# EPCC-SS98-08

# EPIC2001


## Mario Mitas



**Abstract**

The main purpose of this project was to finish an implementation of the EPIC2001 package. EPIC2001 is planned as a replacement for EPIC (*EPCC Interactive Courseware*), a package for distance learning accessible via the Internet. This new implementation was realised with respect to multiplatform use, backward compatibility with the existing Perl version of EPIC as well as adding new features to the old EPIC functionality. The project also involved developing a graphical user interface for EPIC2001. Major reasons for reimplementing of EPIC were: to explore the possibility of exporting EPIC courses to use the MS Windows environment, to improve efficiency of EPIC command execution and to use modern portable tools for implementation so that EPIC remains up to date with current programming technology.

This document provides a description of the EPIC/EPIC2001 functionality, structure and contains detailed information about the EPIC2001 design, architecture and implementation.

# Contents

# 1 Introduction

EPIC is an acronym for *EPCC Interactive Courseware*[1]. The package was initially developed at EPCC in 1994 and has been actively improved since then. It enables users to chose, install and proceed with existing courses. Each course consists of a theory part and exercises. For most of the exercises the user is required to write a short program to apply the knowledge gained from the theory part. At the moment, there are eight courses available under EPIC (Message Passing Interface, High Performance Fortran … etc).

EPIC is interactive, easy to use and accessible via a browser. It is built on WWW technology, which became common for the information provision, so users should not have major difficulties in adapting to this way of education. EPIC is useful for self-tuition and can be used in classroom teaching as well. The user fully directs the rate of learning progress. Online help and external references to other information sources for a given problem can be accessed by clicking on hyperlinks. If any difficulties are encountered, the user has the possibility of asking for specialist help from EPCC via EPIC itself.

WWW technology itself is hardware independent. However for EPIC to work there are certain prerequisites in functionality that must be installed at the client end. At the moment the client programs can only run under the UNIX operating system. Moreover they require additional software to be installed (a make utility, visualisation programs, compilers for HPC…). EPIC has not got a consistent user interface and basically is based on the technology that was available at the date it was created. Also with the advent of Java it became desirable to port EPIC to use this as its basis.

The main goals of this EPIC2001 SSP project were to finish a partial implementation of the existing EPIC2001 package, which introduces different EPIC functionality and usage. The author of this initial version of EPIC2001 was H. W. Yau and it was partly based on an earlier SSP project [Ref. 1]. In addition, new features and a new graphical user interface for EPIC were developed. There was also a need for the multiplatform implementation of EPIC, oriented towards UNIX and NT systems. Backward compatibility with the existing EPIC courses had to be maintained too. Java was chosen as the new environment for EPIC2001, for its portability to a number of platforms and for its expected success in a software development area.

# 2 Background

This section gives a rough description of the EPIC[2] features, implementation and its development history. It also contains certain technical details needed for a full understanding of the principles on which EPIC is based on.

[1] EPIC is accessible via URL http://www.epcc.ed.ac.uk/epic/.
[2] Note that term EPIC can be used in two slightly different ways. It can be understood as a name for the whole distance learning package (including web server, UNIX utilities…) as well as a synonym for the programs originally written to handle and execute the EPIC commands (`epic_com`, `epic_master` scripts – described later in the text).

## 2.1 The user view

In order to easily understand the functionality behind EPIC, let us introduce its main visible components, see Figure 1.

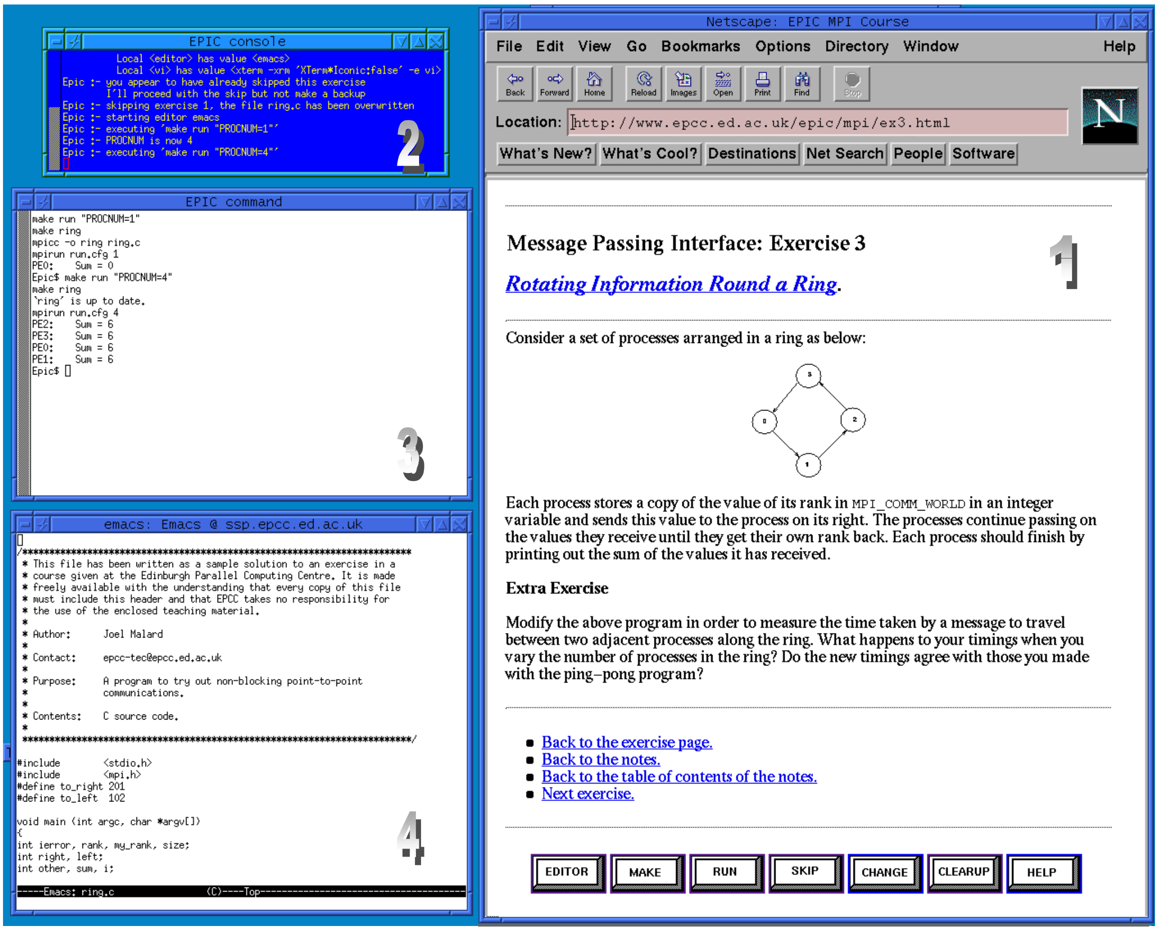



**Figure 1: EPIC running on UNIX.**

The main components that can be seen on this figure are:

- Browser (element 1) – essentially needed to navigate EPIC courses. The browser is in this case displaying one of the EPIC course exercises. At the bottom of the browser window there are a number of buttons available to the user. After clicking on a button a proper EPIC command is generated and the desired action is executed.
- EPIC console window (element 2) – automatically opened when first EPIC command is generated. The user can see the status information generated by EPIC on this window.
- EPIC command window (element 3) – automatically opened when EPIC interacts with the operating system. This window shows the results of the exercise programs.
- A program invoked by EPIC (element 4) – run by EPIC if needed. In this case the program shown is an editor displaying the source code of the exercise program.

## 2.2 EPIC functionality and structure

The EPIC package can be defined as a set of programs and UNIX utilities connected by WWW technology to provide a consistent, interactive tool for distance learning. The EPIC consists of three major parts: Server Side Part (SSPt), Client Side Part (CSPt) and the Courses.

The SSPt is located on EPCC web server [3] and contains:

- HTML pages describing the general functionality and providing help for EPIC.
- HTML pages for the initial CSPt transfer and installation.
- A program to execute the EPIC commands generated while working on a particular course.

The CSPt is installed on the client machine (after it has been transferred from the EPCC web server). It consists of:

- A program to execute user invoked EPIC commands while working on a particular course (different from SSPt commands). It also handles any user operating system commands entered via command window.
- UNIX utilities and programs such as `make`, `AVS` … needed for EPIC functionality. Some of these are course specific, some are needed for general EPIC functionality.

The Courses are initially stored on EPCC web server. After a user has selected a given course, they are required to download and install files containing course exercises on the user's own machine. Courses generally consist of:

- HTML pages located on the EPCC web server. These pages contain the course theory, all the information necessary to complete the given exercises as well as instructions for solving these exercises. An user can also invoke various EPIC commands via these pages.
- Template files for the course exercises, solution source files and any files needed to build executable program from user's solution of given exercise/problem.

The general architecture of the EPIC system is shown in Figure 2. This structure is needed because of a way the EPIC works. EPIC basically functions as an intermediary between the web environment, which has no direct access to the client machine and the programs located at the client end. The EPCC web server and browser must be incorporated as they are used as a general tools to provide access to the requested information and also they fire up the relevant applications on both sides.

[3] http://www.epcc.ed.ac.uk/epic/

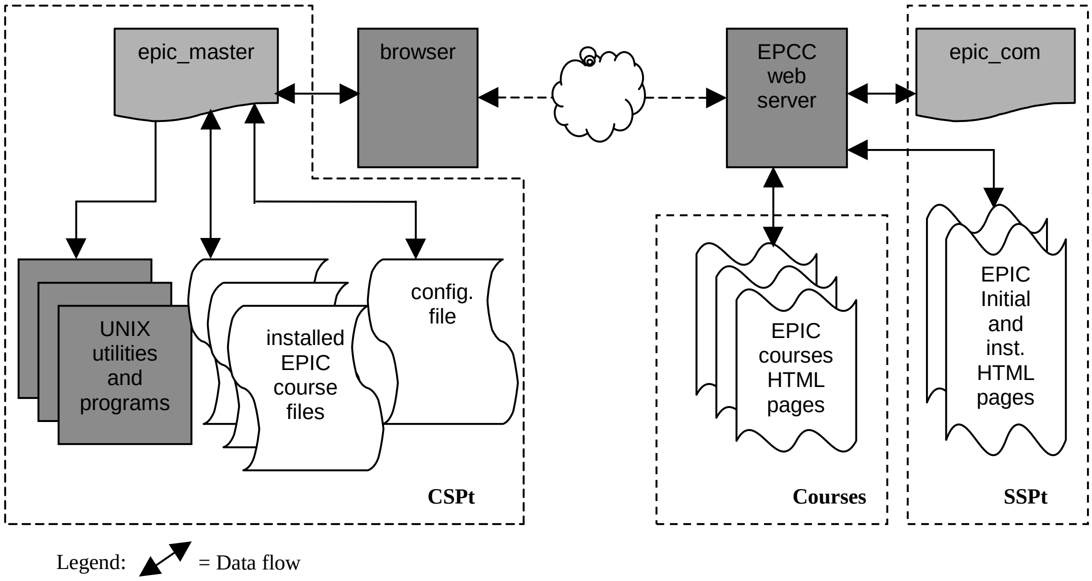



**Figure 2: Schematic diagram of the EPIC system.**

The SSPt program is called `epic_com`. It is a CGI [4]script written in Perl. Each time an user performs an EPIC command, say by clicking on a given button or hyperlink on an SSPt HTML page, the `epic_com` script is invoked with appropriate arguments provided by browser (how and why, will be described later). It executes just part of the EPIC command set.

The CSPt is usually installed in the `epic` subdirectory in the user's home directory. This directory is created during EPIC installation and initially contains the `scripts` subdirectory in which the `epic_master` script and the `config` configuration file can be found. The EPIC home directory also contains a `checkpoint` file in which the user's activity is traced to provide a mechanism for finding the user problems or any EPIC faults. All files for courses are then downloaded on demand and installed in the subdirectories within the `epic` directory.

`epic_master` is the program providing the desired functionality of the CSPt. It handles part of the EPIC commands and manages the console and command windows. The console window is used for showing the EPIC status information such as configuration file loading, error messages … etc. The command window is used for the execution of shell commands typed directly by user (like seeing the current directory structure…) or shell commands generated by EPIC according to user's instruction (building or executing programs from particular course exercises…). `epic_master` is a script file written in Perl, so a basic prerequisite for CSPt functionality is a Perl interpreter (version 4 or 5) to be installed. The `epic_master` script is started automatically by browser (how and why, will be explained later) or by the user (by typing the command directly into command window). Command is passed into this script as input argument. Implementation of the `epic_master` script can be briefly described as:

- Set global variables.
- Call a routine to process internal console command, if such a command is detected.
- Perform system initialisation.

[4] *Common Gateway Interface* is a standard to ensure that a web server can start relevant application within its workspace to provide data for given user request.

    - Determine OS.
    - Open sockets for communication between the console and command window.
- Open the console window, if it is not opened yet.
    - Open the console window and connect it to a given socket.
    - Redirect STDOUT and STDERR to the console window.
    - Write initial strings.
    - Read the `config` file.
- Process EPIC command passed as a command line argument (each EPIC command has an independent function call. Existing commands are: `trans`, `install`, `change`, `check`, `epccchimp`, `epictest` – for setup and `cleanup`, `edit`, `echo`, `make`, `mvExercise`, `skip`, `tutor`, `extern` – directly invoked according to the button pressed).
- Terminate.

The `config` file contains the configuration information, which is loaded each time the `epic_master` script is invoked. The structure of this file consists of pairs of strings "<key> <value>". The content of this file is maintained by `epic_master` and changes according to the exercises being undertaken (various parameters for a given exercise are stored here). The default minimal configuration file is as follows:

```
emacs emacs                                //command for editor invocation
ue xterm -xrm 'XTerm*Iconic:false' -e ue   //command for editor invocation
PROCNUM 4                                  //number of processes to be used
xedit xedit                                //command for editor invocation
editor vi                                  //editor choice
vi xterm -xrm 'XTerm*Iconic:false' -e vi   //command for editor invocation
exec_f77 f77                //command for Fortran77 compiler invocation
exec_hpf hpfff              //command for HPF Fortran comp. invocation
```

The EPIC commands that an user can invoke differ from exercise to exercise of a courses. The complete set of explicit EPIC commands accessible via buttons (see Figure 1) on HTML pages consists of:

- EDITOR – invokes a user-preferred editor to edit the exercise source file.
- MAKE – builds the actual exercise executable file from the user's source file.
- RUN – runs the actual exercise file (uses MAKE command).
- VIEW – displays any graphical representation of the results of exercises execution (uses MAKE command).
- SKIP – skips the current exercise and continues with the next one, copies or shows the solution to the exercise.
- CHANGE – changes the user-defined settings – preferred editor, user defined editor invocation, number of processes, hosts file.
- CLEARUP – kills all the processes and closes all the EPIC generated windows.
- HELP – provides on-line help on EPIC commands and enables the user to send a query to an EPCC nominated tutor.

The commands, which are invoked implicitly by clicking on appropriate hyperlink on EPIC HTML pages, are:

- TRANS – transfers and installs the desired EPIC course on the user's machine.
- INSTALL – re-installs the `epic_master` script.
- CHECK – checks the existence of the given files on the user's system.
- EPCCCHIMP – installs the CHIMP [5] package on the user's computer (needed for MPI course).
- EPICTEST – tests the current CSPt installation on the user's machine.

### 2.3 Behind the scenes

As has been already mentioned the EPIC functionality is basically provided by two Perl scripts – `epic_com` and `epic_master`. The `epic_com` script is located on the EPCC web server and `epic_master` is placed on the user's computer. A user does not explicitly invoke either of them. These programs are run automatically according to the user's actions on a given EPIC HTML page being displayed by the browser. This is possible thanks to the use of CGI processing and *Helper Application* invocation by the browser.

The EPIC command invocation begins with the user's clicking on a hyperlink (or button) on the EPIC HTML page containing the URL, which then invokes the `epic_com` script. Such an URL looks like:

/cgi-bin/epic/epic_com/edit&mpi/hello&.ex1.soln

The first part (/cgi-bin/epic/epic_com) is the full path to the `epic_com` script on the EPCC web server. `edit` is the EPIC EDIT command for the invocation of an editor on the client end. `mpi/hello` is the directory relative to the EPIC home directory on the user's machine containing the file to edit and finally the `.ex1.soln` is the file to edit within the given directory. This URL tells the EPCC web server to invoke the `epic_com` program with arguments `edit`, `mpi/hello` and `.ex1.soln`. The `epic_com` then parses the input arguments and determines whether the command passed should be processed by `epic_com` or by `epic_master`. EDIT is an `epic_master` command so the `epic_com` sends this command back to user's browser via the EPCC web server as an `application/x-epic` response. The browser parses the server response and if it finds the `application/x-epic` response, it then runs the `epic_master` script with the response content as command line arguments. So the `epic_master` runs as '`epic_master edit mpi/hello .ex1.soln`' and it invokes the editor to edit given file before terminating.

To restate the previous paragraph in technical language – after clicking on an HTML reference which makes the browser generate the CGI HTTP [6] request for `epic_com`, the browser receives the HTTP response header with `Content-type` field set to `application/x-epic` and the response code `200` (`OK`). The content of the response will be the string determining the EPIC command and its arguments. MIME [7] type `application/x-epic` is set during the EPIC installation associated with the `epic_master` script, so the browser invokes this script with the received content when it encounters this MIME type.

[5] CHIMP was an EPCC based implementation of MPI. It is no longer supported and in the future EPIC courses will be changed to use some other MPI implementation.
[6] *Hyper Text Transfer Protocol* – communication standard for transferring HTML documents.
[7] *Multimedia Internet Mail Extension* determines the type of the content of the particular HTTP request or respond.

In the case where the generated EPIC command is processed by `epic_com`, the response from the EPCC web server will be a code `204` (`No Content`) and the browser will not take any action.

The Perl EPIC implementation is fully functional and works well but suffers from several disadvantages, the most important of which are:

- It is too dependent on the UNIX environment.
- It requires a Perl interpreter to be installed.
- Each request (particularly EPIC commands) is sent to the EPCC web server first and then is bounced back (wasteful of bandwidth and time).
- `epic_master` and `epic_com` are invoked with each EPIC command, which is time consuming.

## 2.4 A step forward – EPIC2000

The EPIC2000 SSP project [Ref. 1], which was run last year, was oriented towards finding a new implementation for the EPIC system to remove some of its weak points. EPIC2000 had a different, advanced architecture in comparison with the Perl EPIC version. Java was chosen as a new language for this implementation.

The main idea of EPIC2000 was to put a proxy server between the user's browser and the EPCC web server, see Figure 3. This proxy server would run on the user's machine and be responsible for parsing HTTP requests from the browser. It would be able to identify the EPIC command within a received HTTP request and reply back to the browser the proper response, without having to access the EPCC web server.

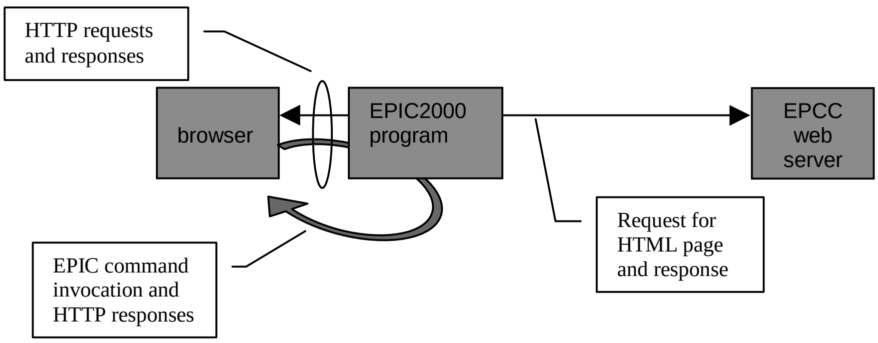



**Figure 3: Schematic representation of EPIC2000 HTTP requests processing.**

EPIC2000 contained the basic functionality needed for the proxy server but the majority of the EPIC functions were not implemented.  This solution was based on a good idea and initiated the work on the new project – EPIC2001. The EPIC2001 system was more advanced, it has implemented a major part of features required for the compatibility with the existing Perl EPIC version and has pushed the new implementation closer to the expected final package.

The main advantages of the EPIC2000 were:

- It was written in Java, which is more suitable for portability and hardware independence.

- It served as a basis for solving some of the weaknesses of the Perl EPIC implementation.
- It removed the network traffic needed for each request in the EPIC system.
- It made possible to integrate the functionality of the `epic_com` and `epic_master` scripts into one program.

### 2.5 The final stage – EPIC2001

EPIC2001 [8] system is intended to be a replacement for the existing EPIC Perl implementation. It has introduced a new reincarnation of the EPIC2000 architecture but has kept backward compatibility with the existing EPIC courses.

EPIC2001 uses a proxy server for the EPCC web server. This means that while running on the user's machine the proxy server will receive and parse HTTP requests addressed to it. If it identifies an EPIC command within the HTTP request, it will try to execute this command and send the proper response to the browser. If no EPIC command is found in the HTTP request, the browser will then retrieve the requested document from the remote web server. EPIC2001 generates its own HTTP request for the required remote document. After receiving the document from the remote server, EPIC2001 sends it onto the client browser as the response to the original HTTP request.

To be more precise, the EPIC2001 can identify the EPIC command by parsing the URI field of the HTTP header (request). The example of an HTTP request sent by browser:

```
GET /epic/hpf/ex7.html HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.03 (X11; I; SunOS 5.5.1 sun4c)
Host: localhost:2001
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

The browser is basically stating by this request [Ref. 6] that it wants to "GET" the document "/epic/hpf/ex7.html" (URI field) located on a host `localhost` listening on port `2001`. The URL for such a document should then be http://localhost:2000/epic/hpf/ex7.html. Since the EPIC2001 acts as a proxy server for http://www.epcc.ed.ac.uk, the real URL of the document becomes http://www.epcc.ed.ac.uk/epic/hpf/ex7.html. The given document is then requested by EPIC2001 and returned to the browser.

The point here is that when a user clicks on the hyperlink which represents the EPIC command (like /cgi-bin/epic/epic_com/edit&mpi/hello&.ex1.soln – as was described in section 2.2) the browser then generates an HTTP request in which the URI field contains this command. The proxy server can easily find the EPIC command in the request by matching it with pattern /cgi-bin/epic/epic_com/.

The features in EPIC2001 can be summarised as:

- It is implemented in Java, which is nowadays standard for multiplatform oriented applications.
- It removes the network traffic previously needed for each request in the EPIC system.

[8] The term EPIC2001 can be also understood to mean two different things. First of all it can represent a whole distance education package or it can simply signify the program implementing EPIC commands handling and execution. In the first case I usually use phrase "EPIC2001 system" in the text. The second meaning is usually referenced as "EPIC2001 program".

- It removes the time delay caused by the remote connection, the CGI program invocation and the local `epic_master` invocation for every request.
- It contains a big part of the EPIC functionality and integrates the `epic_com` and the `epic_master` roles into one program.

# 3 The EPIC2001 project

This part of document contains relevant information about project. It describes the aims of the project, design, architecture and implementation of EPIC2001 and notes any possibilities for future development. It is built on general information provided in the previous parts of the report.

In general, there was no need for detailed requirement analysis in this project and also overall architecture existed at the beginning of the project. The biggest part of effort was spent on implementation phase, then on finding and trying various alternatives to fulfil requirements and on the testing the product and its intermediary versions.

## *3.1 The requirements*

Initial status EPIC2001 package available in time of the beginning of the project included:

- Fully functional proxy server.
- Fully implemented code used for sending/receiving particular EPIC HTTP requests.
- Fully implemented code for saving/loading EPIC properties and code for dispatching the EPIC commands.
- Partially implemented code handling EPIC command.

The EPIC2001 SSP project was expected to fulfil these aims:

- To finish the addition of EPIC functionality into EPIC2001 package, literally several remaining EPIC commands were needed to implement. Examples follow:
  /cgi-bin/epic/epic_com/edit&hpf&MYHOSTS
  /cgi-bin/epic/epic_com/edit&mpi/ring&ring(\..*)
  /cgi-bin/epic/epic_com/edit&hpf/mandel&mandel(\.hpf|\.f90)
  /cgi-bin/epic/epic_com/epcccchimp
  /cgi-bin/epic/epic_com/make&DEM&par
  /cgi-bin/epic/epic_com/make&optimisation/matpower&run3&
  /cgi-bin/epic/epic_com/make&mpi/pingpong&run&PROCNUM
  /cgi-bin/epic/epic_com/skip&ga/Intro&1&anneal.c
  /cgi-bin/epic/epic_com/skip&mpi/ring&4&ring(\..*)
  /cgi-bin/epic/epic_com/skip_page&mpi/ring&1&ring(\..*)
  /cgi-bin/epic/epic_com/skip_page&hpf/golf&1&golf(\.hpf|\.f90)
  /cgi-bin/epic/epic_com/trans&epic/tarfiles/MPI-C-Exercises.tar.Z
  /cgi-bin/epic/epic_com/trans&/epic/tarfiles/data.tar.gz
  /cgi-bin/epic/epic_com/tutor
- To implement rest of the EPIC2001 with respect to its multiplatform use.
- To implement regular expression parsing and decomposition for EPIC commands.
- To implement or reuse the existing graphical user interface.

- To implement .jar files handling and transfer – it is expected that courses for EPIC2001 will be converted into this type of archive.
- To solve home directory path handling in transparent way.
- To convert the EPIC MPI course to NT platform by using a public domain MPI implementation.
- To test available EPIC courses with this new implementation.
- To replace and reimplement parts of given EPIC2001 code to fit these new version requirements.

## 3.2 The top level system architecture

The top level architecture of EPIC2001 system was partially given at the beginning of the project. It is determined by the use of the proxy server approach (as described previously).

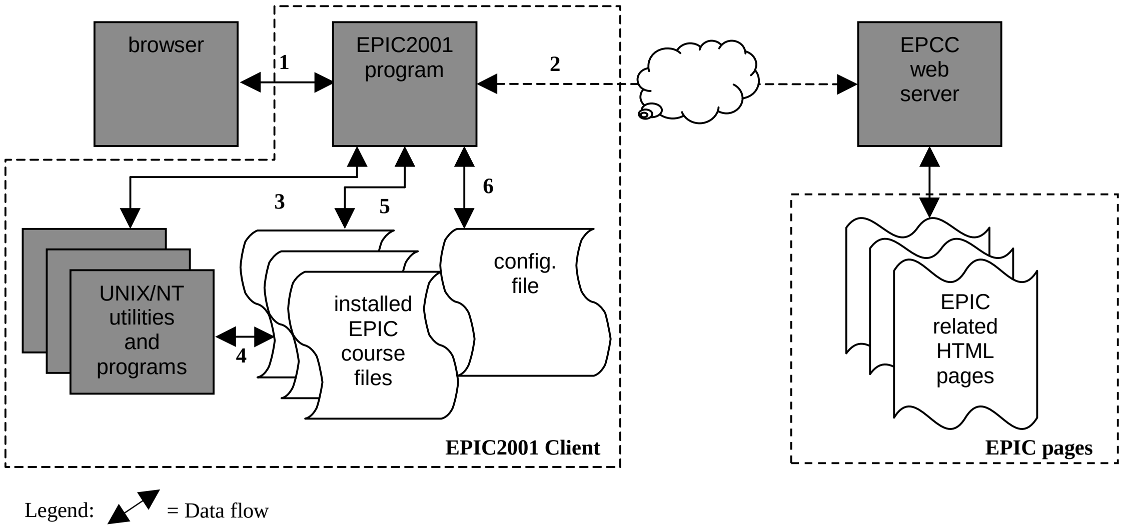



**Figure 4: Schematic diagram of the EPIC2001 system.**

Central part – the EPIC2001 program, concentrates the functionality of `epic_com` and `epic_master` Perl scripts. It directly communicates with the browser and web server (sending and receiving HTTP requests and responses). It can also execute various UNIX/NT programs according to user input or EPIC commands from browser. Invoked programs can read/modify EPIC course files – mainly exercise source files.

Data interchange, see Figure 4:

Data flow *1*: HTTP request/response – can contain embedded EPIC command.
Data flow *2*: HTTP request/response – cannot contain embedded EPIC command.
Data flow *3*: External program invocation, text output (if any) passed back.
Data flow *4*: External program reads/modifies/produces EPIC courses files. For example source files of exercises are read and modified by text editors, text files containing data for graphical interpretation are produced.

Data flow *5*: Create/delete/pack/unpack course files – EPIC2001 program handles also distribution of EPIC course files
Data flow *6*: Read/write to configuration file – EPIC2001 program is customised via configuration file, also various settings for exercise execution are contained here.

General external dependencies (UNIX):

- `make` program – used for building executables and executing exercises within EPIC courses. The majority of exercise files are source code files written in Fortran or C language. Each exercise contains a Makefile according to which final executables are to be built and executed. Makefiles are primary input files for the `make` utility.
- C/Fortran/other compilers – used to build executables from exercise source files.
- External programs run by user from the EPIC2001 command line.
- Original UNIX shell available on the system (`sh`).

General external dependencies (Windows)

- Equivalent UNIX's `make` program – used for building executables and executing exercises within EPIC courses. Currently, the Microsoft's `nmake.exe` utility is used.
- C compiler suite – used to produce executables from C source files of exercises. Microsoft's VisualC++ ver.5.0 is required (`cl.exe` and `link.exe`).
- External programs run by user from the EPIC2001 command line.

Course specific external dependencies (UNIX):

- Various other external programs required for the EPIC exercises (visualisation programs, special compilers…) – required by particular EPIC course.

Course specific external dependencies (Windows NT):

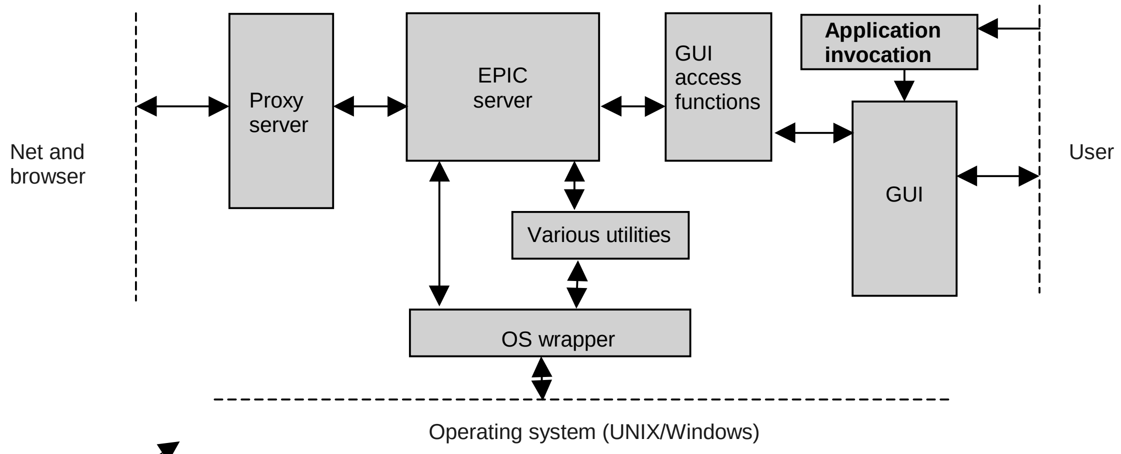
- WMPI1.1 [9]– MPI implementation for Windows NT, needed by the MPI course ported to Windows.

## 3.3 Detailed system architecture and design

### 3.3.1 EPIC2001 program

The primary implementation issue was the EPIC2001 program. EPIC2001 is internally divided into several modules, see Figure 5. Each module is responsible for a consistent set of program functionality. Communication between modules is done via functions invocations.

[9] The WMPI1.1 software is a freely accessible MPI implementation. It can be downloaded from http://dsg.dei.uc.pt/w32mpi/.




Legend: = Data flow

**Figure 5: EPIC2001 program module architecture.**

EPIC2001 program was implemented in Java and module names directly correspond to class names.

Modules of the EPIC2001 program:

*Proxy server* – mapped to `ProxyServer.java`. This module provides a net interface for the EPIC2001 program. It receives incoming HTTP requests from the browser, sends HTTP request to the web server and sends HTTP replies back to the browser. HTTP replies can contain the results of an EPIC command invocation. This part also includes simple security mechanisms.

*EPIC server* – mapped to `EpicComServer.java`. This module contains all functionality of the previous `epic_com` and `epic_master` Perl scripts and also new EPIC functionality according to requirements (.jar files handling…). It detects, executes and responds to all EPIC commands. It also loads/saves data to a configuration file. All the important information about the activity of this module (about commands invocation) are presented to the user via a GUI.

*Various utilities* – mapped to `Utilities.java`. Module contains various helper functions used by EPIC server module. This functions are generic, have more general meaning so they can be separated from server module. Here functions for string handling, simple regular expression decomposition… are located.

*OS wrapper* – mapped to `OSWrapper.java`. The main purpose of this module is to hide dependencies on particular operating system to enable multiplatform EPIC2001 use. Therefore it contains low level functions like OS type detecting, filenames and pathnames handling or external programs invocation.

*GUI access functions* – mapped to `GUIAccessible.java`. This module provides functions for accessing the `EPIC server` from the `GUI`. Such functionality is needed to obtain certain status information and settings from the `EPIC server`. This module also allows independence from graphical "user end" of an application to be achieved. EPIC2001 can be ported to systems, which do not support graphical interfaces, or the current GUI can be easily replaced with another one.

*GUI* – mapped to `GUI.java`. The functionality of this module covers mainly the provision of a graphical interface between the user and program. It allows the user to modify EPIC settings, to see the status information from `EPIC server` and to see the results of the EPIC commands execution. The user is also allowed to invoke his own programs and OS commands via GUI.

*Application invocation* – tiny module to invoke the EPIC2001 application. Does some command line argument parsing. It is mapped to `StartApp.java`.

### 3.3.2 The GUI

The graphical user interface of the EPIC2001 program is derived from the GUI of Perl implementation of EPIC, see Figure 6. It has a console window primarily used for writing the status information of any EPIC2001 activity. There is also a command window, which displays text output from programs run by the user via the command line window of the GUI.

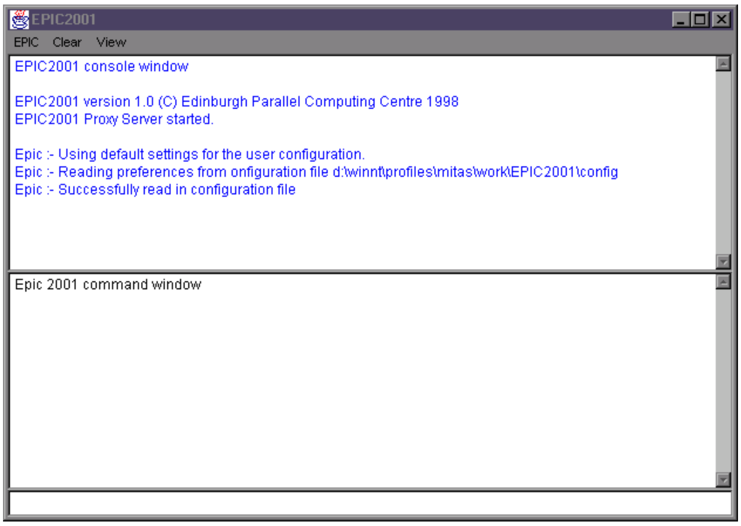



**Figure 6: The EPIC2001 GUI (console, command and command line windows visible)**

The menu of the GUI consists of three submenus – *EPIC*, *Clear* and *View*. *EPIC* submenu contains entries *Settings* – to set various EPIC and GUI parameters, *Help* – to provide help information for EPIC2001 system and *Close* – to terminate the application. The *Clear* submenu consists of entries *Console window* – to clear content of console window and *Command window* – to clear the content of the console window. Finally the *View* submenu contains entries *Console window* – to show/hide a console window, *Command window* – to show/hide command window and *Command line* – to show/hide command line window.

EPIC2001 GUI is resizable, it preserves the position on the screen after closing, the user can also set the number of lines after which content in a particular window will be deleted. The command line window also contains one-command history.

### 3.3.3 Configuration file

The structure of entries in EPIC2001 configuration file is the same as in the Perl implementation of EPIC (section 2.2). The settings of the EPIC2001 are a superset of the EPIC settings. Example of a minimal configuration file is as follows:

```
xLocation 2                 //Y position of the GUI window
consMaxRows 100             //maximal number of rows shown in a console window
commMaxRows 100             //maximal number of rows shown in a command window
PROCNUM 4                   //number of processes
notepad*Editor* notepad     //command for editor invocation
editor vi*Editor*           //user preferred editor to invoke
userDefined*Editor* "D:\Program Files\Lemmy\LEMMY.EXE"
                            //user defined editor to invoke
make nmake                  //command to invoke the make utility
xedit*Editor* xedit         //command for editor invocation
set yes                     //flag to determine if properties were set
editorTag *Editor*          //tag determining the command for editor
yLocation 9                 //Y position of the GUI window
vi*Editor* xterm -xrm 'XTerm*Iconic:false' -e vi
                            //command for editor invocation
exec_f77 f77                //command for Fortran77 compiler invocation
exec_hpf hpfff              //command for HPF Fortran comp. invocation
write*Editor* write         //command for editor invocation
ue*Editor* xterm -xrm 'XTerm*Iconic:false' -e ue
                            //command for editor invocation
emacs*Editor* emacs         //command for editor invocation
```

## *3.4 The implementation*

### 3.4.1 The EPIC2001 program

The EPIC2001 program was implemented in Java, but due to JRE internal errors it requires different versions of JRE on different platforms.

JRE required (UNIX) – version 1.1.6 (provided by Sun)
JRE required (Windows) – version 1.2beta4 (provided by Sun)

The following diagram denotes the class hierarchy of the entire EPIC2001.

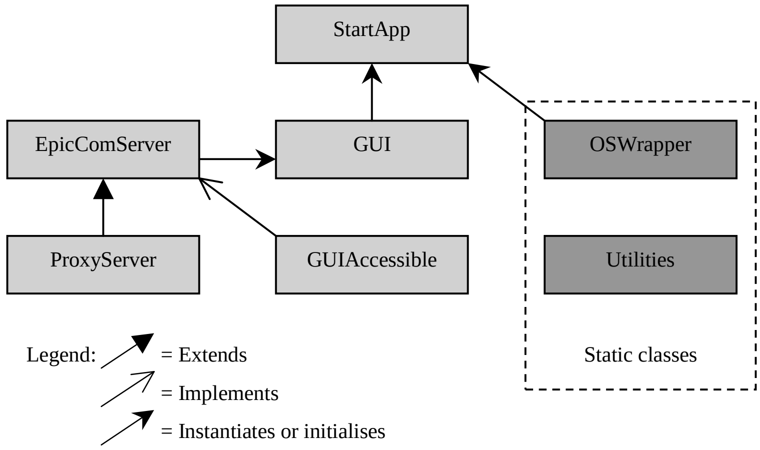



**Figure 7: EPIC2001 class hierarchy.**

EPIC2001 is invoked by typing the following command in DOS or UNIX shell:

`java EPIC2001.StartApp` *`EPIC2001_home_path PATH_content port_number web_server_name`*

Arguments:

- *`EPIC2001_home_path`* – required – EPIC2001 installation directory (note that EPIC2001 is the name of the package, so that in must be installed in a directory named EPIC2001).
- *`PATH_content`* – required – content of the PATH environment variable.
- *`port_number`* – not required (default = 2001) – port on which the EPIC2001 proxy server will be listening on.
- *`web_server_name`* – not required (default = www.epcc.ed.ac.uk) – web server containing the EPIC related HTML pages.

The implemented functionality of EPIC2001 is slightly different from the original EPIC implementation. The differences were caused in part by the initial requirements and also by avoiding several features of EPIC, which are not needed anymore. The differences can be summarised as (the reason [10]for the given difference is noted in squared brackets):

- There is no checkpoint file for loading user activity [Deprecated].
- There is just MPI course supported on Windows platform [Requirement] – and just C exercises of this course are supported [Lack].
- Makefiles of the MPI must be changed – new multiplatform Makefiles were written. Makefiles of other courses can stay unchanged, however it is strongly recommended to rewrite them according to new MPI Makefiles [Requirement].

[10] The reasons can be divided into several groups: [Deprecated] – not needed and not supported anymore, [Requirement] – caused by the requirements of the project, [Feature] – characteristic or innovation, [Lack] – insufficient implementation.

- WMPI1.1 has to be properly installed on NT machine to build and execute MPI exercises [Feature]. The `vwmpi.lib` is used to build user executables.
- WMPI1.1 runs on NT NUMPROC + 1 processes instead of NUMPROC [Feature].
- EPIC2001 program must be invoked manually before using any EPIC command [Feature].
- User have to access EPIC2001 proxy server instead of EPCC web server by typing URL: http://localhost:2001/epic/ (assuming that server listens to default port `2001`). Another way is to configure the browser to use the EPIC2001 like proper proxy server. A disadvantage of the second approach is that EPIC2001 will have to be started with each invocation of the browser and also on high-speed connection to Internet it will become the bottleneck [Feature].
- There is a problem when receiving HTML documents from a remote directory given by URL not terminated by a slash – *Moved Permanently* response [Lack].
- EPIC2001 implements *.jar* files transfer and handling instead of the original *compress*-ed and *tar*-ed archives containing the course files for installation at the client machine [Feature]. This is the reason why local course files cannot be transferred and installed by EPIC2001 as things stand at the moment. Archives on the EPCC web server containing these files will have to be converted into *.jar* archives.
- `epccchimp` EPIC command is not supported [Deprecated]. EPCC will not use CHIMP package in the future.
- Implementation of tutor function wasn't finished completely – archive to send will be created but not mailed to EPCC [Lack].

### 3.4.2 The GUI

There are also differences in the GUI functionality in comparison with the existing EPIC Perl implementation. Following list summarises them:

- The results of the EPIC command execution are written into the console window. This is caused by the fact that the EpicComServer only has one way of communication with the GUI – that is string output to console window. This is not necessarilly disadvantage, in the previous Perl implementation there were separated windows because of the way in which the EPIC commands were executed. Now the outputs are strictly separated – EPIC2001 uses the console window, programs run by user use command window [Feature].
- Programs invoked via the EPIC2001 command line window are blocking. This means that the next command can only be entered after the execution of previous one has finished [Feature]. The program run by the user and the EPIC command of course can run in parallel. There is also parsing of the user command done on the UNIX machines, to disable background execution of the command (it caused problems) [Feature].

### 3.4.3 Configuration file

There were also changes done to the configuration file. These include:

- New entries for storing GUI properties (`xLocation`, `consMaxRows`, `commMaxRows`, `yLocation` – see 3.3.3 for details) [Feature].
- New keys for the EPIC2001 functionality (`make`, `set` – see 3.3.3 for details) [Feature].
- The keys used to specify the editor invocation have been changed. Now such a command ends with an `editorTag` string [Feature]. This is required for editor entry recognition while building list of available editors in GUI to enable user to set preferred editor.

### 3.5 Results summary

The EPIC2001 implementation was realised according to a given set of requirements. It was also successfully tested on Windows NT 4.0 (see Figure 8) and Solaris 2.5.1 (see Figure 9) platforms. On the Solaris machine the entire EPIC courses were tested, on NT just the MPI course – as was proposed. The following pictures shows an example of running the same MPI exercise within MPI course on both platforms.

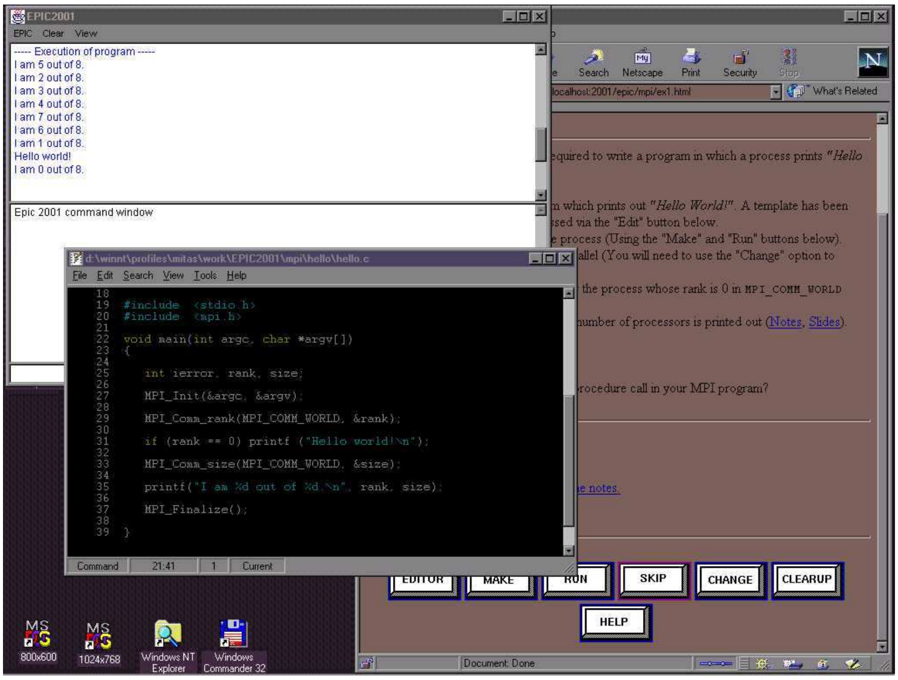



**Figure 8: EPIC2001 executing the Exercise 1 of MPI course (WinNT).**

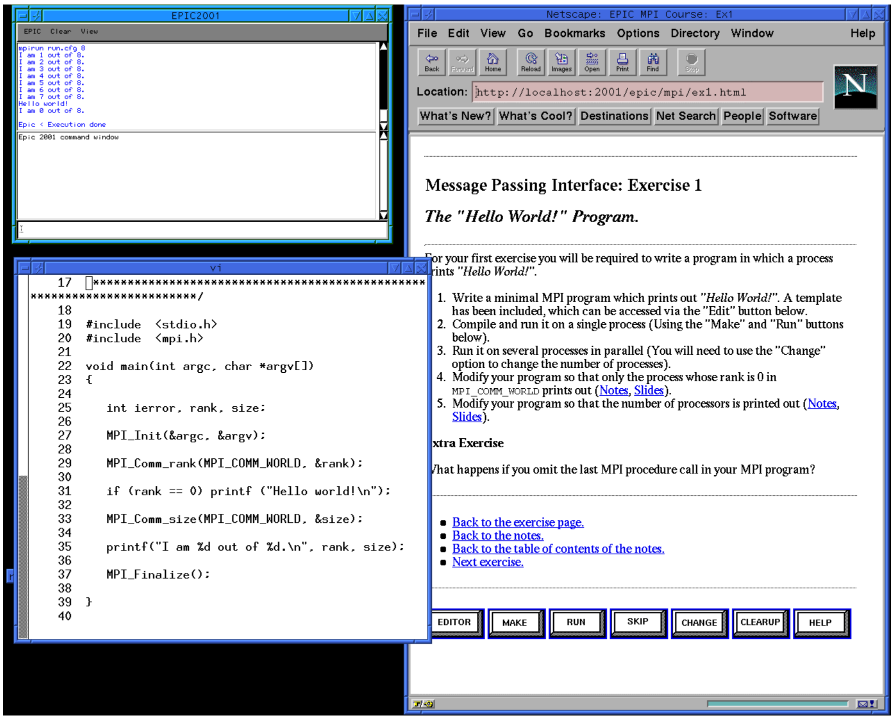


**Figure 9: EPIC2001 executing the Exercise 1 of MPI course (Solaris).**

The following table summarises of the test results.

| Course / Platform | MPI | HPF | PO | CFD | MG | MD | DPP | GA |
|---|---|---|---|---|---|---|---|---|
| Solaris | OK | X | OK | OK | OK | OK | OK | X |
| WinNT | OK | N/A | N/A | N/A | N/A | N/A | N/A | N/A |

Legend: OK – course run without difficulties.
X – course failed to test.
N/A – course not available on given platform.

Notes: The HPF course couldn't be tested, there were not HPF compilers available under my account.
The GA course cannot be used with EPIC2001 because the EPIC commands are hardcoded to EPCC web server.

Configuration: WinNT – Netscape Communicator 4.5b1, JRE 1.2beta4, NT Workstation version 4.0.
Solaris – Netscape Navigator 3.03, JRE 1.1.6, Solaris 5.5.1.

**Table 1: EPIC courses test results.**

# 4 Future development

At the present EPIC2001 is capable of running on both UNIX and Windows platforms. It introduces several new features and has a GUI implemented. The following set of recommendations would be useful for extending the EPIC2001 functionality.

- It would be good to incorporate or implement a Java based text editor into EPIC2001.
- It would be good to add "activity indicator" to EPIC2001 GUI – this would be useful on slow machines where the program response time is poor.

- It is also important to fix all weaknesses and bugs described in section 3.3.3 to improve the functionality.
- A more demanding improvement would be to provide also some tools for the creation of new EPIC courses.

# 5 References

**[Ref. 1] Radwan G.,**
EPIC2000 – The final report of the project

**[Ref. 2] Yau H.,**
“EPIC2001 source code”

**[Ref. 3] Flannagan D.,**
JAVA in a nutshell, O’Reilly & Associates, Inc., 1997

**[Ref. 4] Cornell G., Cay S., Horstmann,**
Core Java; Fundamentals, Prentice Hall, 1997

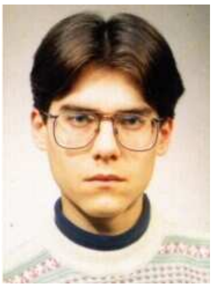
**[Ref. 5] Horstmann, Cay S.,**
Core Java; Advanced features, Prentice Hall, 1997

**[Ref. 6] Wong C.,**
Web client programming with Perl, O’Reilly & Associates, Inc., 1997

## 6 Acknowledgements

I would like to thank my supervisors for the excellent support received while working on the EPIC2001 project, namely Mario Antonioletti and Hon W Yau. My thanks go also to the EPCC staff, especially to Catherine Inglis who was the co-ordinator of the SSP programme. Things have gone very well, I have enjoyed it a lot. Have a luck (and fun).

## About the author



I'm a student of Computer Science at the Slovak Technical University in Slovak Republic. I have finished the 5th year of my studies. I am fond of software engineering (of course), good books, good music, good paintings, vegetarianism, volleyball and the rest of the world.