for the required investment types, but they are not qualified to implement the model with an object oriented programming language. The briefing of a software developer often takes more time than the actual development process. On the other hand a lot of scientists are able to write functions with common mathematic frameworks like MatLab, R, etc. In my opinion, it is not practical to implement a whole project (depending on the scale of the project) with such a framework. The stability, scalability, expandability and the runtime-behavior of an application require the use of "higher" programming languages. This prototype bases on the service oriented and event oriented paradigm, respectively. Therefore every process is an independent part of the system. This characteristic allows the integration of a mathematic framework in the following way: every process has a predefined interface. The implementation itself can be seen as black box. Therefore it's possible to implement a method, which covers a specific part of functionality, e.g. evaluate the risk of a stock option with help of the Monte-Carlo simulation approach. The use of such framework has several advantages. As already mentioned the designers of the risk evaluation models are qualified to use one of the common mathematic software products in many cases. There already exist a lot of functions, e.g. to generate "high-quality" random numbers or to solve specific problems like the implementation of the Cholesky-factorization (see Section 4.1.3), in these frameworks, which save a lot of work and time for the developers. Another advantage is the use of matrix operations, which allow a very

fast and simple way to calculate a huge amount of data.

It doesn't make sense to implement part of the functionality in a mathematic framework, if there is no possibility to integrate it in the existing system. Therefore the objective is to call the functions within a development framework with web service support. In this prototype the mathematic framework MatLab and the development framework MS .net 2.0 are used. There shouldn't be any problems to use alternative frameworks, for example R and Java, but I haven't tested any of these other combinations.

There are two different approaches to integrate MatLab functions into MS .net with different resource requirements. Both approaches use MS Windows (in this project Windows XP SP2) as operating system.

1. One possibility is to use an instance of MatLab to call MatLab-functions directly. To do this, MatLab is started as background process with help of the command "matlab -regserver". The Microsoft .net project has to import the library Matlab.Application. Now it's possible to call MatLab-functions in the same way like in the MatLab-console. The following example shows one function call (where the variable cmd symbolize the term $a = sqrt(9) + 3;$)

   ```
   object oMatlab=Interaction.CreateObject("Matlab.Application","");
   object[] argsGet=new object[1];
   string cmd="a=sqrt(4)+3;";
   argsGet[0]=cmd;
   LateBinding.LateGet(oMatlab,null,"Execute",argsGet,null,null);
   ```

   The advantages of this approach are the simple implementation, a powerful debug-mode and no additional license fees. The disadvantages are that an instance of MatLab has to be installed on every single computer where the functions are needed (one exception is the use of web services, in this case only one instance has to be installed on the web server). Another problem could be the software reliability. If the instance of MatLab shuts down, the whole .net application will fail. One important aspect using MatLab in cooperation with web services is the security. Therefore it's recommended to check possible lacks in security before using it in a productive environment (that's also necessary for the second approach)

2. Another way to integrate MatLab in .net is the use of the MatLab Compiler[5], which creates a dll-file including all compiled user functions. The dll-file can be integrated in Microsoft .net. Mathworks published a very good introduction with all required steps in [MAT07]. One requirement to use the dll-file on a Windows computer is the installation of MatLab or the MatLab Runtime. In my opinion, this approach is more reliable because there is no background process required. That's the reason why this approach is chosen in the prototype. The weaknesses are a very bad error handling (there is no simple mechanism to debug the application) and the complex handling of variables (e.g. a 5x5 matrix has to be transformed to a vector/array with length 25 and the additional information 5 (=number of columns)).

[5] http://www.mathworks.com/products/compiler/

5.3.2 Flexible database management with Hibernate

The database is probably one of the most dynamic components in the system. The integration of every new type of investment requires a change in the database structure. It was tried to model an open database with help of inheritance mechanisms already in the database design phase. But every change in the database model also influences the data access components. Due to the gap between relational databases and the object-oriented paradigm, it's a lot of work to implement the required access functions (e.g.: insert a new dataset, load a single object, delete a dataset, deliver a set of entries, etc.). Object-relational (O/R) mapping frameworks offer an enormous simplification of these tasks.

In this project the open-source toolkit Hibernate[6] is used in combination with Java (alternatively there exists also a ported version for MS .net called NHibernate). Hibernate allows a database independent implementation. Therefore an own SQL-dialect (called HQL), which represents an additional layer over the vendor specific SQL-dialect, is available. Hibernate transforms the HQL-statements in the specific database language. This allows the change of a database, without any additional work, on the fly; only the new database provider has to be defined in the configuration file. The use of Hibernate is very simple; in general three types of configuration files exist:

1. Java beans: define the structure of the data objects, which includes all attributes and the corresponding get- and set-methods.

2. database mapping files: in these configuration files the single database tables are defined. This includes the data attribute specifications as well as the mapping between the corresponding Java bean classes and the related database tables (including the specification of the foreign key-behavior (cascade delete, etc.))

3. Hibernate configuration files: these configuration files include the specific database settings, e.g.: which database is used (Oracle, MS SQL Server, MySQL, etc.), what are the database access data (user name, password, database catalog name, IP address, etc.) and finally a list with all required database mapping files

These configuration files allow Hibernate to create all tables in the database (including all relationships) automatically. Furthermore all basic manipulation and access methods are already implemented within Hibernate. For example adding a new risk factor type can be realized with the following code:

```
HibernateUtil.beginTransaction();
Session session = HibernateUtil.getSession();
RiskFactorType type = new RiskFactorType();
type.setName("Interest");
type.setDescription("Interest␣rate␣...");
session.saveOrUpdate(type);
session.flush();
```

[6] http://www.hibernate.org

5.3.3 User interface

The user interface is realized as web application. The risk evaluation system itself, doesn't require any user interaction, it works completely autarkic. But there are several reasons why an implementation makes sense anyway:

1. documentation: for risk manager, it is of interest to see the historical time series of the underlying risk factors or more general what risk factors exist in the system and in which classes are they categorized.

2. administration: it's necessary to provide a comfortable way to add new risk factors and change the basic system settings. The prototype infrastructure doesn't include a portfolio administration database. Therefore it's necessary to implement another interface, which allows creating portfolios and adding or removing investments, respectively. This functionality is also part of the developed web application.

3. presentation: the risk evaluation system calculates many investments with different approaches. It is all the more important that the results are illustrated in a well-arranged way. Web applications offer very powerful techniques to present statistical data in table form as well as with help of diagrams.

The service oriented and event based approach (with standardized interfaces) allow the use of a broad wide of web application technologies (PHP, MS ASP.net, etc.). In this prototype, the newish framework Adobe Flex 2[7] was chosen, due to the following reasons:

- comfortable development environment Adobe Flex builder (bases on the Eclipse framework[8])
- strict separation of the presentation and business logic. This characteristic fits perfectly to the concept of EDA.
- easy integration of web services
- a lot of already existing components (especially a wide variety of different types of diagrams) allow a fast and easy development.
- The underlying technology of Flex is Adobe Flash. This allows the implementation of rich applications without use of complex Java-script code, also in an asynchronous way (buzzword AJAX).
- my personal interest to prove this technology in a practical context.

The topic Flex covers a huge area and it's not possible to describe everything in detail. Rather, I want to give a short introduction of the basic principles of the technology. The layout of the application, with all the used components (text fields, labels, combo-boxes, diagram-elements, etc.), their styles and workflows, is defined in MXML-files. The business logic has

[7] http://www.adobe.com/devnet/flex/
[8] http://www.eclipse.org

to be implemented with other frameworks. There exist several interfaces for the most common technologies (e.g. Java, MS .net or web services) to connect the graphical interface with the background processes. In certain circumstances it could be necessary to customize single Flex components, therefore the programming language ActionScript (the same language can be found in Adobe Flash) is provided. The compilation of the application generates a SWF-file, which can be integrated in every web page. The deployment of the web application requires the Adobe Flash-player in version 9 at least. This application/plug-in is available for all common operating systems and web browsers. Therefore one of the objectives of this project, the platform independency, is still considered. For more details about the technology and its use, I refer to the web page of Adobe[9], the Flex community portal[10] and the following article [WOL06].

In consideration of the described requirements, the prototype has implemented the following functionality. The first module provides the documentation of the used risk factor data (see Figure 5.1). This view shows all risk factors in the system database, group by their categories (e.g. interest rates, exchange rates, etc.). After selecting a single risk factor, a detailed view with all master data (e.g. which is the source of data: Bloomberg, Reuters, etc.) and a table with all historical price data is shown.

The second important module is the portfolio administration (see Figure 5.2 - 5.3). This component allows the creation of new portfolios and the administration of the single investments (buy and sell transactions). The prototype has implemented the workflow illustrated in Figure 3.5. After buying or selling an investment, the risk evaluation system should calculate the risk of the new investment and reevaluate the risk of the entire portfolio. Therefore the web application has to communicate with the EDA-system. This is realized with help of web services again. The messaging server has subscribed the buy/sell investment method of the corresponding web service. After consuming one of the provided methods the defined workflow procedures are triggered. The last part presents the risk evaluation results. The challenge is to consider the different versions of the risk results as well as the different calculation approaches, which leads to a trade-off between the amount of information and the usability of the application. The application illustrates the value-at-risk and the market value of an investment and the portfolio in a numerical as well as in a graphical way (see Figure 5.4).

The web application provides only a very small functionality in this prototype. There exist many possible extensions, for example extended user interfaces to customize the user settings, downloading time series in different formats (e.g. Excel, CSV, XML, etc.) and many more.

5.4 Results

The designed risk evaluation system allows the calculation of investments with different evaluation approaches at the same time. The prototype implements the workflow from Figure 3.5, where all financial products are evaluated with the delta-gamma approach as well as with the Monte-Carlo simulation approach. In Chapter 4 the theoretical background of these two approaches were discussed in detail. The implemented prototype allows a simple numerical

[9] www.adobe.com
[10] www.flex.org

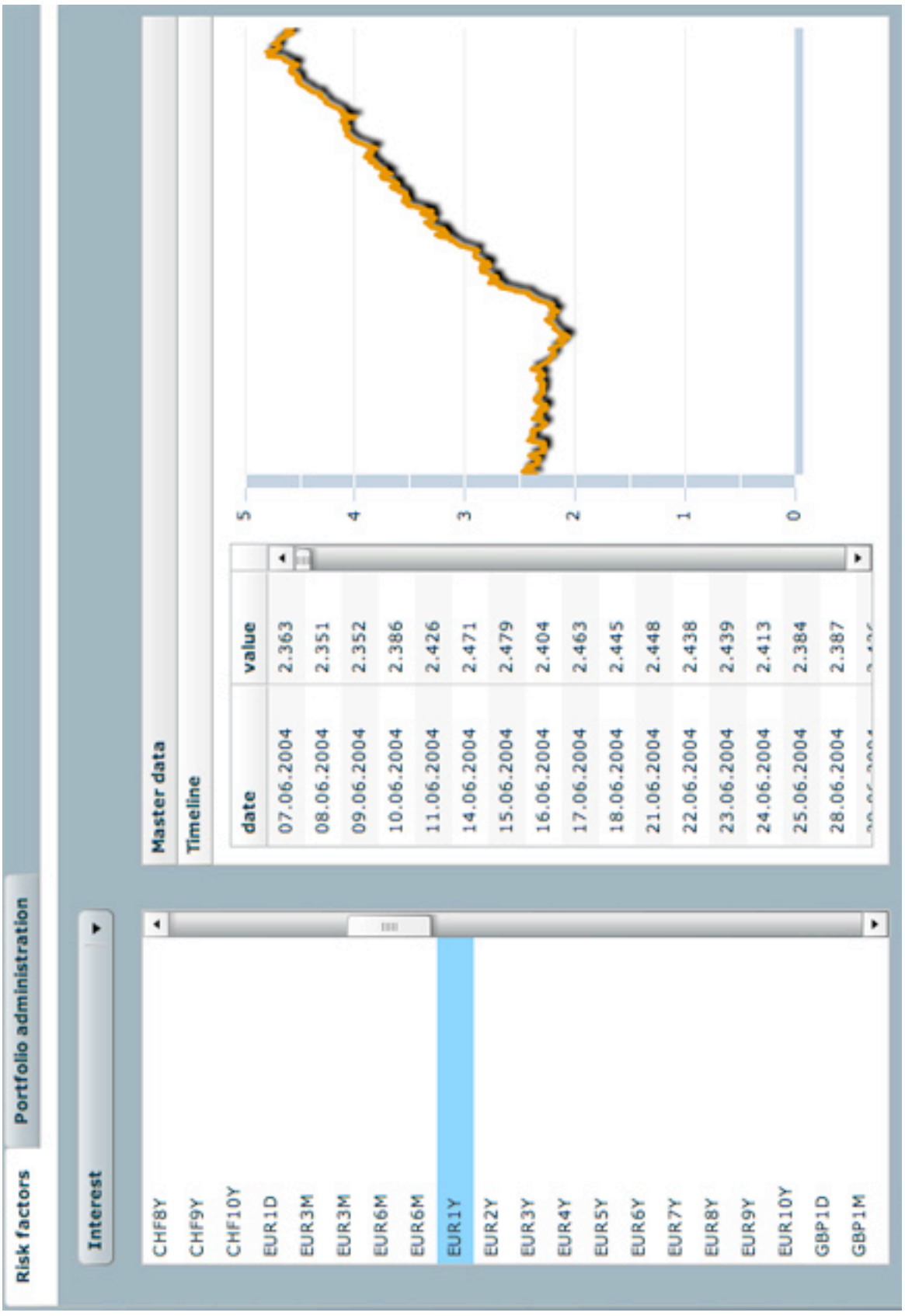

Figure 5.1: Screenshot information system - risk factors

Figure 5.2: Screenshot portfolio administration

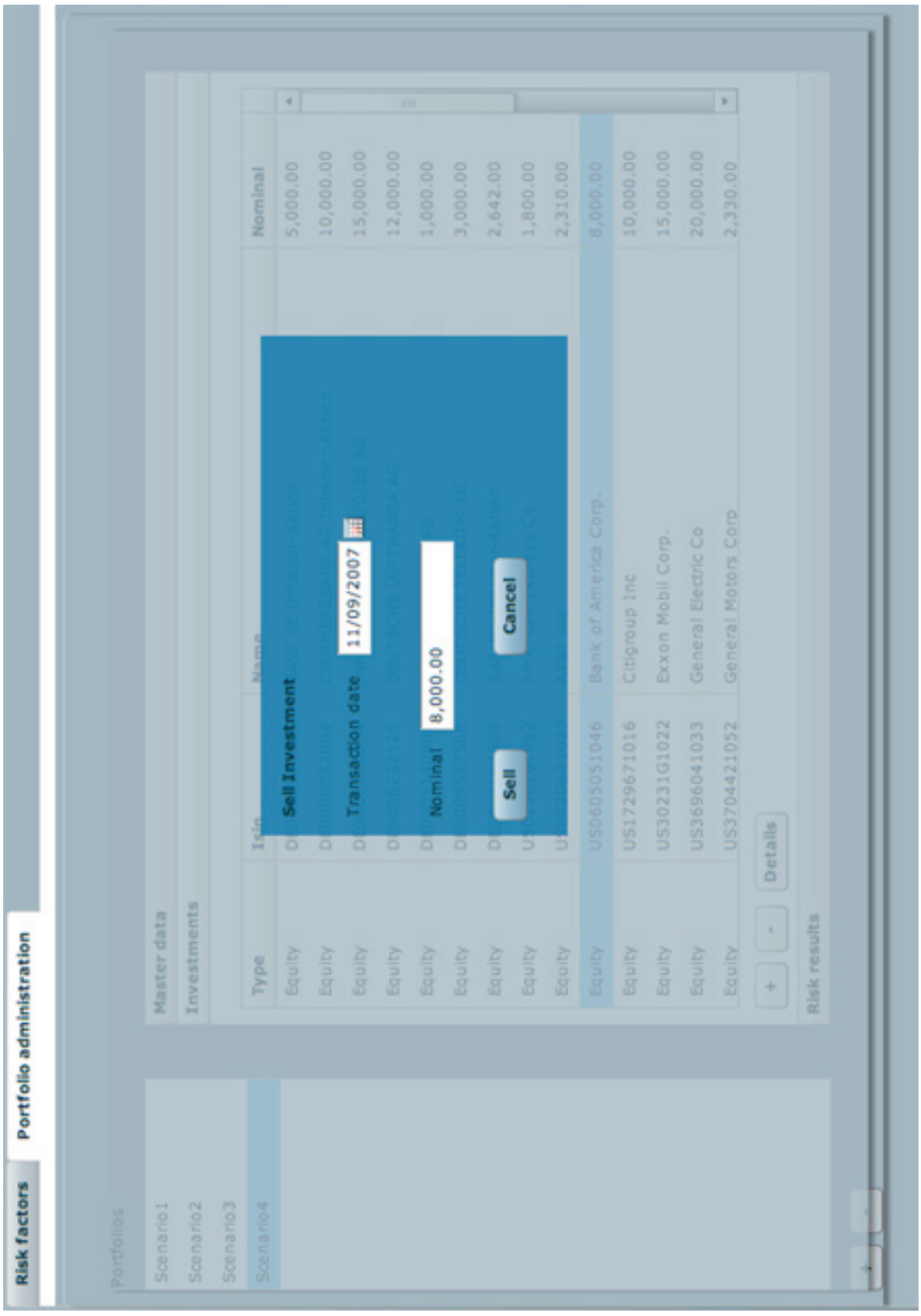

Figure 5.3: Screenshot transaction management

Figure 5.4: Screenshot presentation of the risk evaluation results

comparison of the different implementations. There exist already several papers, which compare the results of the delta-gamma versus the Monte-Carlo simulation approach, for example Mina and Ulmer in "Delta-Gamma Four Ways" [MU99]. Anyway, the comparison allows pointing out the pro and cons of each approach and offers the possibility to test the correctness of the implemented functions.

Therefore four test scenarios are defined. The first three scenarios cover a very simple portfolio with just one item, a stock option. The goal is to observe the value-at-risk of stock options in the following "states":

- out-of-the-money
- at-the-money
- in-the-money

The observed investment is the same in all three scenarios, an European stock option (long) on the BMW share. The evaluation date is November 9th 2007 (the last price of the underlying is 41.37 €), the expiration date in all cases is September 8th 2008. The strike prices are 45€, 41.37€ and 37€. The nominal amounts are 10,000. The prototype uses daily historical data (from 06/07/2004 - 11/09/2007). For calculating the variances and covariances the EWMA approach is used with a decay-factor of 0.94 (see [JPM96]). For every scenario the 95% value-at-risk is calculated with two different time horizons, 1 and 10 days. The reason is, that the results of both approaches should be similar for a short observation horizon. But there should be bigger differences over longer time periods. The Monte-Carlo simulation uses 100,000 simulation paths for each simulation.

The fourth case is a more realistic scenario with a portfolio containing a mix of stocks and stock options. The exact portfolio allocation is illustrated in Table 5.1.

Type	(Underl.) ISIN	Name	Curr.	Nom.
Stock	DE0005151005	BASF SE Inhaber-Aktien	EUR	5 000
Stock	DE0008032004	Commerzbank AG Inhaber-Aktien	EUR	10 000
Stock	DE0006231004	Infineon Technologies AG	EUR	15 000
Stock	DE0008232125	Deutsche Lufthansa AG	EUR	12 000
Stock	DE0005140008	Deutsche Bank AG	EUR	1 000
Stock	DE0005557508	Deutsche Telekom AG	EUR	3 000
Stock	DE0007164600	SAP AG Inhaber-Aktien	EUR	2 642
Stock	US0258161092	American Express Co.	USD	1 800
Stock	US00206R1023	AT&T Inc	USD	2 310
Stock	US0605051046	Bank of America Corp.	USD	8 000
Stock	US1729671016	Citigroup Inc	USD	10 000
Stock	US30231G1022	Exxon Mobil Corp.	USD	15 000
Stock	US3696041033	General Electric Co	USD	20 000
Stock	US3704421052	General Motors Corp	USD	2 330
Stock	US4581401001	Intel Corporation	USD	11 000
Stock	US46625H1005	JPMorgan	USD	233

Stock	US5949181045	Microsoft Corporation	USD	22 000
Stock	US7170811035	Pfizer Inc	USD	3 321
Stock	US92343V1044	Verizon Communications Inc	USD	4 500
Stock Opt.	DE0005151005	Call BASF 90 7/2008	EUR	10 000
Stock Opt.	DE0005140008	Call DB 78 5/2008	EUR	14 000
Stock Opt.	US0605051046	Call Bank of America 50 7/2008	USD	3 000
Stock Opt.	US3704421052	Call GM 30 11/2008	USD	6 000
Stock Opt.	US92343V1044	Call Verizon 41 4/2008	USD	8 000
Stock Opt.	US46625H1005	Call JPM 43 5/2008	USD	15 300

Table 5.1: Asset allocation - scenario 4

For every scenario, a new portfolio was created in the system. In these four portfolios the investments were added via the described web service. As already mentioned, the prototype has implemented the workflow (illustrated in Figure 3.5), which defines the behavior of the system after adding a new investment to the portfolio. The event, adding a new investment, triggers several other procedures. First, the system finds all missing evaluation dates (for instance if the investment was already bought on the 2nd November, but is inserted in the system right now on the 9th November). Next, all needed historical time series are downloaded from the system database and the risk of the investment is calculated with all registered approaches (this includes also the different time horizons). The received evaluation results trigger another procedure to reevaluate the entire portfolio. The tests show one weak point in the portfolio calculation service related to the Monte-Carlo simulation approach. This doesn't affect the correctness of the results, furthermore the performance of the system. The evaluation of the portfolio in scenario 4 (including 25 different investments) takes an immoderate amount of time. The responsible web service has the following workflow implemented. First, all affected risk factors are identified and the variance-covariance matrix is created. After that, all required data are transmitted to a MatLab procedure, which simulates the values and returning the matrix to the web service again. Then, the market value for each investment is calculated with the corresponded simulated data. The value-of-risk of the entire portfolio is equal to the difference between the actual market value of the portfolio and the sum of the simulated worst-case market values (with a confidence level of 95%) of all investments. The problem could be identified in the procedure which determines the alpha-quantile of the result-matrix. All results are added in a $SortedList$ to automatize the sortation of the values, but this element resorts the list after every new added item. That's the reason why the portfolio calculation takes such long time. There is a potential improvement in future versions.

The following table (Table 5.2) shows the results of all tested scenarios:

Scen.	Approach	VaR 1 day	VaR 10 days	MV	Delta	Gamma
Scen. 1	Delta-gamma	5 002.56	15 601.84	30 908.31	0.4639	0.04178
	MC Sim.	5 099.03	13 715.12	30 908.31	0.4639	0.04178
Scen. 2	Delta-gamma	6 564.11	20 657.04	46 467.44	0.6096	0.04230

	MC Sim.	6 697.80	18 458.71	46 467.44	0.6096	0.04230
Scen. 3	Delta-gamma	8 324.89	26 345.82	71 983.51	0.7732	0.05389
	MC Sim.	8 475.63	24 372.14	71 983.51	0.7732	0.05389
Scen. 4	Delta-gamma	151 091.85	477 320.34	4 546 916.62		
	MC Sim.	149 175.34	424 350.00	4 546 916.62		

Table 5.2: Scenario results

The first three scenarios include only one asset, an option on the BMW share with different strike prices. The results are very similar. As expected are the value-at-risk values of the delta-gamma approach for one trading day very close to the results of the Monte-Carlo simulation approach. The ten days observation horizon shows a different picture. The value-of-risk of the delta-gamma approach is much higher than the comparable results of the simulation approach. In general the delta-gamma approach tends to overrate the risk over a long-time period. The comparison of the larger portfolio in scenario 4 shows a similar result. The value-at-risk of the ten days horizon is 12 percentages higher calculated with the delta-gamma approach than with the simulation approach. This spread increases over time. The bottleneck in the Monte-Carlo simulation approach (besides the discussed weakness in the implementation of the prototype), is the portfolio evaluation. The calculation of portfolios with many assets takes a much longer time than the analytic approach. If the risk of a portfolio is calculated only once a day, the runtime behavior is no big problem. But if there is a continuous change in the portfolio during the day, every interaction triggers a reevaluation of the entire portfolio, which could be a challenge for the system infrastructure. The question of the better approach depends on the individual requirements and the available resources. The risk evaluation system can handle all methods. At the end it's a trade off between the precession of the results and the needed calculation time. But there is also wide scope in the workflow design. For example a compromise would be to calculate the analytic risk continuous and evaluate the risk with help of the simulation approach once a day.

6 Conclusion

The controlling of the market risk is an essential part for every financial institution. The risk evaluation consists of a huge amount of single calculations; therefore it's necessary to establish appropriated software applications. But such types of systems are very agile. It's necessary to add new functionality (e.g. calculation methods for new type of investments) or improvements of already existing functions all the time.

The objective of this work was to design a flexible risk measurement system. The term flexible doesn't limit to only easy improvement and extensibility, furthermore the system should run on several different operating systems and different databases as well. The work focused on two rather new software development processes, SOA (service oriented architecture) and EDA (event driven architecture), which offer a more flexible implementation than for example the object-oriented paradigm. Whereas EDA can be seen as an extension of SOA. The core elements of both technologies are services. These services represent an independent part of functionality, which allows an easy reusability and scalability (buzzword "loosely-coupled services" see Section 2.1.1). The main difference between SOA and EDA can be found in the orchestration of the services to form an entire application. SOA bases on a synchronal approach (after finishing service A, call service B, etc.). EDA uses events instead. Every service throws an event after finishing its procedure (the results are stored in messages). Every other method can subscribe these events. For example the service "$saveRiskResult$" has subscribed the events of all risk evaluation methods. After finishing a risk calculation, the results are saved in the database automatically.

The prototype of this work bases on EDA. Due to the complexity and size of such systems, it's impossible to implement a complete application in a reasonable time. Therefore the prototype covers only one part of such a system. The objective was to implement the workflow of Figure 3.5. It should be possible to evaluate the risk of stocks, stock options and entire portfolios with help of the delta-gamma approach as well as with the Monte-Carlo simulation approach.

An essential part of the whole system is the database. The used solution had to consider two major requirements: easy extensibility and no restrictions to a specific database vendor. A good technology that provides these functionalities is the Hibernate-framework (see Section 5.3.2). The exchange of the used database (Oracle, MS SQL Server, MySQL, etc.) only requires altering one line of code in the configuration file. Changes in the database table structure can be handled in a very fast and easy way, too, without changing the whole database-access components.

As already mentioned, the application consists of many independent services. This allows a good reusability and scalability (all services can be hosted on different servers). To show the flexibility of this approach, the prototype uses different techniques to implement such services (in this case Java and MS .net) and also different operating systems. The implementation of

the evaluation services has another special feature. It combines the typical object-oriented development with the advantages of using mathematical software frameworks (see Section 5.3.1). This combination allows a very fast and efficient development. But it has to be mentioned that the security of the system has to be checked before using it in a productive system, because of possible security lacks.

The search for the right EDA framework took a long time. The technology is rather new and there are only a few frameworks in the market with different concepts. Some frameworks focus on the database of the system, for instance if a new dataset is inserted in a database table an event is thrown. But this project bases on the service oriented approach with no database dependency. Anyway, at the end the Mule framework was chosen because of the integrated web service support and the platform independency (it is written in Java). Mule provides two ways to orchestrate the services:

1. in the configuration files
2. with help of Java classes

In practice, it's not easy to avoid a mix of these two methods. But in larger projects the chances are very high to lose track of all invocations. A good organization and documentation (which service has subscribed which other services and where is this behavior defined) is strongly recommended. But the Java classes have other tasks, too. First, it's necessary to define the structure of the in- and output parameter. Second, in many cases the result of a service is part of the input parameter of the next invoked service. The Java functions map the attributes of the different structures and generate the new messages. Last but not least the classes are used to provide all required data for the single services. The whole concept bases on independent loosely-coupled web services. This means the services have normally no access to the system database. Therefore the EDA-system has to transmit all system specific information to the services. The prototype identifies a design failure in this topic. All data-access methods are implemented as web services. In the design phase it was realized that these services couldn't be independent cause of the specific system database. The idea was that some of these methods also could be used in other projects (for example to get historical time series of risk factors). But the implementation of the services was very difficult and time-consuming. The main problem is the handling of complex data types in web services, especially due to the use of inheritances. For instance invoking the method $getInvestment$ delivers, depending on the kind of investment (stock options, bonds, etc.), a different result structure. But web services have the characteristic that all parameters are clearly defined in the corresponding WSDL-file, a dynamic adjustment of the structure of the parameter is not possible. Another disadvantage is the worse runtime behavior due to the numerous service invocations and large data size. A better solution would be to integrate the data-access components directly in the EDA-system, which allows an easier and faster access.

In literature you can very often find the argument that one of the biggest weaknesses of event driven systems is the bad debugging and error traceability. After implementing the prototype I can completely agree to this point. At the beginning it was impossible to find any of the occurred errors. Mule[1] shows very unclear error messages and marks uninvolved functions in

[1] version 1.4.3

many cases. It was necessary to build my own debug environment including network sniffer software (to analyze the transmitted data) and additional artificial web service gateways (to log the in- and output parameters), which improved the situation. Besides the weaknesses in the framework, the general behavior of an event-based system makes the debugging difficult. The occurrence of an event is not predictable; especially in more complex workflows this characteristic makes the identification of the source of error very difficult.

A very positive experience was the usage of the Flex framework[2] for implementing the web application. The development tool is the Adobe Flex builder[3], which is a plug-in in the Eclipse framework and is available for the most common operating systems. One of the features of Flex is the consequent separation of the presentation and business logic. The functionality can be integrated over different interfaces, for instance Java or MS .net, but there exists also a very good web service support. Another benefit is the multitude of already existing control elements. These components provide functionalities, which are very difficult or even impossible to implement in HTML (even with help of JavaScript). For instance, the illustration of the historical value-at-risk in a line chart has in total ten lines of code. Furthermore it is possible to animate the diagrams and react on user interactions (e.g. the selection of a specific date could trigger the display of a more detailed view). Another good example is the data grid element (for displaying tables). The input parameter is a simple array variable. Without any additional work it's possible to sort every column of the table. The provided features and the simple development allow a very comfortable development of powerful web applications.

The event driven architecture has in my opinion a high future potential. The approach bases on a complete new concept and it requires a different type of thinking (in comparison with the object-oriented programming paradigm), for software architects as well as for software developers. The process can be compared with the changeover from the procedural to the object oriented programming. The required time and costs of such change in technique shouldn't be underestimated. The usage of independent modules allows a quite good integration of legacy systems. Mule for instance provides an AS-400 interface, which allows a connection to existing mainframe systems. The prototype shows all advantages of an event-driven architecture. The independence of the modules was demonstrated with the implementation of different evaluation approaches (evaluate stocks and stock options with help of the delta-gamma and Monte-Carlo simulation approach). Each of these services can be used, parallel to the risk evaluation system, in any other application as well (for instance in an asset-allocation software etc.). The different services provide a good scalability, too. The services can be hosted on different servers, which can improve the runtime-behavior of the whole application dramatically (especially for modules with complex, time-consuming calculations). But the biggest benefit is the easy extensibility. The first iteration of the prototype had only implemented the delta-gamma approach. The integration of the new simulation services was very simple. The new modules had just been subscribed to the same events as the delta-gamma approach methods. This allows a good integration of new methods in existing systems. For example at first only the (easier to implement) delta-gamma approach is developed. After a time the

[2] version 2.0

[3] http://www.adobe.com/products/flex/features/flex_builder/

simulation approach is added and will work parallel with the delta-approach. The observation of the different results is a very simple and effective test method. After this test phase, the simulation approach could replace the delta-approach completely. The main problems today are the bad development frameworks, which imply longer development times and costs, and the low acceptance (or even knowledge) of the technique. If this situation changes a new era of fast, efficient and simple software development will start.

Bibliography

[BAW87] Giovanni BARONE-ADESI and Robert E. WHALEY. Efficient Analytic Approximation of American Option Values. *The Journal of Finance*, 42(2):301–320, 1987.

[BGL07] Kevin BIERHOFF, Mark GRECHANIK, and Edy S. LIONGOSARI. Architectural mismatch in service-oriented architectures. In *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, page 4, Washington, DC, USA, 2007. IEEE Computer Society.

[BJS99] Mark BRITTEN-JONES and Stephen M. SCHAEFER. Non-Linear Value-at-Risk. *European Finance Review*, (2):161–187, 1999.

[DOW98] Kevin DOWD. *Beyond Value At Risk*. Wiley & Sons, 2 edition, 1998.

[DW05] Vijay DHEAP and Paul A. S. WARD. Event-driven response architecture for event-based computing. In *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, pages 70–82. IBM Press, 2005.

[GBU07] Gerald C. GANNOD, Janet E. BURGE, and Susan D. URBAN. Issues in the design of flexible and dynamic service-oriented systems. In *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, page 8, Washington, DC, USA, 2007. IEEE Computer Society.

[GHS01] Paul GLASSERMAN, Philip HEIDELBERGER, and Perwez SHAHABUDDIN. Efficient Monte Carlo Methods for Value-at-Risk. *Mastering Risk*, 2:5–18, 2001.

[GLA03] Paul GLASSERMAN. *Monte Carlo methods in financial engineering*. Springer-Verlag, 2 edition, 2003.

[HAU04] Martin HAUGH. The Monte Carlo Framework, Examples from Finance and Generating Correlated Random Variables. "http://www.columbia.edu/~mh2078/MCS04/MCS_framework_FEegs.pdf" , 2004. [Last access: 11/09/2007].

[HH07] Wolfgang HACKENBROCH and Matthias HENNEBERGER. Service-Oriented Computing for Risk/Return Management. In *eOrganisation: Service-, Prozess-, Market-Engineering; 8. Internationale Tagung Wirtschaftsinformatik*, pages 495–512, 2007.

[HOH06] Gregor HOHPE. Programming Without a Call Stack - Event-driven Architectures. Technical report, www.eaipatterns.com, 2006.

[HUL05] John C. HULL. *Options, Futures and other Derivatives*. Pearson Studium, 6 edition, 2005.

[HW03] Gregor HOHPE and Bobby WOOLF. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 1 edition, 2003.

[JOR97] Philippe JORION. *Value at risk: the new benchmark for controlling market risk*. Irwin, 2 edition, 1997.

[JOR05] Philippe JORION. *Financial Risk Manager Handbook*. Wiley & Sons, 3 edition, 2005.

[JPM96] RiskMetrics - Technical Document. Technical Report Fourth Edition, J.P.Morgan/Reuters, 1996.

[LUC02] David LUCKHAM. *The power of events: an introduction to complex event processing in distributed enterprise systems*. Addison-Wesley, 3 edition, 2002.

[MA06] Tatiana MIAZHYNSKAIA and Wolfgang AUSSENEGG. Uncertainty in Value-at-Risk Estimates under Parametric and Non-parametric Modeling. *Financial Markets and Portfolio Management*, 20(3):243–264, 2006.

[MAT07] Mathworks: How do i call a matlab compiler 4.0 (r14) generated shared library from a c# application? http://www.mathworks.com/support/solutions/data/1-X1PFC.html, Februar 2007. [Last access: 12/10/2007].

[ME01] Simone MANGANELLI and Robert F. ENGLE. Value at risk models in finance. Technical Report WORKING PAPER NO.75, EUROPEAN CENTRAL BANK, 2001.

[MFP06] Gero MÜHL, Ludger FIEGE, and Peter R. PIETZUCH. *Distributed Event-Based Systems*. Springer-Verlag, 1 edition, 2006.

[MU99] Jorge MINA and Andrew ULMER. Delta-Gamma Four Ways. RiskMetrics Group, LLC, 1999.

[OAS07] Web services business process execution language version 2.0 primer. http://www.oasis-open.org/committees/download.php/23974/wsbpel-v2.0-primer.pdf, May 2007. [Last access: 08/28/2007].

[PS06] Keshav Pingali and Paul Stodghill. A distributed system based on web services for computational science simulations. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, pages 297–306, New York, NY, USA, 2006. ACM Press.

[RSS07] Szabolcs ROZSNAYAI, Josef SCHIEFER, and Alexander SCHATTEN. Concepts and models for typing events for event-based systems. In *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 62–70, New York, NY, USA, 2007. ACM Press.

[TIR04] Günter TIRLER. *Finanzsimulation unter besonderer Berücksichtigung der automatischen Zufallszahlenerzeugung*. PhD thesis, Wirtschaftsuniversität Wien, 2004.

[vH06] Jack van HOOF. How EDA extends SOA and why it is important. Technical report, soa-eda.blogspot.com, 2006.

[W3C99] W3c: Xml path language (xpath) version 1.0. http://www.w3.org/TR/1999/REC-xpath-19991116, November 1999. [Last access: 08/26/2007].

[W3C07] W3c: Web services description language (wsdl) version 2.0 part 0: Primer. http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/, June 2007. [Last access: 08/14/2007].

[WOL06] Sascha WOLTER. Blitz-Entwicklung - Flex 2: Flash für Programmierer. *Heise-Verlag c't*, pages 188–192, July 2006.