

A view of a Systems Engineer on modern seismological software

How to build scalable software, ready for GRID-based computing

Drs. Fotis Georgatos
<gef@ceid.upatras.gr>

ORFEUS/EMICES Workshop
September 22-24, 2003

MEREDIAN Annual Meeting
September 25-26, 2003

Advertisement

◆ Who am I?

- ◆ \$\$\$ Independent consultant with experience in projects of non-profit organizations \$\$\$
- ◆ A vivid Open Source software advocate (Linux, gcc) in Greece, since the early '90s
- ◆ I enjoy to debug programming languages; while on vacation atop of CTF :)

◆ What is my background?

- ◆ External consultant of Technical University of Athens (NTUA)/Demokritos for GRID-based computing and cluster administration (affiliation still current)
- ◆ Worked on contract-basis in a series of projects for Greenpeace International: Internal Directory, Systems Security, Mailing System, Ships' Communication System
- ◆ Worked for 2+ years as a network engineer at RIPE NCC, Amsterdam (NL), the body that holds together the ISPs in Europe and their IP addresses (partners: ARIN, APNIC). Administrator of the Test Traffic project, a distributed measurement network of 50+ nodes, using GPS for timekeeping, which had surprising similarities with...
- ◆ My diploma work, which was to design a Linux and PC/104 based digital seismograph, suitable to be used as a network node for real time data acquisition; being the terminal project at Computer Engineering & Informatics Dpt, Patras University, during 1998-99.

◆ What do I have to offer?

- ◆ Guidelines on building and evaluating software components, including code+GUIs+UIs
- ◆ Requirements for software systems to be GRID-ready. At best, for cluster computing
- ◆ **Answer your relevant questions. At any moment, I will be delighted to answer.**

Preface

◆ Typical issues addressed in Seismology:

- ◆ data handling and exchange => software that collects & distributes raw data
- ◆ quality control => manual or automated data classification
- ◆ storage/archiving => software that takes care of bookkeeping
- ◆ routine analysis => standard or custom seismological software

◆ Software building IS needed in seismology

Demand is quite pressing because this science is pretty much over promptly and timely deducing conclusions from current natural phenomena (=information is needed on spot).

◆ Motivation for DIY software engineering

- ◆ At first, seismologists have a constant need for a continuous, robust, and (near) real-time stream processing and storage of data.
 - ◆ At second, they apply on them functions of ever changing complexity and, might require information of the present or past depending upon varying criteria.
- The latter is what forces seismologists, at a faster or slower pace, to follow the path of custom software development.
- Albeit they manage quite well in this respect, there exist risks; that are recurring often in this ongoing effort. These, in turn, might prevent applications from scaling up at a later stage of a project, particularly while using massive systems such as GRIDs are.

The evolution of massive computing



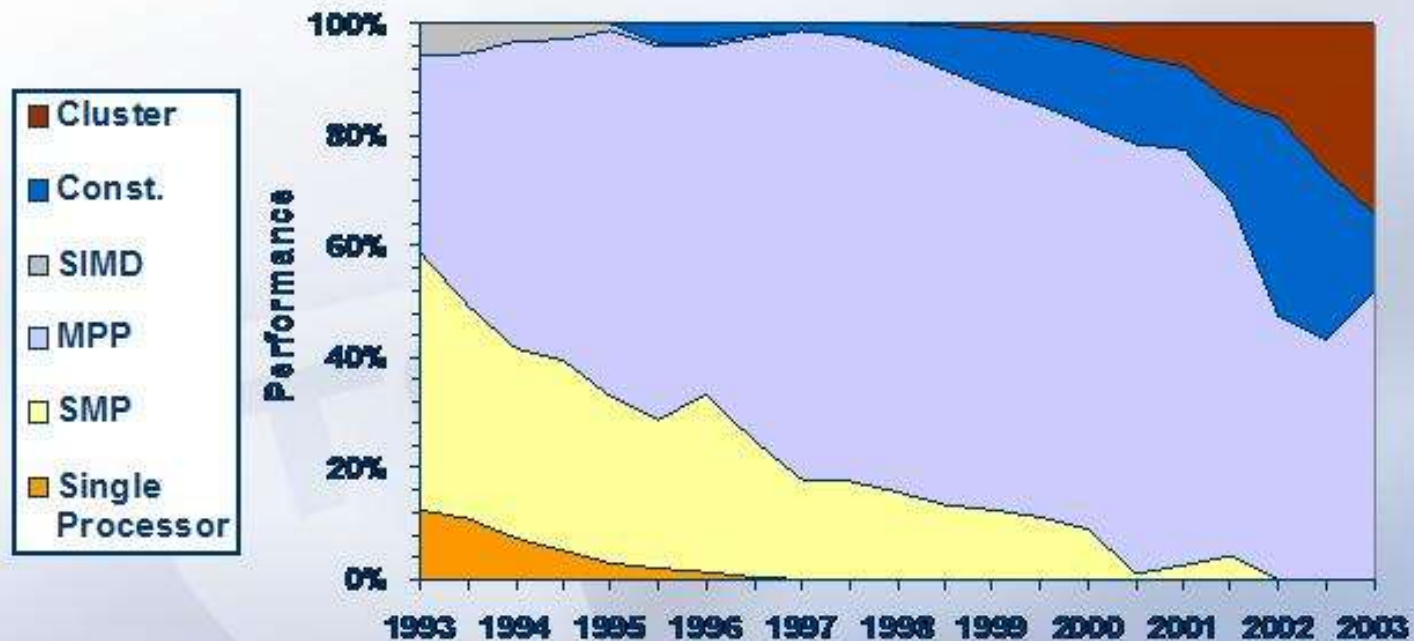
Performance Development



So, what drives this development?

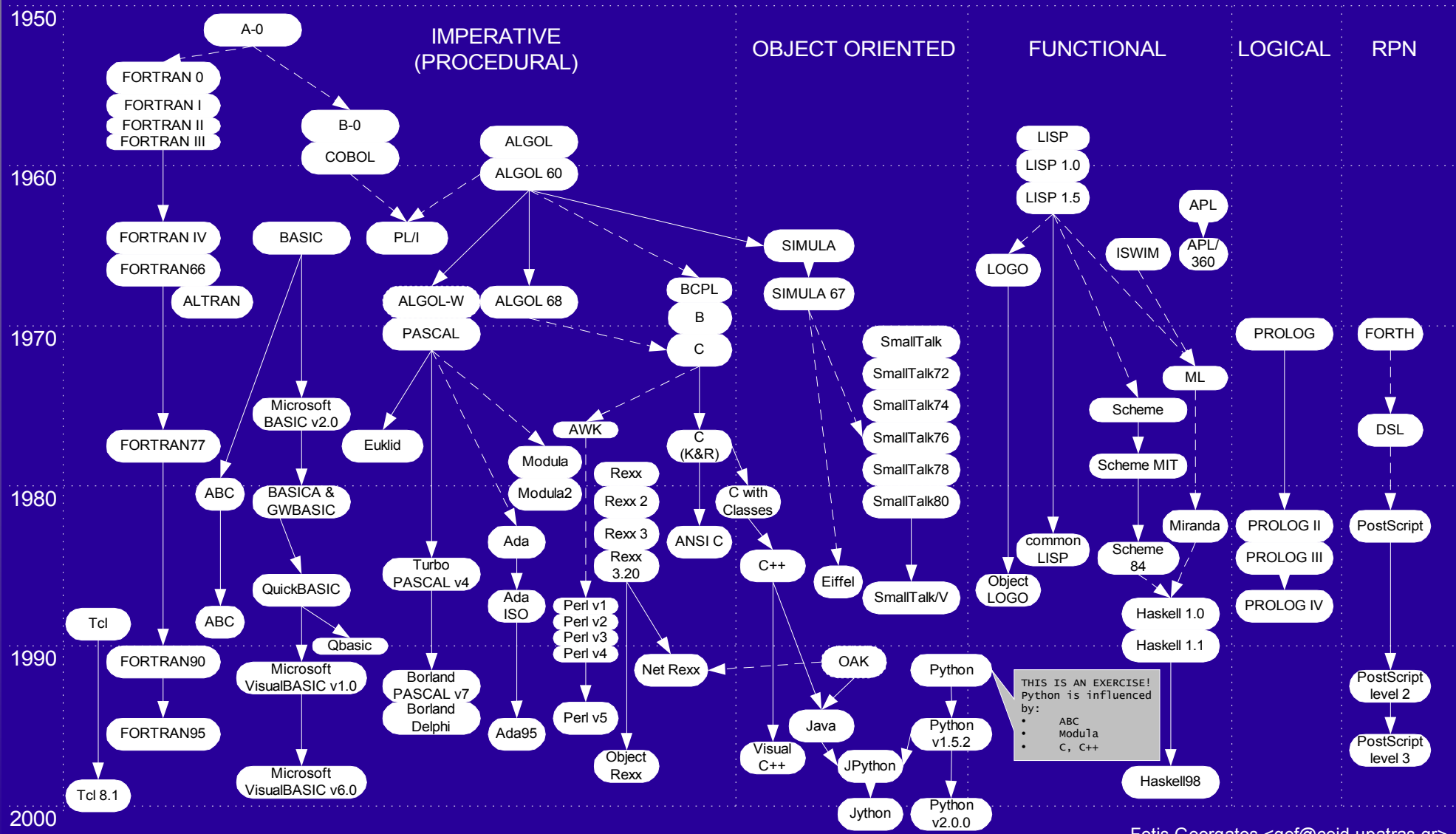


Architectures



How to use all this available power?

COMPUTER LANGUAGES GENEALOGY TREE EVOLUTION BETWEEN 1950-2000



Object Oriented (OO) Programming

◆ What is Object Oriented Programming?

It is a technique of software engineering, not a panacea for every problem. It was invented back in the 60s for simulation applications (SIMULA), but it is particularly demanded, nowadays, for building Graphical User Interfaces. But remember, OO != GUI

◆ Basic concepts of OO

Many of them might seem foreign to non-CS audience (this is normal, so please ignore):

- ◆ object encapsulation
- ◆ multiple inheritance
- ◆ dynamic inheritance
- ◆ multiple polymorphism
- ◆ garbage collection

◆ Popular OO Languages include...

- ◆ Java
 - ◆ C++
 - ◆ Python
- and so-called “modern” versions of Basic, Pascal, Fortran.

OOP, it also has disadvantages

◆ Requires good understanding of concepts

- ◆ Defining the world of objects and their properties (data and methods)
- ◆ Managing the security model of the object world (C++/Java: public, private and so on)
- ◆ Populating the world of objects in memory, while considering their run-time evaluation

◆ Coding complexity, syntactical overhead

- ◆ OO comes at the expense of extra lines of code that have to be written and maintained
- ◆ Teams of programmers must have good, understandable, documentation of object definitions

◆ Performance

- ◆ OO does provide a performance penalty and is not appropriate in heavily computationally intensive applications. This is the reason why SIMULA was abandoned!
- ◆ Operating systems' kernels and language compilers, are NEVER object oriented! Still, GUIs and “glue logic” of Windows, Linux, MacOS often make use of OOP.

→ Watch out! Use OOP only after sceptical evaluation of your real needs.

→ Read the OO FAQ if in doubt: <http://www.avalon.net/~wbachman/OOFAQ/oo-faq-toc.html>

→ If you decide to use OO, consider applying **UML** (Unified Modelling Language) for proper communication of information on your objects, among the development team(s) members.

What computer language to use and when?

DISCLAIMER: When you cannot afford doing or comparing with anything else, use the one you already know very well. This applies, only if you are working on one-man-project.

- ✓ Still, you can do much better, if you have concluded on your application's priority list. Then, you can choose the most appropriate one(s), according to task(s).

◆ Which field does each language excell in?

- ◆ prototyping => Python, awk, bash (in general, any of the scripting languages)
- ◆ portability => C, (think twice before using C++ or Java for GUI)
- ◆ speed of wire => assembly (suggested *only* after *profiling*)

- ◆ Internet Applets => Java (-Virtual Machine!)
- ◆ Web2database => PHP (note that this is a special purpose language)

◆ Prefer a combination of software layers

- ◆ There should be well defined APIs among the layers and trully rational thinking behind interfacing decisions. Prefer existing, tested and working code, it is always better.
- ◆ The languages of these software layers could be different. There are tools that make this process very automated; for instance, think of an inversion code and a library: You could use ScaLAPACK (a library of high-performance Llinear Algebra routines for “distributed-memory MIMD machines” =cluster!) and apply SWIG to get Python hooks.

Some practical hints to remember

◆ C is meant to be a system language

You have to do the memory management on your own.

◆ Fortran does help in memory management, but is quite poor in nearly everything else

Code layout, syntax, portability and interoperability all suffer in Fortran environments. Allocating, say, an array is trivially simple though. The latter is the reason that still many scientists prefer it, along with the fine reason of being around first (!). Interesting notes:

- ◆ major computational kernels are often first written for Fortran
- ◆ well, the Fortran compilers are always written in C!

◆ Java requires you to cope well with Objects

- ◆ Java does the memory management for you, at the expense of verbose code, see:

```
Public class NoTest { public static void main(String[] args){} }
```
- ◆ You can tap the benefits of the Java Virtual Machine without using always Java itself; There are succesful efforts for cross-compilers producing Java bytecode, eg. Jython

◆ Scripting languages are here to stay!

- ◆ It is more convenient to write the User Interface in a scripting language, so do Pros
- ◆ It is unlikely you want the complete code to be a monolithic package in one language, but even if you do, prototype first in a scripting environment, to settle on your ideas.
- ◆ Beware of Tk / GTK / wxWindows. Watch out for Perl, it's contaminating :)

Tools for storage of data

- ✓ Remember that we store information, because we might need to find it back one day!
- ✓ This implies that placing and searching data files on the raw filesystem, might be very inadequate in a range of cases.
- ✓ For storing huge amounts of data, there exist a couple of worthy tools.

- ◆ A true database system is appropriate for record-like objects (the so-called “tuples”)
 - ◆ MySQL is exactly fit, if you want to store calculated parameters of earthquake events
 - ◆ PostgreSQL, alternatively, allows for Object Oriented data storage. Also, it has a replication mechanism that allows distributed sites to share information nearly at speed of wire. This mechanism can be used to prevent Single Points Of Failure!

- ◆ A stream-oriented backend such as ROOT
 - ◆ ROOT solves the problem of continuous recording of vast volumes of incoming data
 - ◆ Addresses well the possibility of adding later "dependent" streams of information, such an STA/LTA ratio or other on-demand computable functions.
 - ◆ It contains an interactive C++ environment, which is ideal for researchers.
 - ◆ ROOT includes excellent bookkeeping support for offline media, such as tapes are.

- Do you really have greater needs in high data volume than CERN has for LHC? Wow!
- You may find an example of the Test Traffic experiment, implemented by RIPE NCC, at: http://www.ripe.net/ttm/General/tt_examples.html

Use the fittest tools at hand

◆ Choose Open and Portable components

If you are building a software system to be deployed on long-term, study in advance and choose carefully the Operating System, the Language(s) of implementation, as well as the corresponding compiler(s).

- ◆ Try to make your software modular, with as simple and few components as possible.
- ◆ Investigate well for relevant, well tested, libraries and tools already written by others.
- ◆ Make choices that allow maximum compatibility and portability among architectures: For example, don't get caught in the Big/Little Endian game!

◆ Do not be afraid of mixing systems

- ◆ Make sure you understand how the subsystems communicate and be willing to take the corrective action to find the part that malfunctions or fails. Replace only that.
- ◆ For example, if you pick Windows as your working environment, consider seriously Linux or another UNIX system for implementing your storage backend and doing batch processing work. It will pay back later on, when your processes will become long-term procedures and overall stability tops as your major requirement.

◆ Hey! We want to avoid two technologies.

- ◆ Well, do you think you will stay long with 32bit computing? Opteron is here and strong!
- ◆ **Get prepared for constant change, do it once and do it in advance.**

What specifically NOT to do

◆ Do not make a single binary, GUI with code

Rather, make it such that the UI "pushes" the User's requests on the major program, which is preferably done through a command line API. This separation is vital for GRIDs.

◆ Do not rewrite computational kernels

...when you may so easily find one suitable for your needs, say, at the least Matlab or, its freeware equivalent Octave. Both are based on BLAS-1/2/3, LAPACK, the Netlib tools:

<http://www.netlib.org/>, <http://www.nhse.org/hpc-netlib/>

Follow this link for more tools: <http://sal.kachinatech.com/A/2/>

BTW. Watch out for proper tuning of the IEEE-754 arithmetic unit!

◆ Do not rewrite existing software components

...because then only YOU know how it works and you can't gain experience of other people. You will likely be losing both on functionality and extensibility.

For example, haven't you written a database-like component for storing and retrieving focal events? Try to query it for magnitudes $2 \leq M \leq 3$ OR $5 < M < 7$ AND NOT $M = 6.4$

Hint: Are you about to write your own XML parser? There are plenty of libraries for this!

◆ Do not write another programming language!

There are more than 2000 already, and another special purpose language that solves only a limited range of problems, is possibly doomed to be forgotten a few years from now...

Maybe all you need is just a configuration file? (think of the case of event detection)

Real world: a review on GMT

- ✓ Generic Mapping Tools is a well established suite for generating maps, which is in wide use among seismologists, due to its open architecture.
- ✓ In summary, it generates Postscript code by processing vector or raster input data.
- ✓ For your information, it is built with C, sh and awk code.
 - ◆ Prototyping: Good; but using sh & awk is poor for portability out of the UNIX family
 - ◆ Portability: Excellent, its major components are written entirely in standard C
 - ◆ Architecture: Probably bad, it generates numerous intermediate files; not scalable
 - ◆ Command Line: Moderate API, because of very hard to apply command line options
 - ◆ Speed: Very good, C is sufficient for the task, there is no need for assembly
 - ◆ Extensibility: Excellent, it is written in multiple and well defined layers
 - ◆ Code reuse: Great; NetCDF is well-established and working code, albeit archaic. PS is an established standard, managing a range of printing devices.
- ✓ Overall: Excellent effort for its era ;-)
Judicium: It does one thing and well, to generate PS maps; intended for writing papers.
 - ◆ It allows an external User Interface to be build (iGMT that is) and allows fine tuning (eg. to split subtasks and run multiple processes on a parallel system is possible).
 - ◆ Generating other formats such as .gif, .jpg, .png and so on, is simply done with ghostscript (gs). Think of how much code has been avoided with this elegant idea.
 - ◆ BUT, it can't take advantage of GRID technologies, because of improper messaging.
 - ◆ Not sufficient for web-based interactive manipulation, due to the slow nature of PS.

Software that might become handy

◆ Octave, a top class computational kernel

[<http://www.che.wisc.edu/octave>]

- ◆ A Matlab-like high level interactive language primarily intended for numerical computations. It can do arithmetic for real and complex scalars and matrices, solve sets of nonlinear algebraic equations, integrate functions over finite and infinite intervals, and integrate systems of ordinary differential and differential-algebraic equations.
- ◆ The underlying numerical solvers are standard public domain Fortran packages such as Lapack, Linpack, Odepack, Blas, etc. packaged in a library of C++ classes. Plotting (2- and 3-D) is fully supported via Gnuplot.
- ◆ A 200+ page manual is included with the distribution, which is available as source code as well as binaries for several platforms, e.g. DEC Alpha and Ultrix, SGI, HP, IBM RS6000, Sun, Linux, Netbsd and FreeBSD UNIX platforms and Next boxes. Binaries are also available for Win NT/95 platforms.

◆ Grass, Geographical Informations System

[<http://grass.baylor.edu>]

- ◆ GRASS GIS (Geographic Resources Analysis Support System) is an open source, Free Software Geographical Information System (GIS) with raster, topological vector, image processing, and graphics production functionality that operates on various platforms through a graphical user interface and shell in X-Windows.
- ◆ It is released under GNU General Public License (GPL).

◆ Cygwin, Unix combatibility suite 4 Windows

[<http://www.cygwin.org>]

- ◆ Cygwin is a Linux-like environment for Windows. It consists of two parts:
- ◆ A DLL (cygwin1.dll) which acts as a Linux emulation layer providing substantial Linux API functionality, and a collection of tools, which provide Linux look and feel.
- ◆ The Cygwin DLL works with all non-beta, non "release candidate", ix86 versions of Windows since Windows 95, with the exception of Windows CE.

Summary

✓ Write scalable code by not repeating work!

- ➔ You don't have to be a software engineer to make use of the mentioned criteria, because **YOU ARE THE USER OF THE SOFTWARE AND YOU MUST ENSURE ITS QUALITY!**
- ◆ Break up your software architecture in sensible subsystems of components.
- ◆ Pick your range of programming languages and tools wisely, according to your needs.
- ◆ Look actively for a library doing the computational work, or a database for the storage and retrieval part. More than 95% of all cases are just matrices operations and there is a handful of well written open-sourced libraries that can likely be compiled and optimized for your platform. IF you believe you can do better, please write a paper on this, let us know! (Do you have your own code for FFT calculation? Or, do you employ software doing so?)
- ◆ Chances are, that you haven't really come up with an overall original problem wanting breakthrough algorithmic solutions in all of its aspects. So, rather someone else has solved (part of) it already. Check these sites, just to be sure:
 - ◆ <http://gams.nist.gov/Classes.html> (search by problem taxonomy)
 - ◆ <http://sal.kachinatech.com/sal1.shtml> (search by sw category)
 - ◆ <http://orfeus.knmi.nl/other.services/software.links.shtml>
 - ◆ <http://quake.seismo.unr.edu/htdocs/students/lchinose/LINUX/>
 - ◆ http://ct.gsfc.nasa.gov/esmf_tasc/Files/Bibliography_b.html