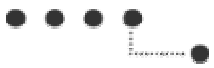


saces

Simple Artificial Chemistry Experiment System

Project Handbook

Anthony Aguillon, Daniel Noelpp isbe2005

Contents

1 PREFACE 3

2 PURPOSE OF THE PROJECT HANDBOOK 3

3 OVERVIEW / PROJECT DESCRIPTION 4

3.1 CURRENT STATE OF ART 4
3.2 PROJECT VISION AND TARGETS 5
3.3 STRATEGY 6

4 PROJECT SPECIFIC STRATEGIC PLANNING 7

4.1 PAKAGES 7
4.2 THE RESEARCH PACKAGE 7
4.3 THE PROTOTYPE PACKAGE 7
4.4 PACKAGE OVERVIEW 8
4.5 TIMETABLE 8
4.6 PROJECT PACKAGE RESULTS 9

5 METHODS AND TOOLS 11

5.1 TECHNOLOGIES 11
5.1.1 Java 1.4 or Java 5 11
5.1.2 SWT 3.1 11
5.1.3 JOGL 11
5.1.4 GCC, gcj, JNI and/or CNI 12
5.2 TOOLS 12
5.2.1 IntelliJ IDEA 4.5 12
5.2.2 CVS (Concurrent Versioning System) 12
5.2.3 Open Office 12
5.2.4 JProfiler 12
5.2.5 JUnit 12

6 STANDARDS AND GUIDELINES 13

6.1 CODING STANDARDS 13
6.2 DOCUMENTATION STANDARDS 13
6.3 TEMPLATES 13

7 ORGANISATION 13

7.1 TEAM 14

1 Preface

This document should serve as a project outline. It should not however, set any barriers for the implementing team when it comes to experimenting with ideas or excursing into other project-related areas.

Due to the academic nature of the project as suggested by the project proposal, many of a market-oriented project's characteristics need not be addressed here. If however, it is in the interest of management, that all Hermes aspects are described in this project handbook, the missing content can be added as the project progresses.

For a clearer understanding of this project, it is recommended that the reader first refer to the project proposal. The project proposal is detailed enough to serve as specification and will be the basis for most project documentation including this handbook. The technical documentation will contain the actual implementation details.

2 Purpose of a Project Handbook

A Project handbook is a very handy thing. Not only does it strengthen the understanding of the project's overall philosophy for all members of the project team, it cements the basic development rules and methodology required for the project's success.

A project handbook can be compared to a map of the project. The project has a starting point and a final destination, both locations on the map. Different routes lead from the starting point to the final destination. Some roads are paved, some are dirt, some are uphill, and some lead to nowhere. The purpose of the map, in analogy to the handbook, is to provide the traveller with an overview of what she can do, to reach her destination.

Depending on the amount of time and resources the traveller has to her disposal to complete the journey; she may want to pass by other interesting places on her way. In these places, the traveller may learn things, which improve the quality of the trip and therefore the quality of the project's result.

This project handbook follows the HERMES model. Although the authors have tried to stick to the HERMES structure, it has remained a priority, to write only useful documents. For this reason, only the project relevant HERMES aspects will be addressed in this handbook. A project handbook is useful, when it can convey the project's essence, motivation and development methodology to readers not especially involved in the project. To the development team, the handbook's value is reflected in the clarity by which the project is executed.

The project handbook is therefore the project's metadata in the form of a bare essential, easily read, project survival manual.

3 Overview / Project description

In the eighth semester of computer science studies at the engineering school of Bern, students are required to submit a final year project. The project usually encompasses executable software, its source, manuals, design specs and HERMES style documents including this project handbook. The software specifications are not always provided at the beginning of the project and when they are provided, they may be subject to change as the project progresses. This does not especially apply to our project due to the precise and detailed project proposal presented to us by Dr. Hinze.

The final year project consists of two parts. We'll call the first, a project work, the second, a diploma work. The project work (pw) is related to the diploma work (dw) in the sense, that it comprises of preliminary studies and preparation pertaining directly to the goals of the dw. During pw, the team will do research on Artificial and Molecular Chemistry, Thermodynamics basics, and the technologies required to finally implement the dw. The pw ends in July 2005 with the submission of the pw packages and documents. The packages and documents are specified in this handbook. The dw begins immediately after the pw and will last until the end of 2006. The exact project deadlines and milestones are listed in the packages chapter.

The pw and the dw can be viewed as one project although they are administratively divided into two projects. The division is mainly on the project management and organisational level.

Saces:
Saces is the working name for the simulation and stands for **S**imple **A**rtificial **C**hemistry **E**xperiment **S**ystem. It will visually simulate particle interaction according to user-defined rules. Within the simulation, a particle takes the form of a small sphere of a certain colour also defined by the user. The particle is placed together with other particles in a three dimensional space.

Input dialogs and data specifications can be found in the project proposal. The overall graphical representation however, has only been loosely specified and will take shape as the application matures. This will allow the team to validate design concepts by building various prototypes.

From the project management standpoint, Saces can be viewed as relatively easy to manage. The project team consists of only two students, one of which has been appointed project leader. Although this does not reflect an unbalanced distribution of responsibilities between the developers, the leader will tend to write more emails than his co-worker. In other words, both students are equally responsible for the project as a whole.

3.1 Current state of art

Again, the authors would like to refer to the project proposal delivered to us by Dr. Hinze. The introduction of the proposal contains useful information pertaining to the current or initial state.

It is important that the reader takes the following fact into account: Final year projects at the engineering school of Bern, do not usually yield professional tools which change the world. Writing professional software requires noticeably larger efforts and resources. The projects should however, contain some novel quality to them, not seen in previous developments. This can, as the reader has probably noticed, be very subjective. In information technology, as in most other places, there are seldom innovations which do not build upon existing solutions.

It is for the reasons stated above, that the implementation of an innovative, professional application, which will be used industrially, is clearly not a Saces objective. One concern that reinforces this notion is the fact that the Saces application is not specified to simulate particles on a quantum level. The Saces simulation is based on the simpler model of classical physics. This of course is a trade-off of realism for simplicity, which can prove to be advantageous in some applications. It is because of this simplification, that Saces is not comparable to other professional simulation applications in molecular chemistry. The Saces application will therefore not try to compete with industry tools, but rather, serve as a lecture and lab program, where Dr.Hinze finds appropriate.

Again, the Saces project is more of an academic one, compared to other final year projects. This quality is more visible in the next chapter, project visions and targets. But before we move on to the main Saces objectives, the reader is encouraged to take a look at the following documents listed below.

Document	Description
„Rechnen mit DNA“	Book by Dr.Hinze, ISBN:3-486-27530-5
http://www.tomilab.net/alife/index.html.en	Artificial life and related topics
http://www.nis.atr.jp/~hsuzuki/confs/2005_ECAL-WS-ACA.html	Workshop on Artificial Chemistry and its Applications
http://www.molpro.net/	The molpro quantum chemistry package
http://www.bioinf.uni-leipzig.de/software/ToyChem/cca.pdf	The ToyChem project

Search keywords: *“artificial chemistry”, “computational chemistry”, “artificial life”, “dna computation”, “molecular chemistry”*

The documents specified above should give the reader an overview, of what is currently being done in the fields of artificial and computational chemistry, artificial life and molecular chemistry (all fields relating in some way or another to the Saces application). The Saces team’s main reference however, will remain the book written by Dr.Hinze. For further reading on the current state of the arts, please consult the main Saces project evaluation document. It is the document we have to submit end of June 2005 for the project work.

3.2 Project vision and targets

The main project objective is the successful implementation and documentation of the java application specified by Dr.Hinze in the project proposal. This pertains to the diploma work (dw) as well as to the preliminary research and documentation in the project work or pw. The Saces team defines a successful implementation as the following: The Saces application will conform in all aspects to the project proposal defined by Dr.Hinze. This includes an ergonomic user interface, a graphical simulation which performs at an acceptable frame rate, data consistency and so on. The Saces target is therefore a simple one: to write quality software according to a detailed project proposal. This of course is only the tip of the iceberg. Below the surface lies the research and prototyping required to understand the subject matter and the technologies required to reach the target, most of which will be done during the project work. The following checklist is a summary of the targets the Saces team considers as vital to the projects success:

- Submit quality software according to the project proposal by the due date
- Submit easy to read, precise and informative documentation by the due date
- Develop an understanding of the molecular chemistry required to implement Saces
- Usage of the application for learning/ teaching purposes at the Technical College, Dresden.
- Plan effectively and adaptively.

Artificial and Molecular Chemistry are large scientific areas. Learning more about these fields will tempt the Saces team to envision application features which may be beyond the scope of our humble project. It is due to the lack of resources, that these visions, although maybe relevant to future projects, will not necessarily be encouraged. If there is still time towards the end of the year, these ideas can be discussed and tested appropriately.

3.3 Strategy

Software developers have a lot in common with one another. Nevertheless, it is still impossible to generalize the personal qualities of a developer. Take working hours as a simple example. Some developers find themselves at the office, every day at eight o'clock; others consider themselves most productive during the night, when undisturbed and alone. Although this example may seem too obvious to even been mentioned here, it represents a palette of personal traits, which play a major role in the success of a project. Forcing the latter developer to be at the office every day at eight o'clock might cause her to sleep less well, lose motivation and become less efficient. This of course can completely destroy a project, especially if the team consists of only two developers.

Learning more about the developers involved in a project can help in other important areas as well, such as communication, pair building, package responsibilities, etc.

As the reader probably has noticed, the Saces team has chosen to apply various eXtreme programming (xp for short) techniques to the project. Although xp programming tends to dictate a sort of radical all-or-nothing approach when applied to projects, some practices such as pair programming will be welcomed but not forced. For now, we'll elaborate on the techniques we will use, rather than what we will not. If the reader is not familiar with the eXtreme programming paradigm, Don Wells offers a gentle introduction to xp at www.extremeprogramming.org.

eXtreme programming works with short development iteration cycles. A cycle consists of the following events:

1. Selecting a story card (the xp version of a uml use case),
2. Implementing the idea expressed in the story card,
3. Writing and performing unit tests
4. Integrating, integration testing
5. Function tests performed by user

Once coding has begun, the Saces team will adapt a two-week cycle similar to the one described above. As the application begins to take shape, Dr.Hinze will be able to test newly added functionality on a two to three week basis. All of the projects documents and source code will be available for download. Since story cards are very similar to use cases, it is possible to take the use cases defined by Dr.Hinze, factor them to a finer grain, and build packages. Each code package therefore co-responds directly to a use case. This is how the code packages will be defined in the next project stage. From the project management perspective, a package is defined by priority and dependency. These parameters are responsible for the position a package may occupy in the timetable and which packages need to be handled first.

Automatic unit testing with JUnit will be included if code becomes complicated. More often than not, the tests will focus on interfaces rather than on classes. An automatic integration test can be written and later serve as an overall system test when the application is completed.

Another xp paradigm is refactoring (merciless). The Saces team considers this important, yet not possible to the extent proposed by xp. Refactoring can be costly in some cases considering the project deadline and available resources.

To sum it all up, xp or agile methods are a legitimate alternative to the usual, more formalized v-model or Hermes approaches. Xp does however, suggest methods which usually cannot be integrated into a project especially if the project is already underway. Methods such as pair programming are not yet a widely accepted practice and most programmers still do not appreciate another programmer watching her every keystroke (although it has been proven to be effective). The Saces team will practice xp where and when possible as long as the techniques stay pragmatic in their specific project contexts.

4 Project specific strategic planning

4.1 Packages

A package is a set of related tasks, which together with other packages, form the project's workload.

Before we can begin assigning tasks or packages to the developers, we must first define them. A package should be relatively small. This will encourage a two-week development cycle and integration testing. A package may also contain a prototype of some sort, documentation or research. Some packages can be worked on in parallel and some require a sequential workflow.

A developer will be able to pick out a package and work on it independently or with the other developer. Before integrating the package (whether documentation or code) into the application, it must be reviewed by another developer. This will insure continuous quality management on the module level. During package integration, if the package is a code package or class, it will undergo integration and function tests. Aside from code and documentation, a package can contain non-productive tasks such as research or reading.

4.2 The Research package

The research of package P1 will be on the lines of Artificial Chemistry, Thermodynamics and Molecular Chemistry. This encompasses a couple of chapters from the book written by Dr.Hinze, among others. A bibliography will be included in the project evaluation document. Aside from the books, the developers will consult the internet for further reading, again all sources will be listed in the appendix of the project evaluation document. Again the developers would like to not specify the actual content of the research in more detail.

4.3 The Prototype package

3D animations with java are not unusual. The more processor power, resources or data an animation requires, the more it becomes necessary to introduce more powerful components into the implementation. These mechanisms mainly concern memory io and the graphics engine.

Increasing performance usually means choosing a more direct approach in the implementation. In C++, programming an assembly block into the C++ code does this. Although not common now a days, this example should illustrates the performance trade-off when abstracting away from the hardware. Java's byte code needs to be interpreted by the vm before it can be executed. This is the trade-off in java, which calls for a native implementation for the critical components.

In preparation for the actual program implementation, we have decided to build a prototype. The prototype will prove or destroy two concepts, which play key roles in the choice of technology to be used for the final implementation. These concepts deal with issues in compatibility and performance.

The Saces team considers one java specific aspect to be vital to the program functionality: Namely, 3D Simulation performance. Prototyping will unfold this aspect and shed some light on the

performance limitations we will be confronted with when designing the final application. The prototype will also test compatibility between **J**ava **O**pen **G**raphics **L**anguage and **S**tandard **W**indow **T**oolkit.

For the reader not familiar with either JOGL or SWT, here is a short description of both.

JOGL is a java library which encapsulates OpenGL, a widely used library for 3D graphics and animation (it is also used by Dooms rendering engine) and is usually interfaced from C or C++.

SWT is a library consisting mainly of graphical user interface components (Buttons and so on). It is the library used by Eclipse, the popular ide. Basically, SWT makes the user interface look native. This means, the user interface has the same look and feel as the rest of the operating system.

The prototype will therefore consist of a simple window or frame implemented using SWT. Once this has been built, the JOGL simulation panel or canvas will be built into the frame. If this is done successfully, then compatibility between the two technologies is guaranteed and we can move on to stress testing. Stress testing in our case would mean adding simulation logic into the prototype and animating a large quantity of objects. The prototype would test for collisions and perform the most costly (time complex) operations on the objects. The Objects could be the spherical particles we intend to use for the final representation. If JOGL turns out to not be compatible with SWT, this will not be considered a major setback. Instead of SWT, Swing and AWT can be used.

The reader might be asking herself, why the saces team has not mentioned java3D in the context of this prototype. Java3D appears to be no longer supported by Sun. It also has a complicated interface. OpenGL on the other hand, is easy to understand. Although neither one of the developers have worked with javaOpenGL, we are certain that basic knowledge of OpenGL will be very useful.

4.4 Package overview

The table below illustrates the packages the Saces team have identified, together with a short description. The reader should note that the packages might (and sometimes must) be shared between the developers to insure a successful package completion.

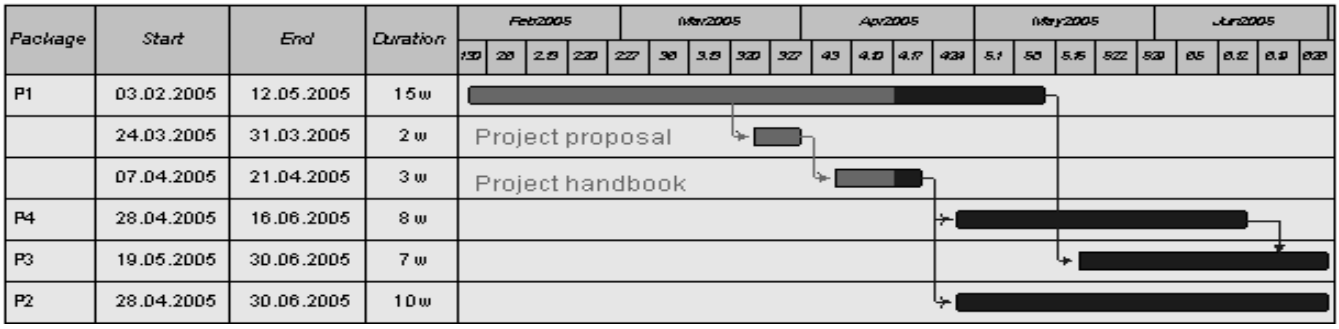
Package	Task	Dependency	Rough size in hours for 1 developer / actual time spend in hours[*]
P1	Research on Artificial Chemistry	No dependency	**100** /
P1	Research on Thermodynamics and Molecular Chemistry	No dependency	**80** /
P2	Write report based on research done in P1	P1, parallel	**140** /
P3	Write a project evaluation	P1. parallel, P4	**100** /
P4	Prototype	No dependency	**115** /

4.5 Timetable

The following Gantt diagram illustrates the size of the packages defined in chapter 4. The project proposal and handbook are in progress at this time. They therefore were not considered when the packages were defined. The duration of each package is defined in weeks. The Saces team works on the package at an average of one and a half to two days a week[*]. A working day is an eight-hour day. Thus, the developers work approximately twelve hours a week on a package. This of course is a rough estimation. The developers tend to invest a little more than this. Larger packages can still be broken down into smaller ones, although this has not been seen as necessary at this point in the project. The prototype package (P4) for instance, can be factored into programming modules. We

[*] Note however that the times given here cannot be taken as is. The time a student needs to complete a subject is measured in ECTS (European Credit Transfer System) and not in hours. But it is virtually impossible to give a rough conversion factor because we are working students, that means we have a job and are studying in our free time. There are official conversion factors but they have turned out to be just theory.

have decided to keep this open and document the task's details as we go along. Overly specifying a package would do more damage than good, considering the fact that packages P1, P2, and P3 are directly influenced by the research done in package P1. As the reader can see, P1 is a larger package which finds its end only towards the middle or end of May. Therefore it is difficult, and somewhat pointless to try to plan the content of P2, P3 and P4 in detail.



As mentioned above, a working week is at most a two-day week. A working day is an eight-hour day. This leads us to the calculation used for the actual package duration in hours specified in the packages chapter. At this point, the reader should be reminded, that the timetable serves as a type of task list. The "what" is more important than the "when", except for the final deadline that is. The final deadline, which will be set to the beginning of July, is our only milestone. Adding synthesised milestones to the already short project timetable seems useless at this time. Below is a copy of the table found in the Packages chapter. Together with the timetable, it should provide the reader with an idea of the packages extent and duration. The actual time spent on a package will be filled in when the package is completed.

Now that we have defined the "what", let us take a look at the "how". The next chapter describes the project's development philosophy and tries to convey the essence of how the Saces team would like to structure the various tasks. Although a very small team, it remains necessary to determine what is needed to insure that the individual developers are comfortable with the development cycles. If a developer is not comfortable, she will not be efficient. If she is not efficient, the project suffers. This is why the development strategy is of such importance and should be tailored, if possible to the general mentality and attitude of the individual developers.

4.6 Project package Results

The Package Result chapter is probably the most important chapter in this handbook because it describes the projects expected results. Every team member is responsible for something; they wouldn't be on the team if this were otherwise. These individual responsibilities are easily deduced from the team member's project role, specified in the <Team> chapter. Internal project responsibilities, such as code responsibility, do not exist when practicing xp. It is therefore safe to say, that for the students, everybody is responsible for everything.

Let us take a look at what the students are required to submit. The following project results are ordered chronologically, according to submission date. The light yellow field indicates the scope of the final year project.

Package	Result	Form / Description	Submission date
Submitted	Project proposal	Document / Project proposal provided by Dr.Hinze	
Submission pending	Project handbook	Document	

Package	Result	Form / Description	Submission date
P2	Project report	Document / Write CSOA based on research done in P1	First half of July
P3	Project evaluation	Document / Evaluates the progress status and course of the project.	First half of July
P4	Prototype	Documented Java source code / tests concepts which apply to the building of the actual application described in the project proposal	First half of July
To be defined	Executable java application	Java source code / as specified in the project proposal provided by Dr. Hinze.	
To be defined	Source documentation	Javadoc or similar html documentation	
To be defined	Design specifications	UML diagrams and design related documents	
To be defined	Users manual	html document	

The reader should note, that the research and study required to understand and implement the program specified by Dr.Hinze, is not included as a result. The research itself has been assigned a total of 80 hours* per student, as defined in the <Packages> and <Timetable> chapter.

4.7 Diploma work

The diploma work will officially begin in July. Unofficially, it has already begun. The project can be considered the analysis phase of the entire diploma work. It is for this reason, that the Saces team does not consider the July date to be the beginning of a new phase, but rather a milestone. Certain documents will be delivered at this time as specified in this handbook.

The result of the diploma work is the application specified in Dr.Hinze's project proposal. This result includes the usual documents such as technical and user documentation. At this point in time (during the project work), it is not possible to plan the duration of all the activities concerned directly with the implementation of the diploma work result. This is because a lot of decisions can only be made after the analysis phase is complete. The Saces team has defined the diploma work packages, which appear to be independent of the project work results. The packages are listed below.

Package	Description	Rough cost in hours	Result
Persistency	Save and restore simulation data to an xml file	40*	xml schema, java classes, documentation
Internal data representation	System model	80	java classes, documentation, uml documentation
Definition dialog 1	Build the working dialog "Reaktionssystem definieren"	40	java classes, documentation, uml documentation
Definition dialog 2	Build the working dialog "Beteiligte Stoffe, Reaktions-gleichung"	40	java classes, documentation, uml documentation
Main application frame	Boilerplate implementation of the main application dialog which will host the simulation	40	java classes, documentation, uml documentation
3D simulation	Design is largely dependent on the prototyping and testing done in the project work. This package contains the logic and presentation layers of the simulation	150	java classes, documentation, uml documentation

* Please see footnote on page 8.

Package	Description	Rough cost in hours	Result
3D simulation integration	Integrating the working simulation into the main application Frame	16	Integration
Unit tests		(Continuous)	automatic unit tests where necessary
Integration tests		(Continuous)	Automatic integration testing where necessary
Function tests		(Continuous)	Test by peer
Diploma work documentation		90*	Complete diploma work documentation

The timetable for the diploma work is not relevant during the analysis phase or project work. Not only is it irrelevant, it would be incomplete and speculative. The reader should note, that only the obvious packages were described above. Negative project work results could complicate the planning and yield more work than is described here. When the analysis phase is completed, the path to be taken will be revealed.

5 Methods and Tools

5.1 Technologies

We use for our prototypes and the standalone application following IT technologies:

- Java 1.4x or 5
- SWT 3.1 (Standard Widget Toolkit)
- JOGL (Java Open Graphics Library)
- Perhaps GCC (Gnu Compiler Collection), gcj and a native interface like JNI or CNI.

5.1.1 Java 1.4 or Java 5

It is not decided yet whether we really use the newest version of Java. While it is very convenient to use the new features of Java 5 like genericity, the enhanced for loop and enums, Java 5 is still very new (but out of beta, however), and is not supported on Mac OS X 10.3 (Panther) and never will be (corresponding to Apple they support Java 5 on their new Mac OS X 10.4 (Tiger) release which is out end of April 2005). Another problem could be the interaction to SWT and JOGL. Most probably we will stick to the older, more established version 1.4.

5.1.2 SWT 3.1

Performance could be a problem for our project. Simulation can turn out to be very resource-intensive. That's why we decided to develop with SWT instead of Swing. SWT works natively a lot and uses the widget toolkit of the operating system instead of emulating the widgets in Java, like Swing does. SWT is not skinnable, but it allows for a compilation to native code with gcj later if we discover that we need native execution to reach our performance aims.

5.1.3 JOGL

It is still not clear whether we really want to do a 3D display of the simulation. We don't yet see whether we can usefully display spheres in a way to give a sense of depth. A very crude approach might turn out to be completely satisfactory: Just ignore the z-axis and paint the circles with the highes z-coordinate first and advance so that spheres at the background are obscured by spheres in the foreground.

* Please see footnote on page 8.

But if we decide to do real 3D graphics, we will need an efficient 3D package. We choose JOGL because we can leverage our knowledge of OGL. JOGL is a very thin Java compatible layer over the OGL standard.

5.1.4 GCC, gcj, JNI and/or CNI

If we decide to compile a part or the whole application to native code, we will use the Gnu Compiler Collection and the gcj Java Compiler. While gcj cannot compile AWT (and therefore does not compile Swing), it has been compiled with SWT with the help of some tricks. An alternative approach is to compile the most resource-intensive Java classes only and call them via a native interface like JNI or CNI. It is completely unclear yet whether use these technology and which ones.

Profiling and careful optimization of Java code might result in very satisfactory code performance-wise.

5.2 Tools

We develop prototypes, a standalone application and elaborate about the current state of art in Artificial Chemistry. We don't need many tools. These are:

- Integrated Development Environment: IntelliJ IDEA 4.5
- Configuration Management with CVS
- Open Office for documentation and printing to PDF
- JProfiler 3.3 for profiling and optimizing hot spots
- JUnit (integrated with IntelliJ IDEA 4.5).

5.2.1 IntelliJ IDEA 4.5

For the software development we need an Integrated Development Environment. Because the team members are used to IntelliJ IDEA we decided to use this IDE for Java development. The company offers educational licenses for a low price.

5.2.2 CVS (Concurrent Versioning System)

Nowadays, sometimes even single-person mini-projects require a configuration management or a version control system. A version control system allows to retrieve older versions of a file and to abandon version branches without deleting them. Much more important is such a tool for projects involving more than one person. CVS allows, for example, that two people simultaneously make changes to file, providing the changes don't overlap (neither syntactically which is cached by CVS nor semantically which is sometimes not even cached by the compiler, lest by CVS).

5.2.3 Open Office

We use Open Office because at home we don't have PDF printing (Adobe Acrobat) and Microsoft Office is not installed on our Mac OS X machines (this operating system supports printing to PDF). We don't have ideological opinions about Open Source and commercial software. It is just a coincidence for our project that Open Office has a feature we need that Microsoft Office does not.

5.2.4 JProfiler

JProfiler is necessary to find the hot spots in our code. While the Sun Hot Spot compiler does a good job of finding hot spots and compiling them to native code, there is still room for optimizing (for example replacing string concatenations to StringBuffer appends in a tight loop). It is important that we optimize late (first get the application running, then optimize) and that we optimize the right thing (JProfiler helps us finding the hot spots in the code). Most probably we will use this tool only in the last few months before the release of the end product.

5.2.5 JUnit

A tool well known and used in the eXtreme Programming paradigm: JUnit (Java Unit Testing), this tool perhaps we won't need. We will decide in a later stage (autumn 2005) whether we apply Unit Testing. In any case, Unit Testing is integrated in IntelliJ IDEA.

6 Standards and guidelines

6.1 Coding standards

Programming in java imposes a certain coding style upon the programmer. Working java code tends to have a typical unintentional structure to it because, unlike c++, there are not hundreds of ways to solve a problem. Saces will be coded in a readable, simple fashion, using conventions from the following document written by Sun Microsystems, it can be found here:
http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

6.2 Documentation standards

Large projects consisting of countless documents require a well-defined body of documentation standards. All project documents should have a similar appearance and reading quality. These qualities are determined by font, layout, formatting and writing style. At best, a project documentation should give the reader the impression, that the documentation was written by one and only one author. This of course applies even more so, to the code base.

The Saces project is in this aspect again, easy to manage. The project handbook will serve as an example for the other documents of the project. The prototype code will serve as an example in code formatting and commenting.

The diploma work documentation may vary slightly from the project work documentation in appearance, depending on our decisions to use a different word processor. All documents will be edited in Open Office and submitted in Adobe portable document format or pdf, as indicated in the Tools chapter. A template can be written containing the conventions used in the project handbook, to avoid listing the conventions here.

6.3 Templates

As discussed in the chapter on Documentation standards, an Open Office compatible template can be written and introduced to the project, after the project handbook has been submitted and evaluated. Other than document templates, the Saces team will use predefined headers and sections for code units and code comments.

7 Organisation

Since the Saces team is a small one, project organisation is no complicated matter. The developers plan to work independently on the packages mentioned above and meet once or twice a week for a progress report and further planning. The team also plans to work together on complicated code once the implementation begins, this includes some parts of the prototype. Other packages, mainly document packages, can be worked on completely independently and are subject to review by the other developer as described in the chapter on development strategy. The developers will not document the meetings in protocols, if the decisions made are only of internal importance. Major decisions, involving project management and the outer circle of the Saces team, will be documented and redistributed to the entire team as protocol in the form of email.

Weekly meetings with the management are not necessary and can be made freely as the project progresses. We have chosen this approach because the problems the developers have had in the past, could effectively be remedied through email correspondence with management. At this point there is no reason to change this procedure. It is however recommended, that all members of the team are up to date on the projects status. For this purpose, the developers are considered publishing the relevant information on a webserver accessible from the internet.

Aside from this, all members of the team will be able to download the latest source code versions from a cvs repository which the team will setup before the actual project kick-off in July.

7.1 Team

The Saces team consists of the following members as indicated by their name and role in the table below. The coders are listed at the bottom of the list in the light yellow field.

Name	Role or field of responsibility	Email
Dr. Thomas Hinze	• Academic and engineering support • Project proposal • Main Client	th1@tcs.inf.tu-dresden.de
Gerhard Schwab	• Chief management advisor • Quality assurance advisor	schwabg@yellowworld.ch
Dr. Peter Schwab	• Academic and engineering support • Project consultant	schwab@hta-be.bfh.ch
Anthony Aguillon	• Student • Programmer • Quality assurance	antona@gmx.net
Daniel Noelpp	• Student • Programmer • Project leader • Quality assurance	d.noelpp@gmx.ch