How to utilize the J2EE^{TM} Framework in Engineering Projects

by
Gerd Stange
University of Applied Sciences Kiel, Germany

Abstract

The specification of the *Java 2 Enterprise Edition (J2EE)* defines a complete platform for the comfortable development of web applications. It involves *Enterprise Java Beans (EJB), Servlets* and *Java Server Pages (JSP).* Both Servlets and JSP, also called web components, are powerful constructs for the dynamic generation of web content. They allow direct access to Java Enterprise Beans which exist in two different forms: *Session Beans* mainly serve management purposes thus outlining the business logic context of a session whereas each *Entity Bean* is related to a well defined business entity which in turn may be directly mapped to a database entry. Thus Session Beans typically deal with non persistent data whereas Entity Beans typically describe persistent data. As part of the J2EE specification the Java Enterprise Beans together with auxiliary support classes on the one hand and the web components on the other hand are to be deployed into so called containers each: Web components will be surrounded by the web container and Enterprise Java Beans will reside in the EJB container. The container specification takes responsibility for all the details of client server communication. It is this concept that allows the application developer to focus her attention on the application specific aspects of a software project.

Whereas most of the literature available on J2EE applications is dedicated to purely administrative tasks as they occur in business environments like e-banking, e-commerce etc. the present talk studies the usefulness of the J2EE framework for engineering consultants by way of an example. Typically applications in this field rely on heavy computing capabilities required on the server side, ideally to be accessed by a thin client. It will be shown that business objects may be easily extended to technical objects represented by Entity Beans and that the business logic may be easily extended to the complex technical logic underlying the solution domain to some technical problem. As an example the trajectory of an electrically charged particle inside a magnet, which will be specified by a thin client, will be calculated as the RUNGE-KUTTA solution of the underlying set of two coupled differential equations of second order. The result set will be dynamically generated by Java Server Pages (JSP) and thus sent back to the client. Similar solutions may be used in complex computer simulation environments employing the FINITE ELEMENT METHOD (FEM) or the BOUNDARY ELEMENT METHOD (BEM). It is obvious that the technical part may be elegantly coupled with a business part, the latter taking responsibility for customer authentication, charging for received services etc. As a result the J2EE framework turns out to be an ideal platform for complex consulting services offered by engineering consultants including the business organization.

The J2EE platform, an overview

The development of distributed applications in computer networks is a demanding task. Most of these applications are business applications. They follow a *3-tier client server architecture*. In the ideal case the remote client as the first layer should be a “thin” client, mainly responsible for the pure presentation of business data. The second layer located on a distant server should take care of the business logic. Finally the persistent business data entities reside on the third layer which is physically realized by a database. Such an architecture is extremely well suited for typical business demands such as scalability and ease of replacement of business logical and business entity units.

As Figure 1 illustrates, the J2EE platform has been designed according to this principal architecture following a component model with JSP Pages and Enterprise Beans as the most important components.

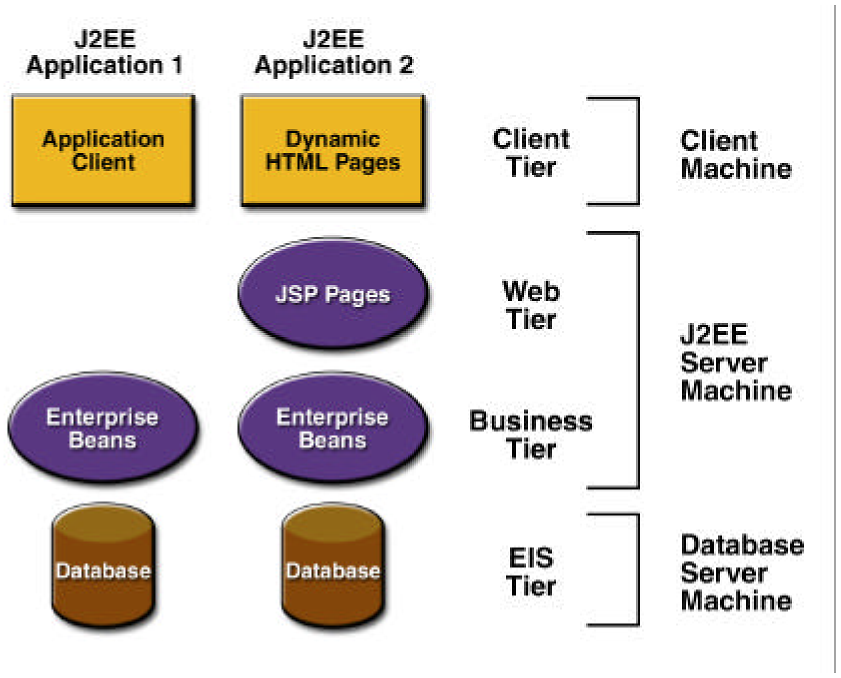


Figure 1: Multitiered Applications /J2EE^{TM} Tutorial/

One of the most important benefits of the J2EE platform is that all the technical details necessary for establishing and maintaining the communication link between client and server, for the safe treatment of transactions, for the reliable management of persistent data, for user authentication and for keeping track of concurrent operations etc. are under the responsibility of the component transaction monitor (CTM), which is part of its specification. The J2EE platform hides all these complex details from the application developer, thus allowing her to focus on the business specific aspects of the application.

The many technical services just mentioned are provided by the *J2EE server* with its underlying services in the form of a *container* for every component type. *Containers* are the interface between a component and the low-level platform-specific functionality that supports the component. Before a Web, enterprise bean, or application client component can be executed, it must be assembled into a J2EE application and deployed into its container.

The assembly process involves specifying container settings for each component in the J2EE application and for the J2EE application itself. Container settings customize the underlying support provided by the J2EE server, which includes services such as security, transaction management, Java Naming and Directory Interface (JNDI) lookups, and remote connectivity. Here are some of the highlights:

- The J2EE security model guarantees access by authorized users.
- The J2EE transaction model lets appear all methods in a transaction as a single unit.
- JNDI lookup services provide a unified interface to naming and directory services to look up remote components.
- Through he J2EE remote connectivity model a client invokes methods of a distant component as if it were in the same virtual machine.

The following Figure 2 illustrates the relationship between the different J2EE components and their corresponding containers (WEB- and EJB-Container) and how these containers are embedded in the J2EE Server.

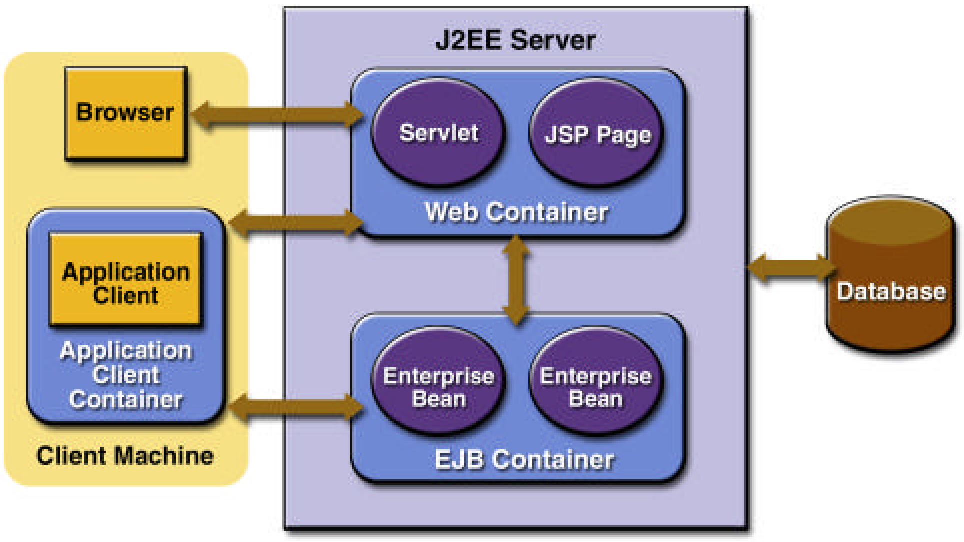


Figure 2: The J2EE Server and Container Concept /J2EE tutorial/

In the above figure it is important to note that the J2EE architecture includes access to the server resources by the traditional so called *Application Client* as well as by the modern concept of a “thin” client represented by a browser.

Java Enterprise Beans as the work horses

As mentioned above there are two types of Java Enterprise Beans:

- *Session* Beans stand for logical, business and session management with all the required methods.
- *Entity* Beans represent the (mostly persistent) business entities.

From the viewpoint of scalability and the ease of change it is very important to clearly distinguish between these two: Whereas *Session Beans* are responsible for the system *behavior* as a whole, *Entity Beans* represent system its *state.*

Enterprise Beans will never be directly accessed by clients or some other components (including other beans), which then play the role of clients again. Rather they are visible to the outside world only by their interfaces, which however are implemented by them. Every Enterprise Bean has two types of interfaces for each local and remote access. This results in a total of 4 interfaces:

- The HOME local/remote interface is responsible for life cycle events such as the creation or the destruction of the bean etc.
- The local/remote interface exposes the business logical content of the bean.

One may think of this concept as of a contract between the bean and its interface, allowing an easy replacement of the bean by another one in the case of change of the underlying business rules.

The J2EE platform for engineering projects

Most of the applications of the J2EE platform described in the literature cover the field of business applications like e-banking, e-commerce etc. There is no reason, why this same platform should not be similarly well suited for engineering applications. Obviously there is a strong analogy between business entities and engineering entities – this giving rise to the proper use of Entity Beans – and between business logic and engineering logic, which mainly consists of mathematical rules – this giving rise to the use of Session Beans. What is even more interesting: Both fields of application may be directly combined into one unique application with the result that engineering services offered by the engineering part of the system may be immediately integrated with the business part. The latter may take care of all kinds of customer services and volume dependent accounting. In the following section the *Magtrajectory* project will be presented as an example for the engineering aspects of a J2EE solution.

The Magtrajectory Project as an Example

To show the benefits of the J2EE platform for the solution of engineering problems the Magtrajectory project offers computer simulation services to a traditional application client and to a "thin" WEB client. The service is the solution of a set of coupled ordinary differential equations as they describe the trajectory of a charged particle inside a magnet. The client requests the user to enter a magnet Id or to enter her own magnet data and then run the application. The magnet data are persistently stored in a database table of the MagnetDB database.

The following figure 3 shows the UML use case diagram:

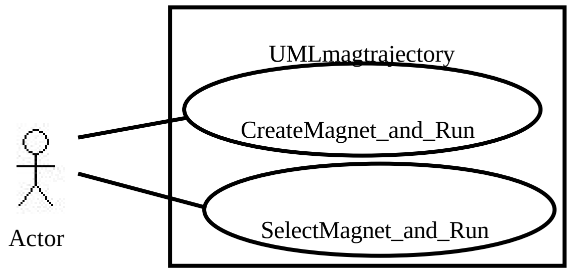


Figure 3: Use Case diagram of the Magtrajectory Application

The magnets to be selected or to be created are the engineering entity units. They are under the responsibility of the *MagnetsBean Entity Bean*. All the management and calculation steps necessary will be managed by the *MagtrajectoryBean Session Bean*, which in turn employs the following helper classes:

- *MagnetsBean* to select or create a Magnets object from/in the database.
- *Magnet* to create and hold a technical magnet object with the implemenation of a suitable interface
- *DiffEquation*, which describes the set of coupled differential equations.
- *StartPar* to describe the particle and its start conditions.
- *Trajectory* to describe the trajectory data and
- *RungeKutta* being responsible for the numerical calculation procedure.

The following figure 4 gives an overview over all these classes in an UML class diagram.

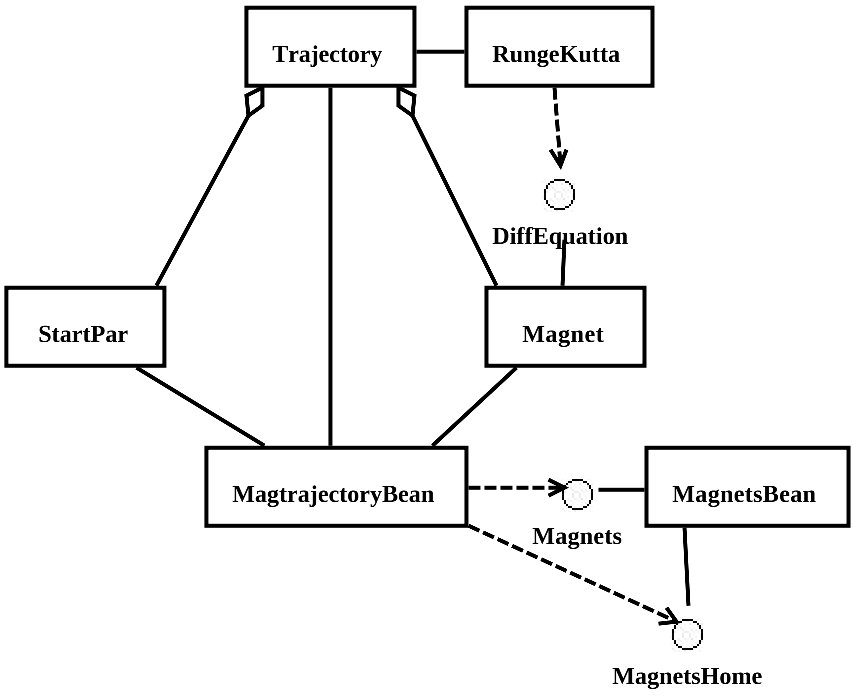


Figure 4: UML Class diagram of the Magtrajectory Application

It is obvious that the complete analysis of the Magtrajectory project requires an intimate knowledge of the underlying physics of relativistic particle dynamics inside magnetic fields and furthermore of all the detailed steps necessary in the numerical solution procedure of a coupled set of differential equations using the Runge-Kutta scheme. This is beyond the scope of the present report. Rather the emphasis lies on showing that the J2EE platform is well adapted to the needs of a distributed solution of fairly complex engineering problems. Therefore in the following figure 5 a rough UML collaboration diagram will be presented, which may be considered as a first outflow of the physical and numerical analysis mentioned above.

This figure clearly illustrates the dominating management role the MagtrajectoryBean plays in the Magtrajectory project.

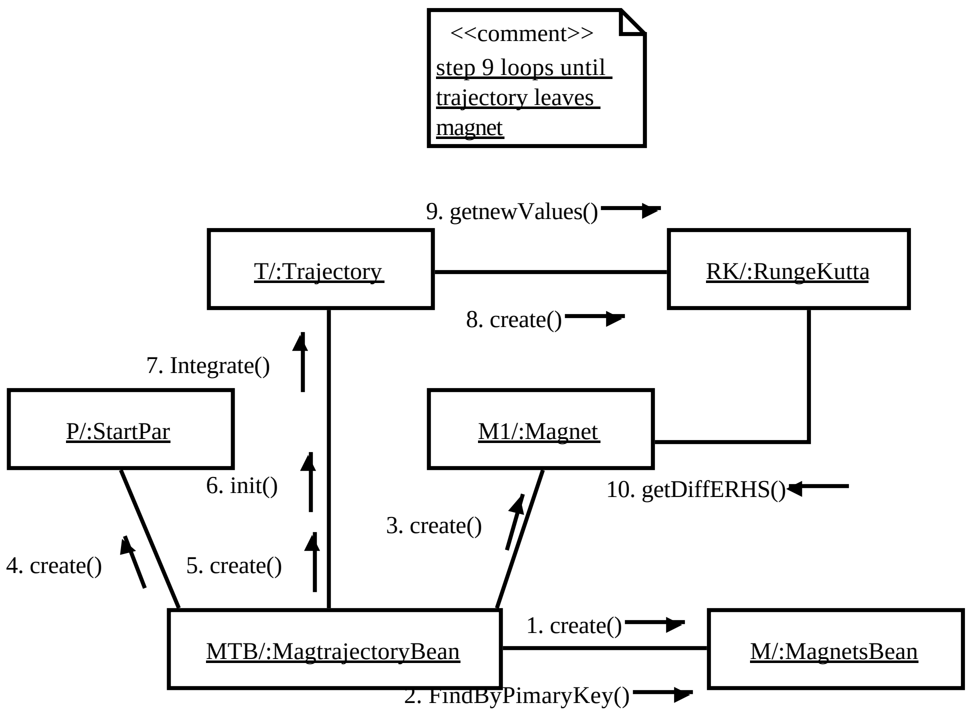


Figure 5: UML collaboration diagram of Magtrajectory project

All the above considerations show the strong parallelity between business and engineering projects. In both cases the Session Beans take responsibility of the system behavior as a whole, possibly employing a large number of helpers, whereas the Entity Beans take care of the entities in the corresponding context.

To run the system as a client server system the software has to be properly deployed into the server and client containers mentioned above. The following UML deployment diagram figure 6 shows the details.

The Beans are located in the Server/EJBContainer, the JSP Server Pages have been deployed into the Server/WEBContainer. There are two sorts of Clients: The traditional Application Client accesses the Server via a LAN- / WLAN- / WAN- link and the interfaces of the *MagtrajectoryBean*, whereas the WEB Client connects to the server in a completely different way: First the corresponding browser downloads the *index.jsp* Java Server Page from the WEBContainer, which is located on the server, via the Internet link and will then be connected to the *MagtrajectoryBean* interfaces via the the *runtrajectory.jsp* Java Server Page after the user has pressed the *runtrajectory* button displayed on the *index.jsp* page.

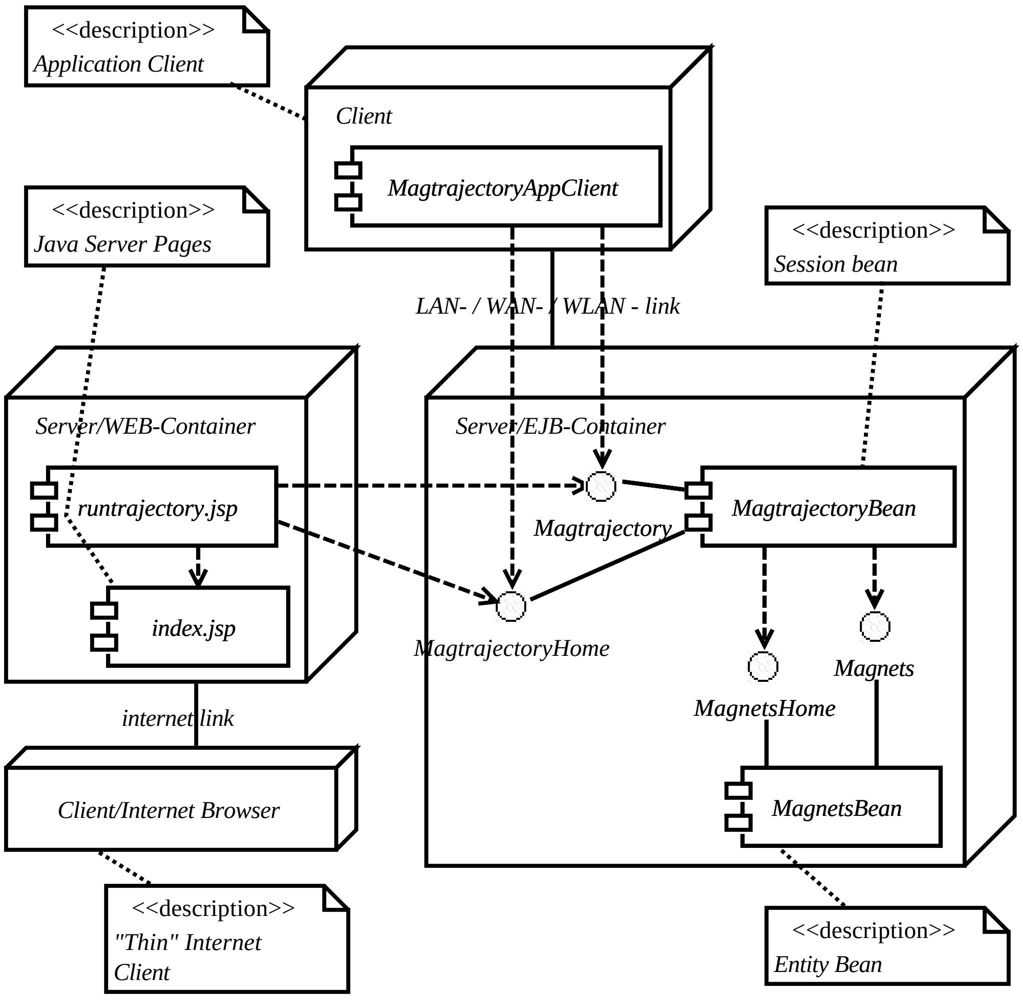


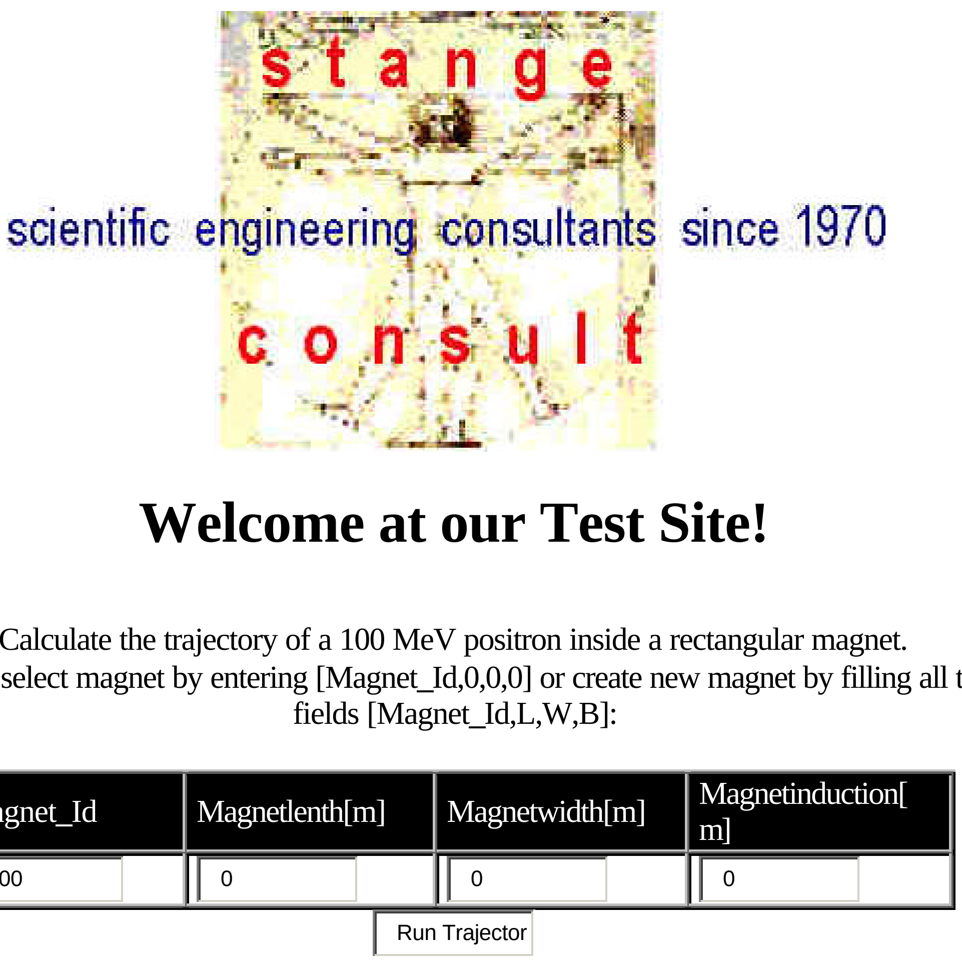
Figure 6: UML deployment diagram of Magtrajectory Project

It requires mention that the JSP Server Pages participate in the overall scenario in a very restricted way: They are solely responsible for presentation and for receiving some simple input data, all the computational work being left to the server. Especially for heavy weight engineering projects as e.g. FEM- or BEM- computer simulations this concept brings remarkable benefits: The resources and competences may be concentrated on well equipped servers, which may be accessed by lightweigt clients from everywhere.

Sample Output from the Magtrajectory Project

The following figures illustrate some sample output from the Magtrajectory System. The first figure 7 shows the start screen as delivered from the *index.jsp* Java Server Page, where the user is requested to either select a magnet by entering the magnet_Id or to enter new magnet data. Here the magnet_Id 100 has been entered with all the remaining fields set to 0. This means that the magnet with the magnet_Id = 100 must have been previously entered into the database.

When pressing the *Run Trajectory* button control will be transferred to the *runtrajectory.jsp* Java Server Page. This in turn will connect to the MagtrajectoryBean interface and hand over the input data from the previous page.



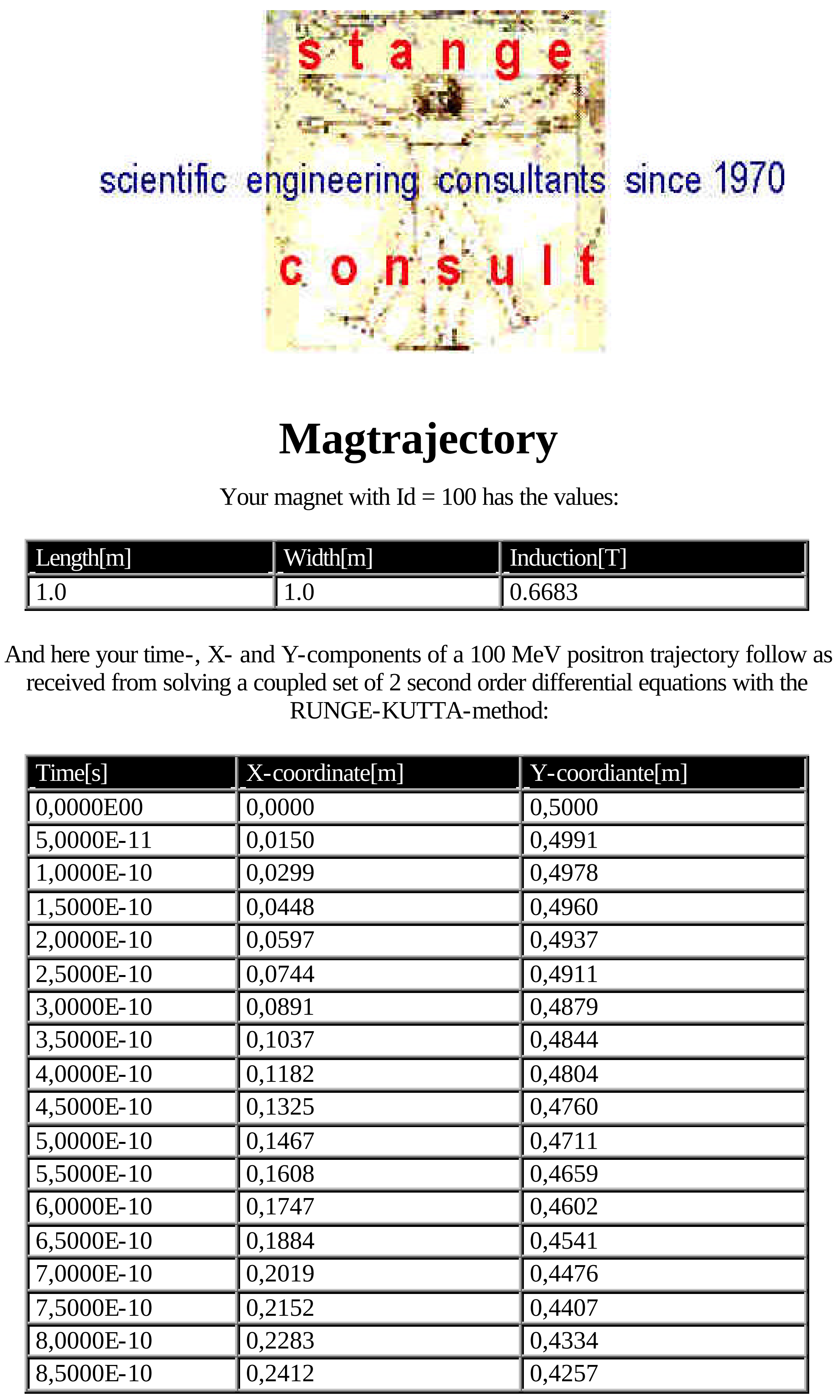
Welcome at our Test Site!

Calculate the trajectory of a 100 MeV positron inside a rectangular magnet.
Please select magnet by entering [Magnet_Id,0,0,0] or create new magnet by filling all the fields [Magnet_Id,L,W,B]:

Magnet_Id	Magnetlenth[m]	Magnetwidth[m]	Magnetinduction[m]
100	0	0	0

Figure 7: Browser Start Page of Magtrajectory Project

The next figure 8 shows the response after having pressed the *Run Trajectory* button.



s t a n g e

scientific engineering consultants since 1970

c o n s u l t

Magtrajectory

Your magnet with Id = 100 has the values:

Length[m]	Width[m]	Induction[T]
1.0	1.0	0.6683

And here your time-, X- and Y-components of a 100 MeV positron trajectory follow as received from solving a coupled set of 2 second order differential equations with the RUNGE-KUTTA-method:

Time[s]	X-coordinate[m]	Y-coordiante[m]
0,0000E00	0,0000	0,5000
5,0000E-11	0,0150	0,4991
1,0000E-10	0,0299	0,4978
1,5000E-10	0,0448	0,4960
2,0000E-10	0,0597	0,4937
2,5000E-10	0,0744	0,4911
3,0000E-10	0,0891	0,4879
3,5000E-10	0,1037	0,4844
4,0000E-10	0,1182	0,4804
4,5000E-10	0,1325	0,4760
5,0000E-10	0,1467	0,4711
5,5000E-10	0,1608	0,4659
6,0000E-10	0,1747	0,4602
6,5000E-10	0,1884	0,4541
7,0000E-10	0,2019	0,4476
7,5000E-10	0,2152	0,4407
8,0000E-10	0,2283	0,4334
8,5000E-10	0,2412	0,4257

Figure 8: Browser Result Page of Magtrajectory Project

Conclusions

In the present paper the possibilities of applying the J2EE framework to engineering projects have been studied by way of the example application *Magtrajectory*. After a brief introduction into the J2EE platform the *Magtrajectory* project has been introduced by a series of UML diagrams. Magtrajectory calculates the trajectory of a charged particle inside a magnet by numerically solving a set of coupled ordinary differential equations using the Runge-Kutta scheme.

It turns out that all the benefits of the J2EE framework applying to business applications are equally valid in the context of engineering applications. These benefits are to be seen in hiding all the technical details of a distributed application from the application developer. The combination of the Session Bean *MagtrajectoryBean*, which is responsible for all the management and calculational work, with the Entity Bean *MagnetsBean*, which is responsible for keeping track of the engineering entities *magnets* is as straightforward as in business applications. This concept guarantees for scalability of applications as well as for the ease of replacement of components.

Future development will be focused on the combination of engineering aspects and business aspects, the latter including such topics like automatic load dependent accounting, authentication and user access control due to different user roles There is no doubt that this combination will be an ideal base for consultance work in engineering since besides the demanding engineering services all kinds of customer services may be included.

References

/J2EE Tutorial/	Java Sun Online Tutorial
/Monson-Haefel 2003/ Monson-Haefel, Richard	Enterprise Java Beans, 3. Ed., O'Reilly, Cambridge 2003
/Farley e.a. 2003/ Farley, J. e.a.	Java Enterprise in a Nutshell, 2. Ed. O'Reilly, Cambridge 2003
/Stange 2003/ Stange G.	Lecture notes and several internal notes, source programs and deployed programs