SGPEMv2 User Manual

for version 1.0, 18 September 2006

Filippo Paparella (ironpipp@gmail.com)
Paolo Santi (psanti@studenti.math.unipd.it)
Matteo Settenvini (matteo@member.fsf.org)
Marco Trevisan (evenjn@gmail.com)
Djina Verbanac (betalgez@yahoo.com)
Luca Vezzaro (lvezzaro@studenti.math.unipd.it)

This is SGPEMv2 User Manual (version 1.0, 18 September 2006).

Table of Contents

History

2006, September 12th — Luca Vezzaro
Updated section "From the commandline"

2006, September 9th — Luca Vezzaro
Written documentation for section "The Schedulables/Requests tree" and section "The Resources list"

2006, September 8th — Luca Vezzaro
Written documentation for section "Overall view of the main window"

2006, September 8th — Matteo Settenvini
Update chapters about building and installation. Rewrite some of the chapter about extending SGPEMv2 with custom CPU policies, and add a more complex example. Document interfaces exported to Python. Quickly describe built-in scheduling policies.

2006, September 7th — Luca Vezzaro
First attempt at expanding the manual structure with the stuff we'll need in the forthcoming beta testing

2006, March 10th — Djina Verbanac
Added chapter Writing new policies

2006, March 9th — Djina Verbanac
Add chapters Overview of SGPEM and Starting with SGPEM.

2006, January 26th — Matteo Settenvini
Add subsection about how to generate code documentation via Doxygen.

2005, December 11th — Matteo Settenvini
Added full license text.

2005, November 8th — Matteo Settenvini
First draft of this document.

1 Overview of SGPEM

1.1 Description and aims

SGPEM is an Italian acronym, standing for "*Simulatore della Gestione dei Processi in un Elaboratore Multiprogrammato*" (in English, "*Process Management Simulator for a Multitasking Computer*"). It was initially developed for use inside the "Operating Systems" teaching, part of the Computer Science course of the University of Padova, Italy. The aim of SGPEM is to provide an easy-to-use environment for simulating process scheduling policies, and for assigning resources in a multitasking computer. SGPEMv2 is an educational software, and it can help students to better understand the functionality of operating systems.

1.2 How to read this manual?

We recommend that you read the manual following the the structure that we layed out for it. You will be gently led trough Installation, Configuration and Usage of SGPEMv2. If you find yourself in trouble reading the manual, please don't hesitate to contact us at `swe@thgnet.it`.

1.3 Reporting Bugs

We welcome bug reports and suggestions for any aspect of the SGPEM v2 system, program in general, documentation, installation... anything. Please email us at `swe@thgnet.it`. For bug reporters, include enough information for us to reproduce the problem. In general:

- version and number of SGPEM v2.
- hardware and operating system name and version.
- the content of any file neccesary to reproduce the bug.
- description of the problem and any erroneous output.
- any unusual option you gave to configure.
- anything else you think might be helpful.

If you are ambitious you can try to fix the problem yourself, but we warmly recommend that you read the Developer Manual first.

1.4 Features

Main features are:

- You can both use SGPEMv2 via commandline or via a graphical user interface. For more information see Section 4.2.1 [SGPEM Commands], page 17.
- You can schedule threads or processes, and threads can make atomic request to one or more resource at each instant of the simulation.
- It is displayed an Holt graph of the resource allocation in the graphical version.
- Statistics are shown at each simulation step, separated for processes and threads.
- You can easily jump at different instants of the simulation, seeing what happened at a given moment.

- Editing an existing simulation is easy and quick.
- Savefiles are by default written in XML, making it easier for external tools to provide compatibility with SGPEMv2.
- You can write your own policies using python, or easily extend SGPEMv2 with you own plugin to add more scripting languages. For more information see Section 5.1 [Writing new policies], page 21.

2 Installation

2.1 Prerequisites

Some software is needed in order to build and install SGPEM on your personal computer. You will have the need of different pieces of software installed, whether you are a developer, a user building it from sources, or just a user that's running the binary a packager has given to him.

And if you find this section misses something / it lists the wrong version of a program, please let us know!

2.1.1 Runtime dependencies

To run SGPEMv2, you require:

Gtkmm >= 2.8 with Cairo support
: The popular C++ jacket for the even-more popular GIMP ToolKit. We use Cairo to draw our custom widgets.

Python >= 2.3
: We use Python to let the user write her own policies in a simple and complete language.

libXML2 >= 2.6.10
: An XML library we use to save and load files to/from disk.

2.1.2 Building from source

Other than the runtime dependencies, you'll need:

SWIG >= 1.3.21
: SWIG generates the C++ sources needed to build a module that Python can use, starting from a simple interface specification.

2.1.3 Developers

Other than the tools needed by users building from sources, you'll need:

GCC with C++ support
: as well as the other standard GNU binutils and tools: make, sed, ld... GCC version >=3.4 is highly recommended. Please don't report compiling-related problems with any previous version. There are some known issues with certain versions of GCC 4.0. See Section 2.2 [Building], page 5.

Automake >= 1.9
: We use a single '`Makefile.am`' to avoid recursive make. Older versions of automake didn't play right with it. See `http://aegis.sourceforge.net/auug97.pdf` for the motivations that led to this choice.

Autoconf, libtool, autopoint ...
: The standard autotool family.

Subversion >= *1.2*
If you need to update the sources from our repository, or commit your changes, you'll need Subversion built with SSL support.

Dejagnu >= *1.4*
The testsuite framework we use as a platform for running tests.

2.2 Building

To ensure a clean build, follow these steps:

```
cd <the package root directory>
mkdir =build
cd =build
CXXFLAGS="what you want" ../configure --prefix=/usr/local
```

This will check you have all the needed software installed.

Choose good `CXXFLAGS` to optimize your build. For example, on my machine, I would use:

```
CXXFLAGS="-O3 -pipe -march=pentium4" ../configure --prefix=/usr/local
```

Being a developer, though, if I had to debug SGPEM, I would type:

```
../configure --prefix=‘pwd‘/../=inst --enable-debug
```

Please note that those around "pwd" are backticks, and not normal apostrophes.

Warning: at the moment, we are aware that passing '`--disable-shared`' to configure doesn't work. We'll look into it sooner or later, but in the meantime just build shared libraries.

Once succesfully configured SGPEMv2, just type:

```
make
```

Some versions of GCC 4, usually those before the 4.1 series, present some problems with the newly-added visibility support for DSO object symbols. For example, OpenSuSE 10.0 is known to have such issues. If you encounter problems during building and in linking stage about unresolved symbols in libraries, please re-run `configure` with the '`--disable-visibility-support`' option. You'll then have to run `make clean && make`.

Upon a succesful build, you can install SGPEMv2 just by hitting:

```
su -c "make install"
```

Root password will be required (of course, if you're installing it with a prefix placed inside your home directory, you won't need administrative rights, and just "`make install`" will sufficit).

See the "'INSTALL'" file in this folder for an overview of other (less common) autoconf options.

2.2.1 Generating API documentation

We added Doxygen support to the project. If you've installed it, you can simply run `make apidox` from the package top source directory. The documentation will be outputted into the '`${BUILD_DIR}/docs/API/`' dir.

If you'd like to generate nicier inheritance graphs, you've just to install `dot`, part of the *Graphviz* package. If you didn't have it previously installed, you may need to re-run `configure`.

3 Basics

3.1 The Scheduler

From the scheduler's point of view, the simulated environment is populated by processes and resources. Processes are spawned at diffent instants and compete for the CPU and other resources until their termination.

Processes have an arrival time, i. e. an instant at wich they are spawned, and a priority.

Our application simulates the scheduling of threads, not the scheduling of processes. Anyway, it is possible to simulate processes scheduling simply placing one single thread within each process, and hiding thread details on the GUI.

In SGPEM, a process is quite just a container of threads. Threads have a required cpu time, a priority within the process, and an arrival time delta. The arrival time delta of a thread is relative to the execution time of the parent process, and not to the arrival time of the parent process.

The scheduler's task is to assign the cpu and the other resources to the processes. Both resources and CPU are mutually exclusive, meaning that no two processes may use them at the same time.

A thread may raise requests at any time of its execution: a request has a raising time delta, which is relative to the execution time of the owner thread, and not to the arrival time of the owner thread.

A request specifies a set of resources and the time they are requested for. The specified set of resources will be acquired atomically, meaning that either all of the requested resources is given to the thread, or none of them is.

A thread may raise any number of requests at any instant. Requiring four resources may be done either atomically, specifying one request with four separate subrequests, or non-atomically, specifying four requests with one subrequest each. A subrequest is the specification of which resource and for how much time.

Resources have multiplicity, or places. A resource with two places acts like two indistinguishable resources.

3.2 Policies

3.2.1 What is a policy in SGPEM?

A policy is a rule used by the scheduler to decide which thread should run next. Our scheduler needs two different policies to perform this choice: one is called a cpu (scheduling) policy and the other is called a resource (scheduling) policy.

3.2.1.1 CPU Scheduling Policies

The first, from now on called simply "policy", is the rule telling which of the ready (or running) threads is the best candidate to get the cpu. For example, the FCFS policy is a rule which tells that, among the ready threads, the one which asked the CPU first is the best candidate. The Lottery policy is a rule which tells that, among the ready threads, one chosen at random is the best candidate.

Being the best candidate means to get the CPU and try to run: anyway, getting the cpu does not mean to be able to run: a thread may need a resource to complete its work, and mutually exclusive resources may be locked by other threads. In this event a thread is said to raise a request for some resources, and to get blocked by those requests.

3.2.1.2 Resource Scheduling Policies

The second policy is the rule telling, for each resource, which of the raised requests are the allowed to be satisfied, according to the places offered by the resource. For example, the FIFO resource policy is a rule which tells that, among the raised requests, the ones which came first are allowed to be allocated. An other example, the Priority policy is a rule which, roughly speaking, tells that, among the raised requests, the ones having higher priority are allowed to be allocated.

SGPEM provides some resource policies, but it does not allow the user to create its own. Like cpu scheduling policies, resource policies are parametric, altough at the moment none of the included is. Resource policies are largely dependant on the mehcanism of the scheduler, and since is very complex to understand the mechanism of scheduler, it would be wasteful to provide an extension mechanism for resource policies: the user willing to implement a new resource scheduling policy would better understand and adapt the SGPEM source code.

3.2.1.3 Policy Parameters

A policy in SGPEM is in general a parametric rule: this means that the user should set some parameters to actually use the policy. Parameters are either integer, float or string values, which further specify the behavior of the policy: for example, the round-robin policy needs the user to choose the length of a time slice. Parametric policies always provide default values for their parameters, thus the user is not forced to set them manually. (see gui_set_policy)

3.2.2 What kind of policies are there?

SGPEM defines four classes of policies, and the scheduler uses different kinds of policies in different ways. The four kinds are: Simple, Time-sharing, Preemptive, Preemptive and Time-sharing.

3.2.2.1 Simple policies

Simple policies may change the running thread only when the running one has blocked or has terminated. A simple policy is allowed to change the running thread at any instant during the simulation, replacing it with the best candidate among the set of all the ready threads.

3.2.2.2 Time-sharing policies

Within SGPEM, a policy is said to be time-shared when the policy may change the running thread after it has been running for a full time-slice (or time quantum). The size of the time-slice is supposed to be fixed, and varying the size of the time-slice during the simulaiton is possible, altought not very useful. A time-sharing policy is allowed to change the running thread only when it has exhausted its quantum, or it has blocked, or it has terminated, replacing it with the best candidate among the set of all the ready or running(*) threads.

* At the moment any running thread which used up its quantum is set to ready, therefore there is no running thread to choose when a time-sharing policy is used.

3.2.2.3 Preemptive policies

Within SGPEM, a policy is said to be preemptive (or priority-preemptive, too) when the policy may change the running thread for priority reasons. A preemptive policy is allowed to change the running thread at any instant during the simulation, replacing it with the best candidate among the set of all the ready or running threads. Note that this meaning of the adjective "preemptive" may not match the one found in your favourite operating systems reference book.

Actually, our application does not check if the preemption is done for priority reasons, so one could, in principle, implement time-shared policies without specifying a fixed size for the time slice, i. e. without declaring the policy as time-shared. Time-sharing may be implemented using an internal counter, relying on the fact that a preemptive policy is called exactly at every instant.

3.2.2.4 Preemptive and Time-sharing policies

These policies are used by scheduler roughly in the same way as preemptive policies are.

Note that altough this distinction is enough to understand most of the common policies, SGPEM is not that simple (wasn't it simple?). The actual implementation does not partition the space of policies in four classes: a real SGPEM policy may in fact dynamically "change its class", thus not fit in any of the previously listed.

For using full-blown policies, advanced users should look directly at the mechanism itself.

3.2.3 Built-in policies

3.2.3.1 CPU scheduling policies

FCFS: First come first served

> The first thread to arrive to the CPU will run until it ends. This policy never pre-empts; it is probably the simplest of them all.
>
> This policy has no options to configure, too.

SJF: Shortest job first

> The thread with the shortest required CPU time will run until it ends. If '`Is pre-emptive?`' is set to true ('`1`'), given that a thread requiring less than the remaining time of the current running thread arrives at the CPU, the latter will pre-empt the former.
>
> In this case, the policy is also called "Shortest Remaining Time Next".
>
> You can configure if you want this policy to be pre-emptive or not.

RR: Round Robin

> This policy executes a thread for a given amount of time (the time-slice value), and then puts it at the end of the queue. It does not pre-empt before the end of the time slice, since it doesn't take priority in account. Use "RR priority" for that.
>
> You can configure the duration of the time slice.

RR priority

No lower priority thread can run if a higher priority thread exists. If pre-emptive by priority, a higher-priority thread becoming ready, even in the middle of a time slice, will pre-empt the running thread. Else, the time slice will have to end before the higher-priority thread can run.

You can configure if this policy is preemptive or not, and the duration of the time slice.

Lottery scheduling

Every time slice, a thread will be selected from the ready queue by random. This policy does not pre-empt before the end of the time slice.

3.2.3.2 Resource scheduling policies

First in first out

A resource policy which satisfies earlier requests before older ones.

This policy has no options to configure.

Last in first out

A resource policy which allows a request to be immediately allocated if there is enough space.

This policy has no options to configure.

Higher Priority First

A resource policy which satisfies higher priority requests before lower priority ones.

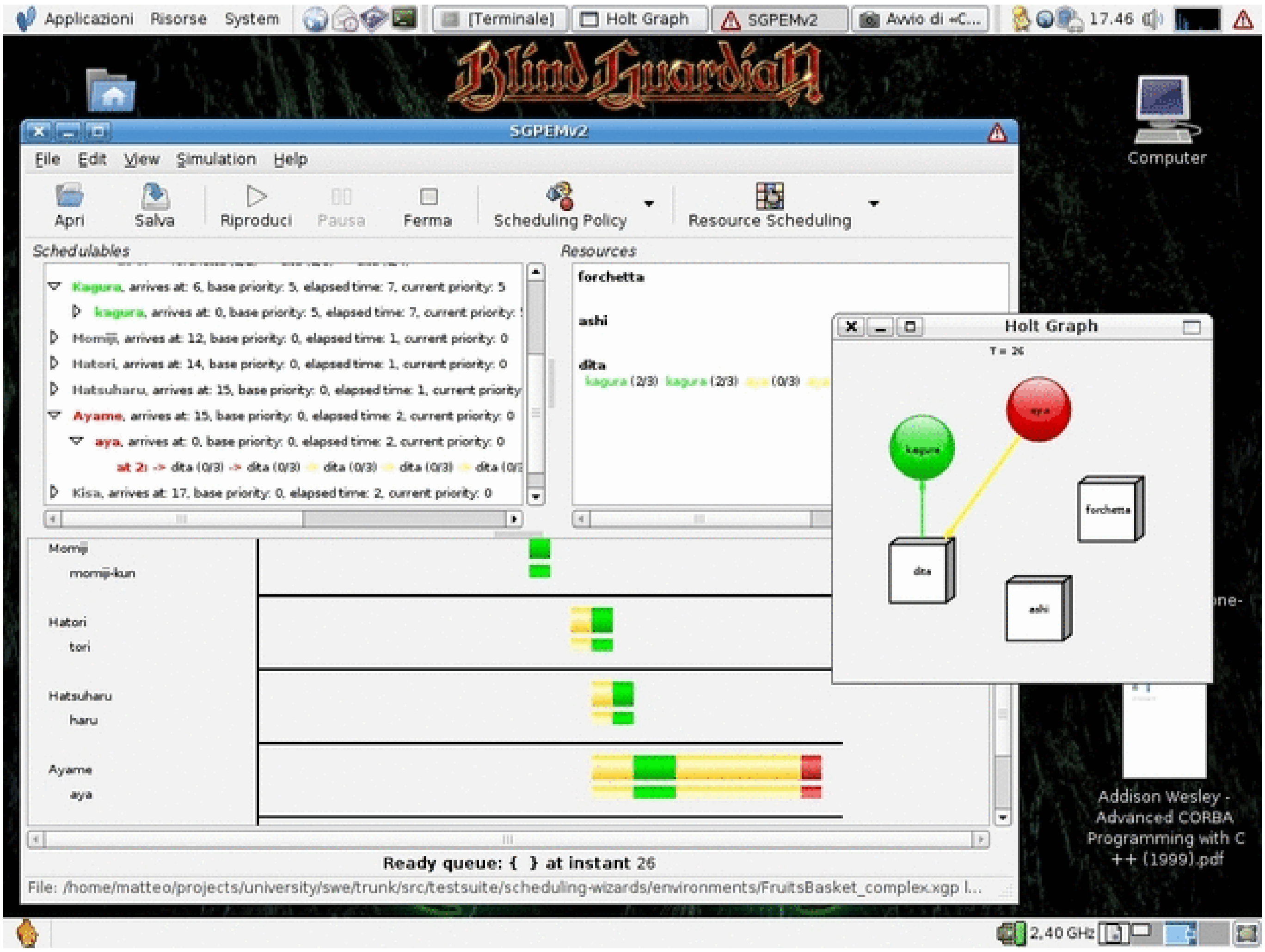
Note that a thread with priority 0 has an higher prioriy than a thread with priority 5.

This policy has no options to configure.

4 Using SGPEM

4.1 From the GUI

4.1.1 Overall view of the main window



Just below the menus, there's the toolbar. The purpose of most toolbar buttons is easily understood. For example, you can instantly change the current scheduling policy by using the menu just to the right of the "Scheduling Policy" toolbar button. Similarly, you can do the same with a resource allocation policy. The aforementioned "Scheduling Policy" and "Resource Scheduling" toolbar buttons can be used to configure the policy's parameters, if there are any.

To know more about the other toolbar buttons, such as "Pause", "Play" and "Stop", see Section 4.1.7 [Controlling the simulation], page 17.

Normally, the window is split into three sections.

- The top left section is briefly called the "Schedulables tree", every entity, except resources, in the SGPEMv2 is shown and edited in this tree view. The interface of this widget is straightforward, but in case you need to know more about it, see Section 4.1.2 [The Schedulables/Requests tree], page 12.
- The top right section is the resources list, you can interact with it in the same way you do with the Schedulables tree. We won't get into the details here, as there is Section 4.1.3 [The Resources list], page 13 for this widget.
- Finally, the bottom section contains the "Simulation vidget", which displays how the scheduling is proceeding. This widget is too complex to be described here, so we'll leave that to Section 4.1.4 [The Simulation widget], page 13.

Well, in fact that's not all, folks. There's also the "Holt graph", which is displayed in a separate window, so it doesn't steal precious window space to the simulation widget, and also because you may not need it if you don't use resources and/or requests in your simulation. For more information on this widget, see Section 4.1.5 [The Holt graph], page 15.

4.1.2 The Schedulables/Requests tree

This widget is used to add/edit/remove processes, threads and requests. To perform an operation on it, simply right-click, and a context-sensitive menu will popup.

Each tree level is dedicated to a specific entity:

- The first level is for **processes**
- The second level is for **threads**
- The third level is for **requests**

Right-clicking on any location over the tree will always allow you to add processes, while to add threads or requests you must select a process or a thread, respectively. To remove or edit an entity simply select it, and the popup menu will contain the remove or edit operation specific for that entity.

Anyway, these functionalities are only useful for a stopped simulation. While the simulation is not in a stopped state, a lot of dynamic information is displayed by the widget.

Let's begin by describing what's the meaning of the colors used to highlight the entities' name:

- **Light Grey** is used for "future" processes, threads, requests and subrequests. "future" means an entity in the real world will still not exist, since it will "arrive" at a time greater than the current instant
- **Green** is used for running processes, threads and for allocated requests and subrequests
- **Yellow** is used for ready processes, threads and for allocable requests and subrequests
- **Red** is used for blocked processes, threads and for unallocable requests and subrequests
- **Dark Grey** is used for terminated processes, threads and for exhausted requests and subrequests

Anyway, to improve readability, the state is also written in the second column of the view.

The dynamic display for processes and threads simply consists of their "elapsed time"/"required time" (between parenthesis), and a "current priority" field, which is obviously their dynamic priority which may change if the scheduling policy decides to do so.

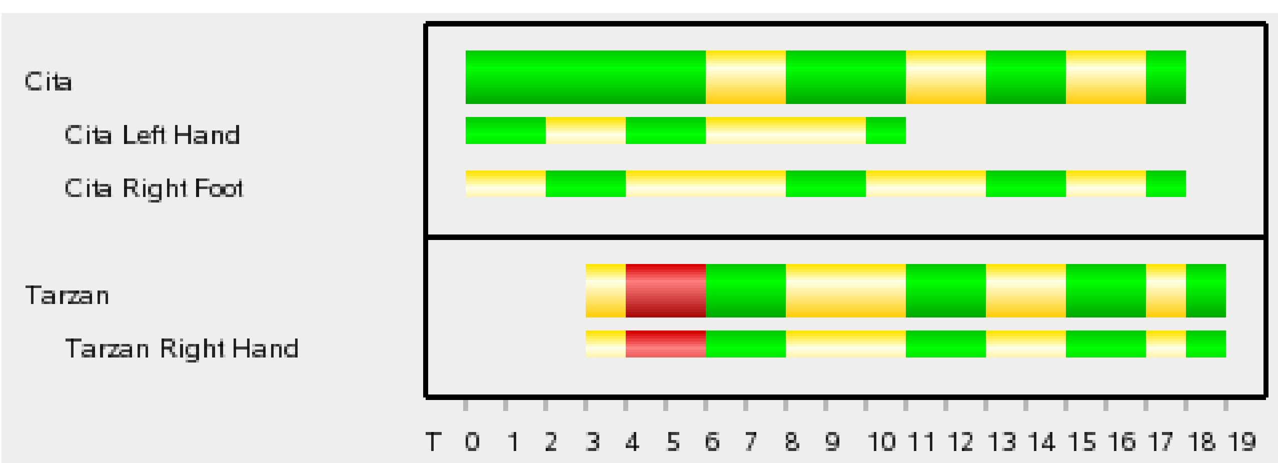
Probably the format used to display requests is a bit less trivial (yes, I'm sarcastic), but since a request has no additional information other than its state, it makes sense to condense requests and its associated subrequests on a single line.
So the color of the **at <n>:** represents the state of the request, the `<n>` being the instant at which the request is raised.
Then there are a series of subrequests, which are displayed as `->` (arrows), followed by a colored resource name and two numbers separated by a slash. The color of the resource represents the state of the subrequest, and the numbers between parenthesis are its "elapsed time"/"required time".

4.1.3 The Resources list

You can interact with this widget in the same way you interact with the Section 4.1.2 [The Schedulables/Requests tree], page 12, but since it's a plain list, not a tree, it's much more simpler. As you may have guesses, since a resource has no elapsed and required time, the numbers between parenthesis must be something else. And you are right! The numbers displayed just after the resource name are the "allocated"/"places", that is, the number of subrequests for that resource currently allocated "over" the number of places of the resource.

So let's get to the hot stuff: when the simulation moves from the boring stopped state to a running or paused state, below each resource will be displayed the subrequests queue. Since a subrequest has no name, the name of the thread owning that subrequest will be displayed, instead.
As if that wasn't cool enough, the thread name in the queue is colored accordingly with the state of the subrequest!

4.1.4 The Simulation widget



The simulation graph, as his name tell, show graphically the simulation progress along the time.
It represent the processes status at each instant from the simulation beginning to the actual one.
Into the graph is possible to view the processes only or both processes and threads.

Watch out: this graph illustrates the *past*. After each simulation step is gone, the corresponding processes'/threads' states are drawn.

The graph is divided in three areas:

- At left there are the processes (and optionally threads) names list
- From center to the right take place the graphical area
- At the bottom there is the time ruler

The Processes/Threads names list

Here each process is listed in insertion order.
If the thread visualization is enabled, below every process is shown a list of his threads.

The graphical area

It's a rectangular region wich contains some horizontal bars. Each bar correspond to a process or thread; the processes' bars are fat, the threads' are thin.

The bars are composed horizontally to show the story of each process and thread. If the process (thread) state change, and this is the rule, the corresponding bar change color.
As default the colors are: green, yellow, red.

- **Green** is used for running processes/threads
- **Yellow** is used for ready processes/threads (waiting to run)
- **Red** is used for blocked processes/threads (waiting for a resource)

The bar starts when the process or thread begin, ends when it die.
The length of the bar correspond to the time life of the process or thread.

The time ruler

Below the graphical area there is a time ruler from 0 to the current instant.
The last represented time is the *past* instant.

The first click on play button will show only notch 0 and no process bars.
At the second time, on the ruler, will be notches 0 and 1 and eventually the squares corresponding to living processes or threads.
The other clicks... are all the same!

How to show/hide threads

With the menu item `"Show/Hide Threads"` under the `"View"` menu the user can enable or disable threads visibility.

Scaling the graph

The user can select a scaling mode to view the graph.
This option is available with a popup menu right clicking in client area.

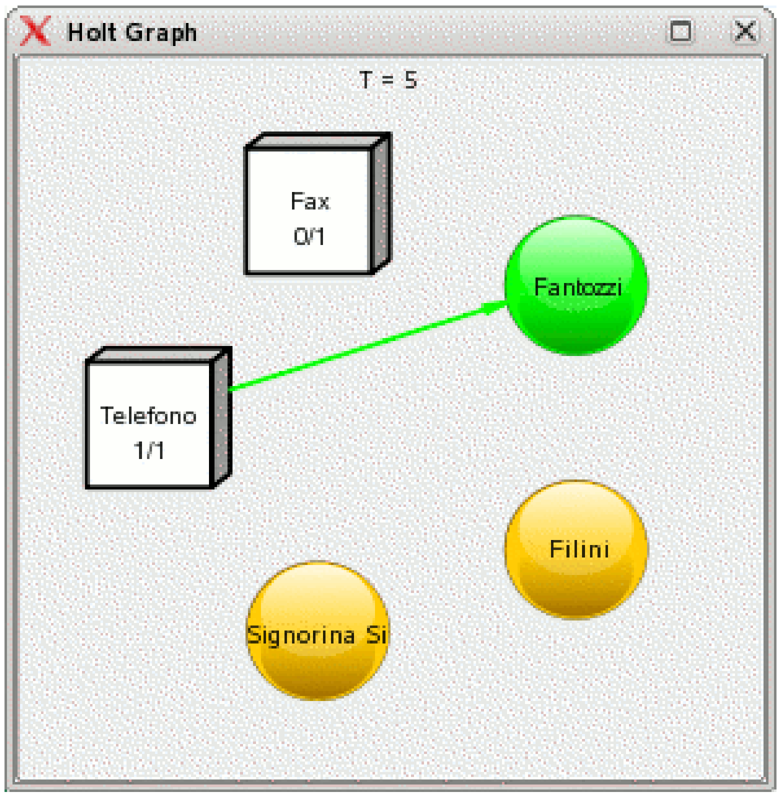
The options available are:

- **No scaling** (default mode) the graph isn't scaled at all. A white space can appear at right or bottom of the graph or even the dimension can exceed client area. With horizontal and vertical scrollbar the user can view all the graph surface.

- **Fit in window** the graph is resized to make visible every part of the graph. A white (sometimes big) space can appear at right or bottom of the graph.
- **Stretch in window**like above the graph is resized but even stretched to cover all client area.

Always one of these commands isn't available at a time; the current mode doesn't appear because there isn't any reason to choose it.

4.1.5 The Holt graph



The graph show the simulation status at *this time*.
It represent resources, processes or threads (and status), requests for resources and allocation.

If the user choose to view processes then a circle per process is displayed, if she/he choose to view threads only a circle per thread (and no process) is displayed.

Resources are drawn as squares, processes and threads are circular, requests and allocations are in form of arrows.
In center of resources are printed two lines: the name at top, the used/total places at bottom.
Into schedulables is shown their name.
An arrow from process (thread) to a resource is a request from the process to have the resource; an arrow from the resource to the process denote the allocation of the resource to the process.

The colors, as usual, are: green, yellow, red.

- **Green** is used for running processes/threads
- **Yellow** is used for ready processes/threads (waiting to run)
- **Red** is used for blocked processes/threads (waiting for a resource)

How to show processes or threads

With the menu item "Show/Hide Threads" under the "View" menu the user can switch from processes to threads visibility.

How to show or hide the Holt Window

Holt graph, for pratical reasons, is placed in a separate frame out of the main application window.
With the item "Show/Hide Holt graph" of the "View" menu is possible to show or hide this window. To close is always possible to use the standard close button or system menu command.

Changing graph disposition

The user can select the disposition of elements in the graph.
This option is available with a popup menu right clicking in client area.

The options available are:

- **Dispose vertical** items are arranged vertially in two columns, resources at left, processes (or threads) at right.
- **Dispose horizontal** items are arranged horizontally in two rows, resources at top, processes (or threads) at bottom.
- **Dispose circular** the items are disposed along a circle.
- **Auto dispose** (default mode) one of above is select in function of the aspect ratio of the window

Always one of these commands isn't available at a time; the current mode doesn't appear because there isn't any reason to choose it.

Changing size and shape

The user can change size of the Holt window.
As the window change size, his contents is scaled to fit into the client area.

If the disposition is set in "*Auto dispose*" mode then the disposition can change during the resizing operation as described following.
If the height/width ratio is >= 5/3 the items are arranged vertially in two columns, resources at left, processes (or threads) at right.
If the height/width ratio is <= 3/5 the items are arranged horizontally in two rows, resources at top, processes (or threads) at bottom.
Otherwise the items are disposed along a circle.

4.1.6 The Preferences dialog

The preferences window allow the user to set the simulation speed. The simulation speed is minimum waiting time between a step and an other; since computing the next step of the simulation may require the allocation of many resources, the specified speed may only be set as a minimum.

The preferences window also allow the user to add and remove the directories where policies and the plugins are found and loaded from.

Changes regarding policies and plugins will be applied at the next run of SGPEM.

Preferences are saved and loaded from the sgpem.cfg file located in the installation directory. Preferences are loaded when the application is started, and saved when the "Close" button of the dialogis pressed.

4.1.7 Controlling the simulation

The simulation itself is not interactive, so it may be thought as a recording.

From a mathematical point of view, every simulation has an instant, called its **end**, after wich no significant changes invove the simulated entities. Our simulator does reproduce simualations from the beginning to the end, and not further.

4.1.7.1 Simulation reproduction controls

Controls over the simulation reproduction are very similar to those of a digital audio player.

The "play" button starts the reproduction, the "pause" button pauses it, and the "stop" button stops it. After the simulation is stopped, the last reproduced information is left on the screen, as if the simulation were paused. Anyway, pressing play after having stopped the simulation will start the reproducion from the beginning of the recording.

4.1.7.2 Simulation reproduction modes

If the simulation play mode is set to **continuous**, reproduction of the simulation will continue until the end is reached. Otherwise the simulation will pause after every single advance in reproduction. The simulation mode may be selected on the "Simulation" menu.

4.1.7.3 Caching issues

The content of the simulation itself is calculated on demand, and cached, so the first reproduction will usually be slightly slower than the following ones.

When a simulation is stopped the cache is **not** erased. The cache is erased each time the user **modifies** the simulated environment, by adding, removing or editing any kind of entity, or by changing any policy or any of its parameters.

This is also the reason for simulations using the lottery policy will sometimes be reproduced identical.

4.2 From the commandline

4.2.1 SGPEM Commands

SGPEMv2 commands are case-insensitive, and use extensively numerical identifiers, which is annoying, but since there is no restriction to the name of the entities, it is the only way to be sure they're uniquely identifiable.
Use the `show` command to obtain the numerical identifiers you need. For most kind of entities, identifiers should not be influenced by additions, but they may be affected by removals. Also, policies are dynamically loaded at startup, so it is highly recommended you don't make assumptions on the relation between policies and their identifiers if the

application is run several times.

A list of the commands, with a detailed description follows:

`help <string>`
: If <string> is a valid command, it prints the usage instructions for that specific command, otherwise prints the list of supported commands

`run`
: Starts the simulation. It can be continuous or step-by-step depending on the mode configured with set continuous (default=true).

  The output of run is a snapshot of the state of the simulation at each instant. The instant 0 represents the initial state, during which no process is running. The scheduler activity begins at instant 1.

`pause`
: Pauses the simulation. The next call to run will continue it.

`stop`
: Stops the simulation. The next call to run will bring the simulation to the first instant and start it.

`configure <entity>`
: Where <entity> may be cpu-policy or resource-policy.
  This is currently the only way to control the behaviour of policies without modifying their source code.

`get <attr_name>`
: Where <attr_name> may be simulation-tick or continuous.

`set <attr_name> [=] <value>`
: Where <attr_name> may be simulation-tick, continuous, cpu-policy or resource-policy.
  simulation-tick is the time between steps in a continuous simulation, in milliseconds, **continuous** is a boolean ("true" or "false") indicating whether the simulation should advance continuosly or step-by-step. By default it's value is "true".

`show`
: Displays the name of the entities (if available) and other informations prefixed by its numeric identifier.
  Syntax depends from entities being displayed:
  - `show processes | resources | cpu-policies | resource-policies`
  - `show threads <process_id>` With <process_id> being the numeric identifier of the parent process
  - `show requests <process_id> <thread_id>` With <thread_id> being the numeric identifier of the thread child of process identified by <process_id>
  - `show subrequests <process_id> <thread_id> <request_id>` Where the numeric ids follow the same logic of the previous commands
  - `show statistics` Shows statistics for the whole simulation for the current instant

`add`
: Adds an entity by using a questionary-like approach.
  Syntax depends from entity being added:

- add process | resource
- add thread <process_id> With <process_id> being the numeric identifier of the parent process
- add request <process_id> <thread_id> With <thread_id> being the numeric identifier of the thread child of process identified by <process_id>
- add subrequest <process_id> <thread_id> <request_id> Where the numeric ids follow the same logic of the previous commands

remove
: Removes an entity.
Syntax depends from entity being removed:
  - remove process | resource <id> Where <id> is the process or resource identifier
  - remove thread <process_id> <thread_id> With <process_id> being the identifier of the parent process, and <thread_id> the id of the thread to be removed
  - remove request <process_id> <thread_id> <request_id> Where the numeric ids follow the same logic of the previous commands
  - remove subrequest <process_id> <thread_id> <request_id> <subrequest_id> Where the numeric ids follow the same logic of the previous commands

save <filename>
: Saves the simulation to file <filename>, which may be a path in a format suitable for the operating system used.

load <filename>
: Loads a simulation from file <filename>, which may be a path in a format suitable for the operating system used.

quit
: Gently closes the program. You may also use the *C-d* combination to obtain the same effect, but only from the "main" command prompt, not inside wizards for adding entities or for configuring policies.

4.2.2 SGPEM Output

The output of RUN is pretty complex.
Example:

```
>>>> 4
READY QUEUE: { Anassimandro ~ }
RESOURCES:
  0. forchetta, with 1 places
         queue: { [Anassimene] || Pitagora ~ Pitagora }

PROCESSES:                   state   arrival  requiring  elapsed  priority   res_id
 1. Pitagorici             BLOCKED         0          4        0         0
 1. Pitagora               BLOCKED         0          4        0         0
       1.1 forchetta   UNALLOCABLE         0          4        0                  0
       1.2 forchetta   UNALLOCABLE         0          4        0                  0
       2.1 forchetta        FUTURE         2          4        0                  0
 2. Scuola di Mileto    >> RUNNING <<      3          8        1         0
 1. Anassimene          >> RUNNING <<      0          6        1         0
```

```
          1.1 forchetta     ALLOCATED         0          2        1                  0
       2. Anassimandro          READY         0          2        0         0
       1.1 forchetta           FUTURE         0          2        0                  0
```

The first number (4, in this example) is the current instant of the simulation.
Just below there's the ready queue, containing the threads ready to be executed, it'll be up to the scheduling policy to decide what to do with them.

Then there are resources. The number just before their name is their numerical identifier (the one displayed also by `show`). Each resource has its subrequests queue, where the leftmost element is the first in the queue (since subrequests have no name, the name of the thread issuing it is used). Elements in the queue are normally separated by a `"~"`, while a `"||"` is used to separate allocable subrequest from unallocable ones (allocables are to the left of the separator, unallocables to the right).

Finally there are processes, threads and requests. The hieararchy is similar to the one used for the Section 4.1.2 [The Schedulables/Requests tree], page 12, except that requests are expanded, and only subrequests are shown. The number used for processes and threads is simply their numerical identifier, as it is for resources.
There are two number separated by a dot for subrequests, the first is the numerical identifier of the request, the second is the indentifier of the subrequest itself.

For this kind of entities, a tabular format is used, and fields are left blank if the information is not available for an entity. The name of the columns should be self-explaining.

5 Extending SGPEM

5.1 Writing new policies

All built-in policies are implemented in Python, but don't worry: you don't have to be a Python expert to write a new policy. We'll explain you how to write a new policy on an simple example of FCFS policy. Then a more complex example will follow: a Round Robin policy that uses pre-emption by priority.

Now let's get started, all you have to do to create your own policy is to change the few bold lines of the following example. Also remember that the name of the class have to be the same of the name of the file (minus the `.py` file extension, of course).

5.1.1 A beginner example: First Come First Served

```
01 from CPUPolicy import CPUPolicy
02 class fcfs(Policy) :
03     def __init__(self):
04         pass;

05     def configure(self):
06      print 'No options to configure for fcfs'

07     def is_preemptive(self):
08      return False

09     def get_time_slice(self):
10      return -1

11     def sort_queue(self, event, queue):
12      cmpf = lambda a, b: \
          a.get_schedulable().get_arrival_time() +     \
             a.get_process().get_arrival_time      <=  \
          b.get_schedulable().get_arrival_time() +     \
             b.get_process().get_arrival_time
13      self.sort(queue,cmpf)
```

body of `def configure(self)`: line 06
: Configure policy to initial values. This is called just before a simulation starts, and it is responsible to define the parameters the policy wants to expose to the user. For example, it may make the return value returned by `is_preemptive()` configurable, or to register an integer value for a the time slice duration.

body of `def is_preemptive(self)`: line 08
: It says whether the policy wants to be preemptive, other than by normal time slice termination (if a positive time slice has been provided).

  The possible return values are:

1. `True`: If the policy returns True, it declares that it wants the running thread to be released if a thread at higher priority is put at the beginning of the ready threads queue.

   This is achieved by putting the current running thread, if there is one, onto the ready queue. It is up to you, into the `sort_queue()` method, to manage this special case.

2. `False`: The policy always waits the end of the time slice (or a thread blocking/termination) before selecting a new running thread, even if it has greater priority than the current one.

   There will never be a running thread in the ready queue passed to `sort_queue()`.

Please note how the word "priority" here has a general meaning: it indicates every thread than can bubble up the sorted ready queue and come before another. So it's up to Policy.sort_queue() to give it a precise meaning.

body of `def get_time_slice(self):` line 10

Returns how long is a time-slice for this policy. A time sliced policy should return a positive integer value, a policy which doesn't use slices should instead return `-1`. You're encouraged to use a user-configurable parameter via `Policy.configure()` if the policy is time-sliced, to ensure greater flexibility.

body of `def sort_queue(self, event, queue):` line 12,13

Sort the queue of ready threads. This method is called by the scheduler at each step of the simulation to sort the ready threads queue. It is the core of your policy: when scheduler has to select a new thread it will always try to take the first of the queue. If it cannot run for some reason (for example, it immediately blocks), the second is selected and so on, until the end of the queue.

Remember that if `is_preemptible()` returns True, you may have a running thread in the queue. See the following example for some tips about how to manage this case.

Pay attention to the fact that we used the `<=` relation at line '`12`', and not a simple `<`. This is because `queue.sort()` uses a in-place implementation of quicksort. See [ReadyQueue.sort_queue()], page 23. If your policy behaves strangely, this may be the cause.

5.1.2 Exposed interface: what you can use

This is a list of exported interfaces that you can use from your policy script to manipulate SGPEMv2 exported objects.

If you want to see what methods a Python object exports, remember that you can also use the built-in `dir()` Python function.

5.1.2.1 Configuring parameters

TODO: list and describe all methods exposed from PolicyParameters. In the meantime, see the example below about the RR policy with priority.

5.1.2.2 Methods for manipulating the ready queue

The parameter `queue` passed to `CPUPolicy.sort_queue()` is of type `ReadyQueue`. This is a description of the available methods:

`ReadyQueue.sort_queue(queue, compare_function)`
: This is the function that actually does the sorting of the queue for you. You can of course avoid to call this method and sort the queue by hand (the "lottery" policy for example doesn't call it).

  It takes two parameters: the first is the queue, and the second is a compare function. Usually you'll want to use a simple lambda-function defined in the way you can see in the above and following examples.

  Remember that this function will internally use a in-place version of quicksort, which is a stable sorting algorithm only when employed with a less-or-equal relation("`<=`") or a greater-or-equal one ("`>=`"). Otherwise the queue would still be sorted, but two adjacent threads that have the same value for a given property would be swapped. This might be indesiderable with certain policies, and could lead to unexpected results, so be careful.

`ReadyQueue.size()`
: Returns the number of elements in the queue.

`ReadyQueue.get_item_at(position)`
: Returns the thread contained at the given position of the queue, where `0` means the front, and `queue.size() - 1` means the last element (the back) of the queue. Trying to access an element outside the range [0, queue size) will raise an exception.

`ReadyQueue.bubble_to_front(position)`
: Moves the item at the given position up in the queue until it reaches the front, preserving the order of the other threads. Trying to access an element outside the range [0, queue size) will throw an exception at you.

`ReadyQueue.swap(position_a, position_b)`
: Swaps the element in position a with the element in position b. This is used mainly by the internal quicksort implementation, but you may want to employ it directly in some cases, too. As you may have already guessed, trying to access an element outside of the queue will raise an exception.

5.1.2.3 Properties of schedulable entities

All schedulables, both threads and processes, implement the following methods:

`get_arrival_time()`
: Returns the time a schedulable arrives to the CPU. For a thread, it is relative to the time his parent process is spawned. For a process, it is the absolute time value.

  So, a thread will arrive to the CPU after `get_arrival_time() + get_process().get_arrival_time()` units.

`get_elapsed_time()`
: Returns for how many time units a schedulable has been running up until now.

`get_last_acquisition()`
: Returns the last time a schedulable has been selected for scheduling (that is, to become the running one).

`get_last_release()`
: Returns the last time a schedulable had stopped being scheduled as a running and has been preempted. Note that this also happens every time a time-slice ends.

`get_base_priority()`
: Returns the priority a schedulable has been spawned with.

`get_current_priority()`
: Returns the current priority. It is usually given by `get_base_priority() + priority_push`. See below.

`set_priority_push(new_value = 0)`
: Sets the priority push to change the base priority of a schedulable. It is the only method available that changes the state of a schedulable.

`get_total_cpu_time()`
: Returns the time a schedulable will run before terminating.

`get_state()`
: Returns a string describing the state of a schedulable. It can be:

  1. "future"
  2. "ready"
  3. "running"
  4. "blocked"
  5. "terminated"

`get_name()`
: Returns a string with the name the user gave to the schedulable.

Class `Thread` has another method, which is `get_process()`. It returns the father process. Class `Process` behaves similarly by providing a `get_threads()` method that returns a list of children threads.

5.1.3 A more complete example: Round Robin with priority

Now, let's see a more interesting (and a little more complex) example: a Round Robin by priority policy that can optionally also work with pre-emption by priority.

```
from CPUPolicy import CPUPolicy

class rr_priority(CPUPolicy) :
   """Round Robin scheduling policy that takes priority in account.

```

```
No lower priority thread can run if a higher
priority thread exists. If pre-emptive by priority, a
higher-priority thread becoming ready even in the middle
of a time slice will pre-empt the running thread. Else,
the time slice will have to end before the former can run."""

   def __init__(self):
       pass;

   def configure(self):
       param = self.get_parameters()
       param.register_int("Time slice", 1, 10000, True, 2)
       param.register_int("Is preemptive?", 0, 1, True, 1)

   def is_preemptive(self):
       value = self.get_parameters().get_int("Is preemptive?")
       if value == 0:
           return False
       else:
           return True

   def get_time_slice(self):
       return self.get_parameters().get_int("Time slice")

   def sort_queue(self, queue):
       by_ltime = lambda a, b: \
           a.get_last_acquisition() <= \
           b.get_last_acquisition()
       by_prio = lambda a, b: \
           a.get_current_priority() <= \
           b.get_current_priority()

       self.sort(queue,by_ltime)
       self.sort(queue,by_prio)

       # manage preemption: see if we've a running thread
       # in the ready queue, and if it can still run
       if self.is_preemptive() == True:
           higher_prio = queue.get_item_at(0).get_current_priority()
           i = 0
           while i < queue.size():
               sched = queue.get_item_at(i)
               priority = sched.get_current_priority()
               if(priority != higher_prio):
                   break
               if sched.get_state() == "running":
                   queue.bubble_to_front(i)
```

```
52                  i += 1
```

We've also added a description of the class immediately following the class declaration (lines '03-09'). This is what is returned as the policy description in the frontend. You may want to document your policies in the same way too.

Now, let's see the most complex parts together:

`configure()`
: There are three types of parameters you can register in the value returned by `self.get_parameters()`, and they are integer parameters, float parameters and strings. Usually boolean values can be simulated by registering a integer parameter limited in the interval [0, 1]. See [Configuring parameters], page 22, for the exposed interface.

`is_preemptive()`
: TODO: write me

`sort_queue()`
: Here there are quite a lot of things going on, so let's tackle them one by one.

  At line '30' we create a lambda-function that says to sort the queue by last aquisition time, so that threads that have been aquired recently end up at the back of the queue (which is exactly what a Round Robin policy should do).

  Then, at line '33', we create another lambda-function, this time because we want to sort the queue by priority, too.

  Done this, we let quicksort do the hard job at lines '37-38'.

  Since we may have pre-emption enabled, we may have a running thread on the ready queue (if one exists at the current instant). But what happens if the running thread was put in the queue, and we just sorted it?

  Unfortunately, having the greatest last aquisition time, the running thread would end at the back of the queue, thus never being selected to run for more than a single time unit if the queue is non-empty and there are other threads with the same priority!

  The solution is to check if there is a thread with state "running" at the beginning of the queue, between those that have the same priority. If there's one, we make it bubble to the top of the queue.

  This is the explanation for lines '42-52'.

5.2 Writing plugins

Writing plugins for SGPEMv2 goes outside the scope of this manual. For some informations on how to extend it with a plugin of yours, See section "Writing your own plugins" in *SGPEMv2 Developer Manual*.

Appendix A License

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

   The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

   Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.1 *Addendum*: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index