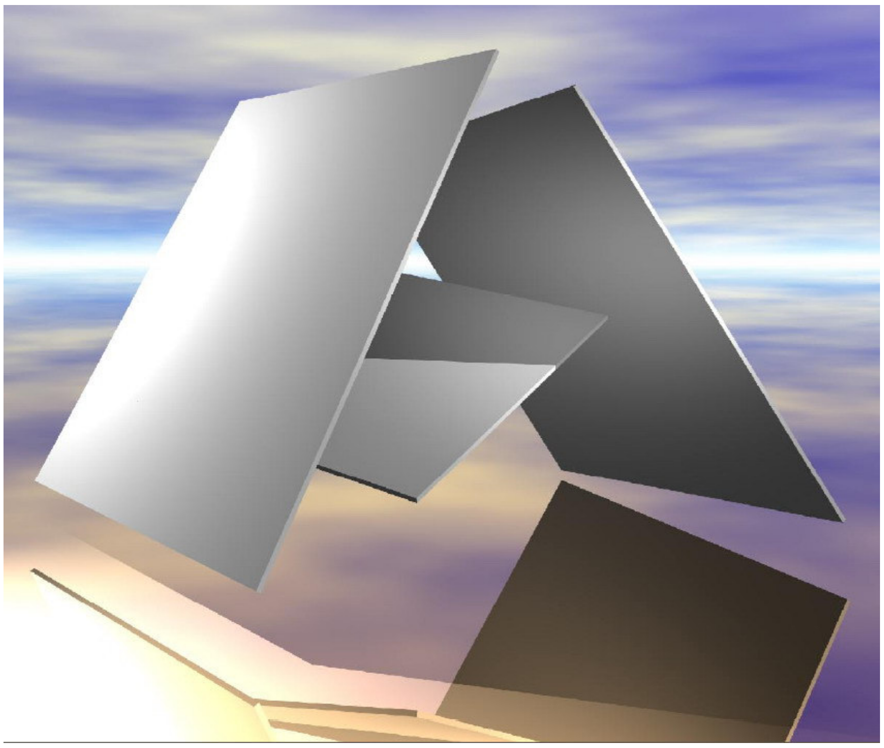

Amadeus Presentation Tomsk

AMADEUS
DEVELOPMENT

The **Amadeus** set of development tools
was written and designed
by **Stefan Metzeler**

You can reach the author for support questions, training
and consulting contracts at the following address:

Stefan Metzeler
Ch. du Stand 19E
1024 Ecublens
SWITZERLAND

Telephone: +41-21-881.57.54
Fax : +41-21-881.57.55
Email : amadeus@neomailbox.com
Internet : www.amadeusITSolutions.com

Foreword

Amadeus Software, now Amadeus IT Solutions, exists since 1986, when the first development framework was created to build business software under MS-DOS. It was initially used in the creation of Medisoft II by Gespower SA in Geneva, a system for medical service providers which was later used by both the emergency services operating in Geneva.

Amadeus Software went on to develop – among others - exhibition and expert systems for Du Pont de Nemours in Europe and the US, personnel management for Hewlett Packard, a payroll system for a canton (state) administration of Switzerland and many more. Over 500 licences for the development framework were sold to developers and institutions

In 1994, Amadeus-3, an object oriented framework for Windows development replaced the previous DOS tools and was used immediately in major projects for HP (Hotel reservation system, Telecom 95, International Salary Survey), Royal Bank of Canada for a private banking CRM (Customer Relationship Management), Du Pont de Nemours (Expert Systems, Ballistics, Literature distribution etc.), Deutsche Bank and IBM, just to name the major clients. An advanced system for airborn laser topography was created by TopScan GmbH in Germany and Optech Inc. in Canada.

Given the wide range of applications implemented with Amadeus software, the framework became very flexible and practice-oriented, leading to small, efficient, fully customized solutions that are user friendly at a very low cost.

Amadeus applications are easy to install and maintain, making them ideal for widespread use by people without an IT support staff. Given the very low resource use, Amadeus applications even work on older machines, low powered portables and are ideal for remote use via Terminal Services, Citrix and other such methods.

1. Presentation of Amadeus and Oberon-2

This presentation is about an alternate approach to programming that has proven very successful in real world environments.

The main points we wish to stress are the benefits of

- Simplicity
- Readability
- Consistency

and we will back up our claims with a

- Proven success record

The programming language Oberon-2

You may or may not have heard about Oberon-2. It is the successor of Pascal and Modula-2 by Prof.Wirth from Zurich. Everyone knows Pascal, so you might be tempted to think that it is very similar, but except for a few syntax elements, it is not. It is far, far more advanced.

Why should we be interested? If it was so good, we'd have heard about it.

This is the problem: information! You don't just wake up in the morning thinking "Oh, there is this great programming tool Oberon-2 I should start using". You have to learn about it and that is what we're here for.

Microsoft can advertise their Windows and Office all over the world, they have lots of money and you know that it's not because their tools are good that they sell so many of them.

Better does not always mean well known. Especially in programming, tradition, habits and status are very important. Fortran is stll being used very heavily!

Why is Oberon-2 a good choice?

It's very small – 20 pages of documentation – and hence easy to learn. It's extremely readable, has a regular syntax and induces you to write consistent code.

It's very efficient – executable code is highly optimized and very fast, does compare with the best compiled languages and the compiler itself is blazingly fast. The executables are also incredibly small, e.g. a large application will typically be only 2.5 – 4 MB on average.

It has a garbage collector and finalization that are so fast that you never feel any delay. Both are invaluable for writing complex code.

Does notation really make a difference?

- Think of doing mathematics in roman numerals vs. our decimal system; it's almost impossible to do a multiplication in roman numerals

- You spend 95% of your time reading code, only 5% writing it, so readability is essential. With good editors, you don't type any program structures anyway, they will be filled in for you
- "But I like C style coding": yes, that might be a habit that is hard to break, but once you switch, you'll never look back; the human brain is extremely good at recognizing patterns of a certain length and the ideal is more than 1 character; try it!
- Terse source code does **not** translate into efficient code – your compiler actually has to work harder to understand what you're trying to get it to do

Main stream vs. less known tools

Is it important to use something that is mainstream?

- **The end user absolutely doesn't care what you use to write your code**
- You you are the one who needs to be comfortable with your own code
- Most likely, no one outside your company or group will ever see your source code
- You can use other tools and libries with Oberon-2
- C++ is not really that standard; everyone uses it in different ways
- Companies who hire C++ programmers always need experienced people (expensive)
- They also need to re-train them in their own standards
- Companies that use Oberon-2 can very quickly train and re-train programmers, even not so experienced ones

What's your benefit as programmer in using Oberon-2?

Your main goal is PROBLEM SOLVING. When you get paid for writing code, you don't want the "intellectual challenge" anymore. You want to write good and reliable code quickly for your customers who pay for your work.

If you are fast and good, you get paid more and you have more fun. Your programming tools should assist you, not the other way around.

Oberon-2 supports solving problems by taking away the worry about typing errors, inconsistencies etc., allowing you to focus on the algorithms you want to use.

Garbage collection, modularity and very readable code make it easy to implement even very complex systems quickly.

Isn't bigger better?

C++ is a very large language with many features. Isn't that good?

- No, it's actually one of it's main defects: bit means huge complexity
- Large compilers are complex and hence riddled with their own bugs
- Very few people actually use all the features, so you have added complexity without benefits
- Some features are even harmful to good design (multiple inheritance etc.)
- Even experienced programmers may not understand any given piece of code
- Many features are late add-ons, such as name spaces
- No compiler strictly conforms to all the standards
- Despite the size, it doesn't even support garbage collection, which is essential

And what about all the huge libraries, tools etc. that you can get for C++?

- You can't freely mix and match any given set of libraries and tools
- Each large library requires a huge learning effort
- Many libraries include redundant code
- You might end up with different memory models, conflicting names etc.
- You become totally dependent on the tools and libraries that you use
- Few libraries are available with source code giving you some independence
- Many environments now generate a lot of code (e.g. .NET) to implement user interface features, so you have less control over code quality, size and consistency; it does tend to lead to gigantic executables

RISC vs. CISC

Many years ago, there was a huge battle between rivalling microprocessor architectures. There was the CISC approach, where the designers tried to add more and more instructions to the processor to be executed in microcode, assuming that this would make the code faster as the applications could directly use those high-level instructions instead of lots of simple functions.

Then came the RISC architecture, which took exactly the opposite approach: it reduced the instruction set to the absolute minium while making them extremely fast, putting all the complex functions into libraries that could be used by the programmers and that were easy to modify.

The battle has been decided a long time ago: the RISC model won hands down!

The basic tools must be as simple as possible and the complexity should be in libraries. This makes it possible to optimize the basic tools to the max while keeping complexity in easily maintained code libraries, avoiding major sources of error that could affect every application running on the system.

The same approach should be used in compiler design. The compiler is the equivalent of a microprocessor to the programmer using a high-level language.This basic tool should be very simple, totally reliable, well known and defined precisely.

Writing your own tools vs. buying tools

If there are not many Oberon-2 libraries, do we have to write all our own code?

- Depending on your task, it may well be possible to write your own objects with Oberon-2
- It is often better to do so than to import large libraries if you need only simple features or a small subset
- External libraries imply dependence and potential error sources
- You can still buy commercial libraries and use them with Oberon-2, especially for very complex features (e.g. OCR, image processing etc.)

Presentation of existing applications

The proof of concept are the existing applications realized with Amadeus-3 and Oberon-2.

Good examples to include in a life presentation would be the following:

- A3Edit
- BMan
- Amadeus Confidential
- Amadeus Address