General-Purpose Autonomic Computing Framework

User Guide

GPAC 0.13

The GPAC framework is released under the GNU Affero General Public Licence (AGPL) version 3 or later. The main terms of the GNU AGPL licence are reproduced below for convenience.

GPAC is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GPAC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with GPAC. If not, see <http://www.gnu.org/licenses/>.

Copyright © 2009 Radu Calinescu <Radu.Calinescu@comlab.ox.ac.uk> (University of Oxford)

Contents

1. Introduction 4
2. Installation 5
   2.1 Policy engine installation and configuration 5
   2.2 Admin tool installation and configuration 10
3. Policy engine parameters 12
4. Application development with the GPAC framework 12
   4.1 Generation 13
   4.2 Deployment 20
   4.3 Exploitation 21
5. Audit trail 25
   5.1 Complete audit trail 25
   5.2 Full policy details 25
   5.3 Policy evaluation steps and policy actions 26
6. Types of autonomic computing policies 28
   6.1. Action policies in GPAC 28
   6.2. Utility-function policies in GPAC 28
      6.2.1 Operational models 31
      6.2.2. Supplying an operational model to the autonomic manager 32
   6.3 Utility policies using PRISM quantitative analysis 37
References 39
Appendix A. Installing IIS and .NET on a server 40
Appendix B. GPAC autonomic computing policies 41
   B.1 Overview 41
   B.2 Policy syntax 42
   B.3 Policy semantics 45
      B.3.1 Policy scope 45
      B.3.2 Policy condition 45
      B.3.3 Policy action 46
Appendix C. Setting up PRISM within GPAC 47
Appendix D. Known limitations 48

1. Introduction

The General-Purpose Autonomic Computing (GPAC) framework is intended to reduce the time and expertise required for the development of autonomic computing applications. GPAC achieves this goal by reusing software components across application domains, and by assisting the developer in building those components of an autonomic computing solution that are application specific.

The generic architecture of a GPAC autonomic computing application is detailed in Fig. 1:

1. The *system resources* in this architecture represent the components of the system to which the application adds self-management capabilities.
2. The *reconfigurable policy engine* represents the component that monitors the state of the system resources and adjusts their configurable parameters in line with the high-level objectives of the overall systems. These objectives are specified in the form of *autonomic computing policies* by the system administrator/user.
3. The *manageability adaptors* are thin interfaces that enable the policy engine to access the state and configuration parameters of the system resources.
4. The *policy engine admin tool* provides a GUI interface that the system administrator can use to configure the policy engine, e.g., to supply the autonomic computing policies for the system.

The GPAC framework provides a fully operational policy engine (implemented as a .NET web service) that can be used across applications; and a GUI policy engine admin tool (implemented as an ASP.NET application). The system resources from Fig. 1 represent pre-existing, legacy components of the application. Therefore, the only application components that need to be developed are the manageability adaptors ─ one adaptor is required for each *type* of resource in the system, and the GPAC framework simplifies this task by automating many of steps associated with the implementation of these manageability adaptors as .NET web services.

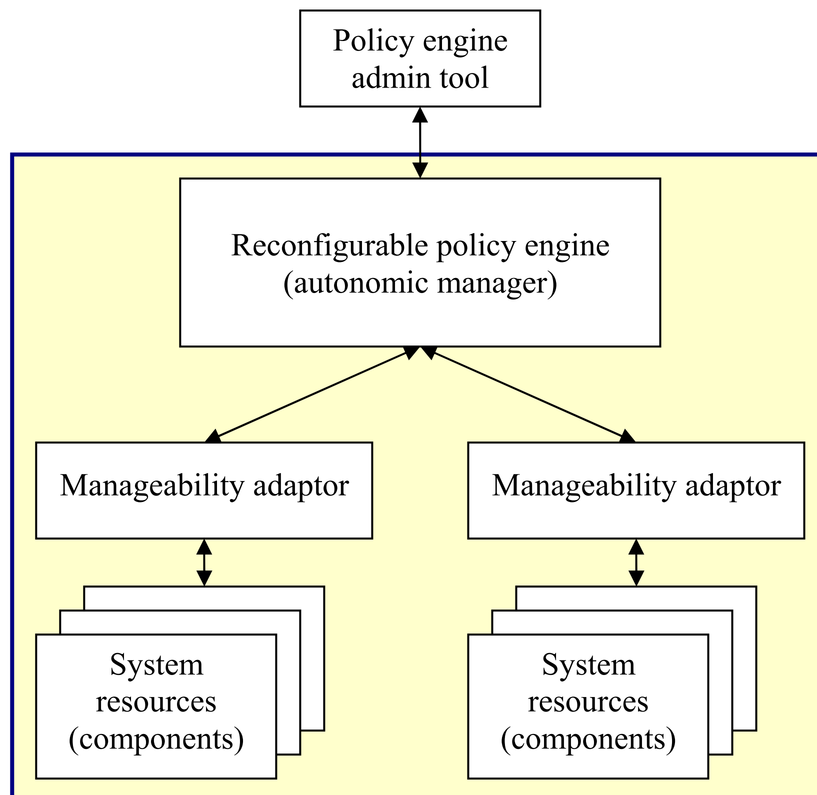


Fig. 1 Architecture of a GPAC autonomic computing application

2. Installation

The GPAC framework comprises the components is depicted in Fig. 2:

- `adaptor/` contains the base, abstract class specialised by the manageability adaptors.
- `client/` contains the ASP.NET policy engine admin tool.
- `doc/` contains this user guide.
- `engine/` contains the reconfigurable policy engine.
- `examples/` contains sample applications.
- `metamodel/` contains the meta-model for the system XML models used to configure the policy engine, as explained later in the user guide.
- `thirdParty/` contains the licences of third-party code used by and supplied with GPAC.
- `tools/` contains tools used to ease the development of autonomic computing applications.
- `COPYING.txt` represents the GNU Affero General Public Licence (AGPL) version 3 under which GPAC is released.
- `README.txt` contains a brief description of GPAC.

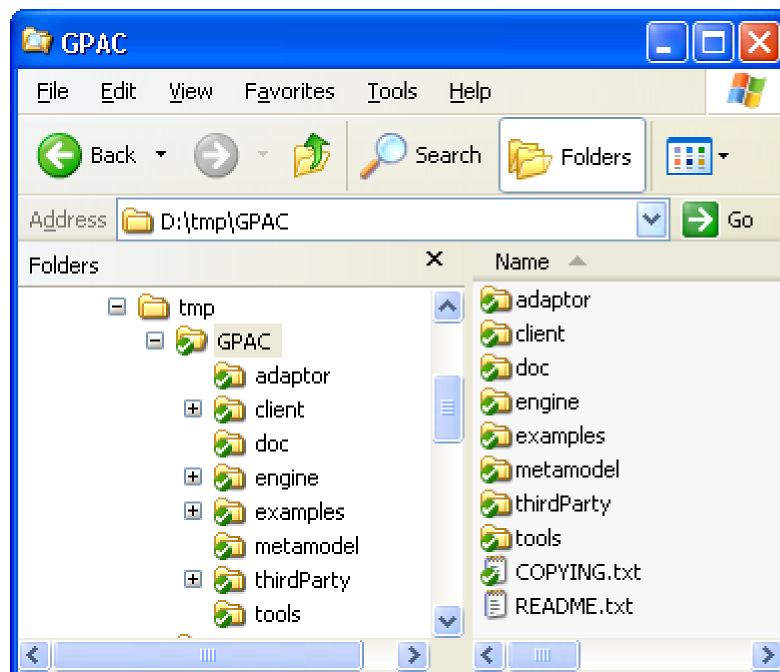


Fig. 2 The GPAC framework

Two components of GPAC must be installed and configured for any application, namely the policy engine and the policy engine admin tool. Each of the two components must be installed on a Windows machine running IIS and .NET framework version 2, from an account with administrative rights on the machine. Note that it is possible to install the policy engine and the admin tool on the same Windows server, or on different servers that can communicate with each other using the IIS ports on those servers.

2.1 Policy engine installation and configuration

1. First, make sure that IIS and .NET framework 2 are installed on the server on which you intend to deploy the policy engine. Appendix A gives details about how to do this and how to install IIS and .NET framework 2 if needed.

2. Start by copying the policy engine folder `engine/` to the IIS "Default Web Site" location on the install server (typically `C:\Inetpub\wwwroot\`, but this may be different on your server) — Fig. 3.

Note: You may want to change the name of the copied folder (i.e., `engine/`) to something suitable for your application, as this name will appear in the policy engine URL.

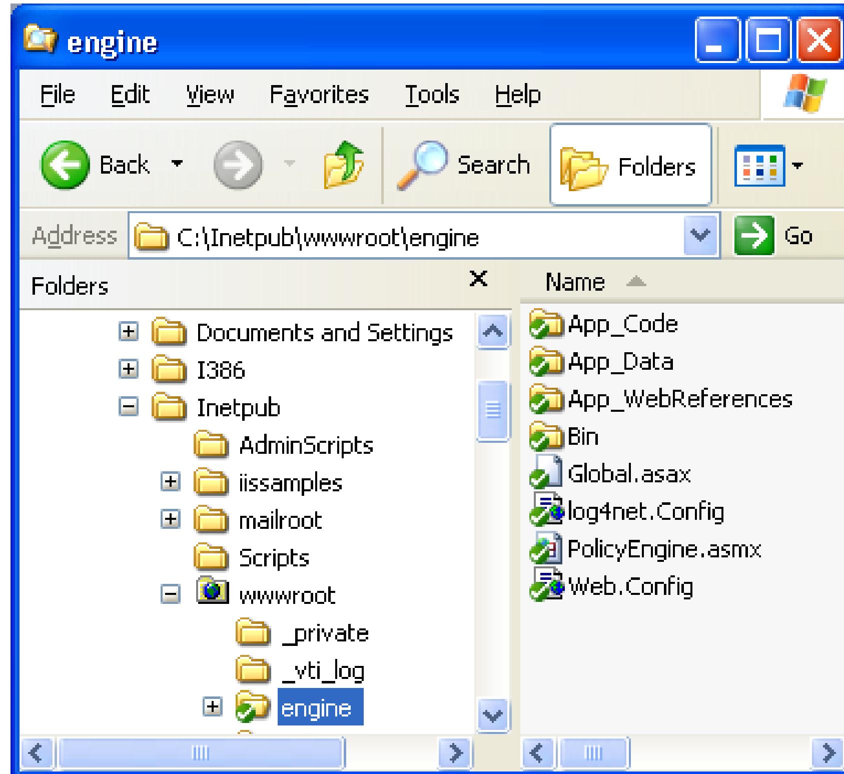


Fig. 3 Policy engine location on the IIS "Default Web Site"

3. In order to handle system resources whose types are unknown until runtime, the policy engine web service generates code (and data) dynamically and places these in its App_Code, App_Data and App_WebReferences directories. To enable this, the ASPNET account under which IIS is running web services must be given permission to write to these directories.

- Open Windows Explorer and, for each of the three directories mentioned above, right-click on 'Properties' to open their property dialog — Fig. 4.
- In the 'Security' tab, press 'Add' , specify the ASPNET account on the local server, and click OK — Fig. 7.
- Allow the ASPNET account not only to Read, List and Read&Execute the contents of the folder (i.e., the default permissions), but also to Modify and to Write it — Fig. 8.
- Ensure that the permission have been set in this way for each of the App_Code, App_Data and App_WebReferences directories.

4. In an XML or text editor, open the policy engine configuration file `engine\Web.Config`, and make sure that value attribute in the line

```
<add key="servicePath" value="c:\inetpub\wwwroot\engine"/>
```

specifies the actual path for the policy engine installation on your server. This allows the policy engine to establish where to store the code it generates dynamically at runtime.

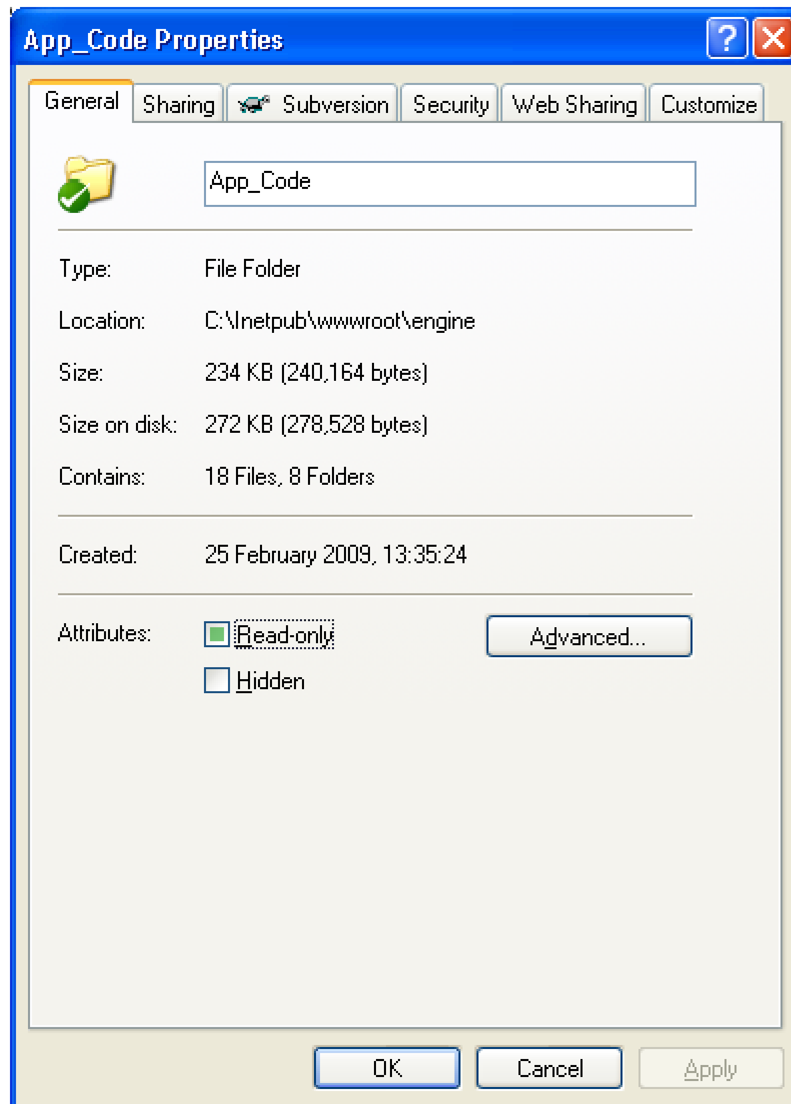


Fig. 4 'App_Code Properties' dialog

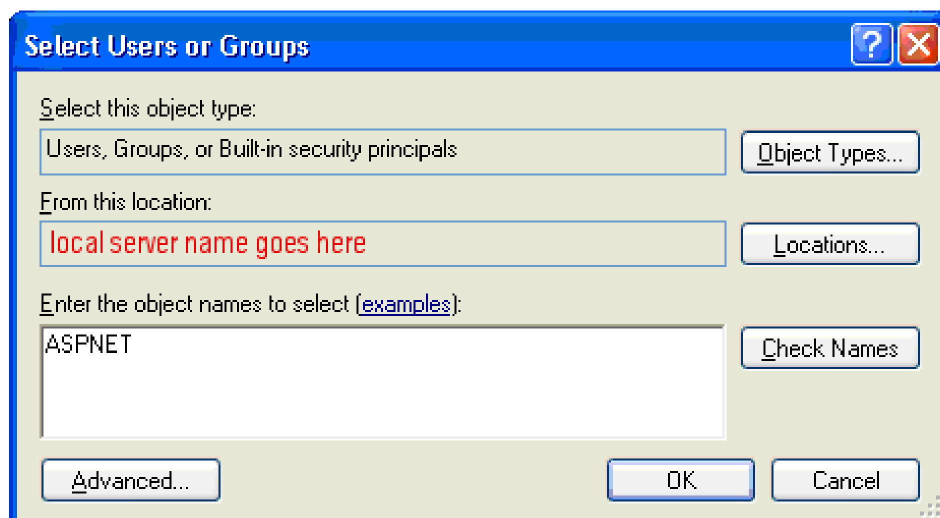


Fig. 5 Selecting the ASPNET account on the local server

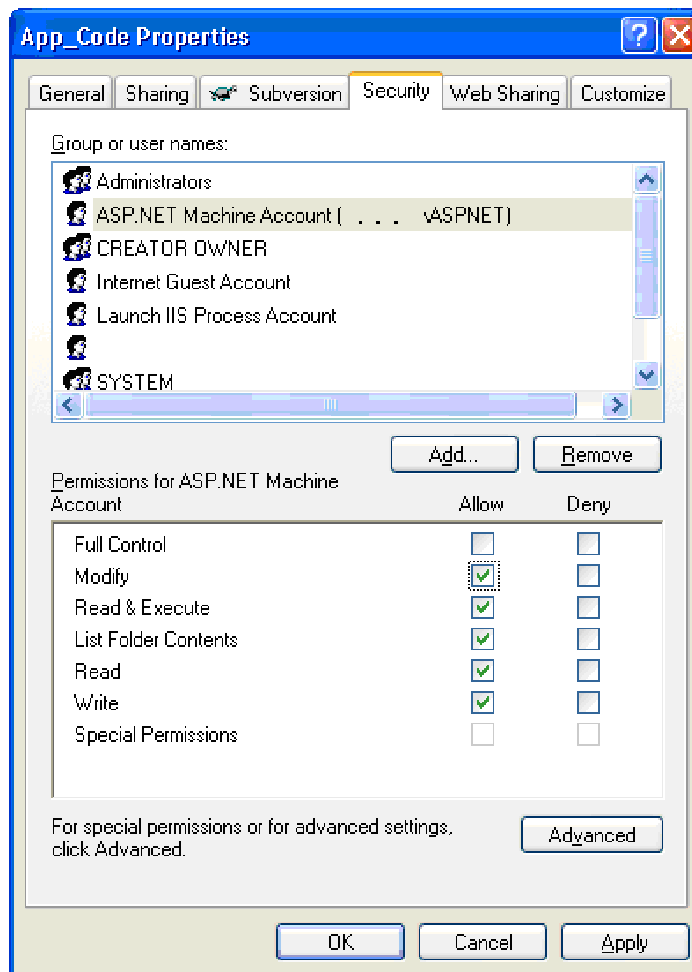


Fig. 6 The ASPNET account needs Modify and Write permissions

5. In an XML or text editor, open the asp4net logger configuration file `engine\log4net.Config` and make sure that the line

```
<file value="C:/Inetpub/wwwroot/engine/App_Data/GPAC.log" />
```

specifies the full path for a log file that the policy engine can generate. Optionally, other parameters in this configuration file can be modified as described in the log4net documentation that can be obtained from http://logging.apache.org/log4net.

Note: The ASPNET account must have permission to write and modify this file, so unless the file is in a directory for which these permissions were already set up (e.g., engine\App_Data) these permissions must be set up separately.

6. Next, open the IIS Control panel
- Start → Control Panel → Administrative Tools → Internet Information Services

open "Default Web Site" and navigate to "engine" (or the name that you gave to the copied policy engine folder) — Fig. 7.

7. Set the policy engine up as an IIS application
- Open the 'engine' properties (right-click on 'engine').
- In the 'Directory' tab, under Application Settings, press 'Create' (application).
- Press 'OK'.

The policy engine icon in the IIS control panel should change to that of an IIS application ─ Fig. 8.

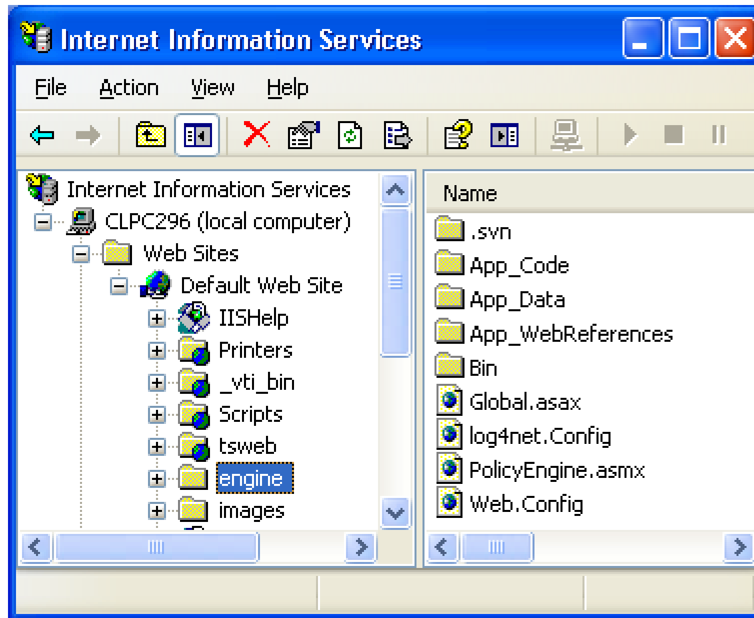


Fig. 7 Policy engine web service prior to IIS configuration

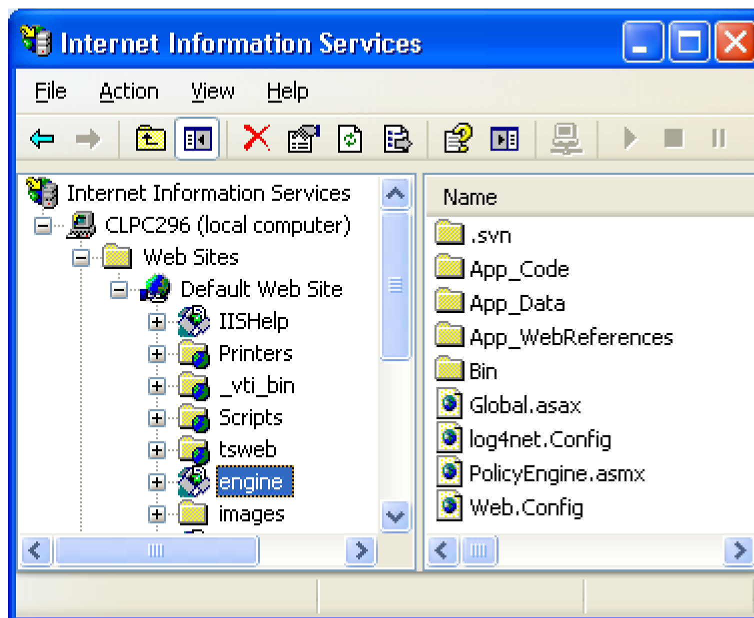


Fig. 8 Policy engine set up as an IIS application

8. Close the IIS Control panel.

You should now be able to point a web browser to the newly installed policy engine and see the summary in Fig. 9. Make sure you specify the correct port for your IIS server, as the port in Fig. 9 (i.e., 8080) may be different from yours). As additional checks, click on the 'SupportedResource' link and then on the 'Invoke' button that allows one to test this web method ─ Fig. 10 (N.B.: only works from a web browser running on the same server as the policy engine).

Depending on how the firewall settings on the server, you may or may not be able to access the policy engine web service (but not the testing functionality) from other machines.

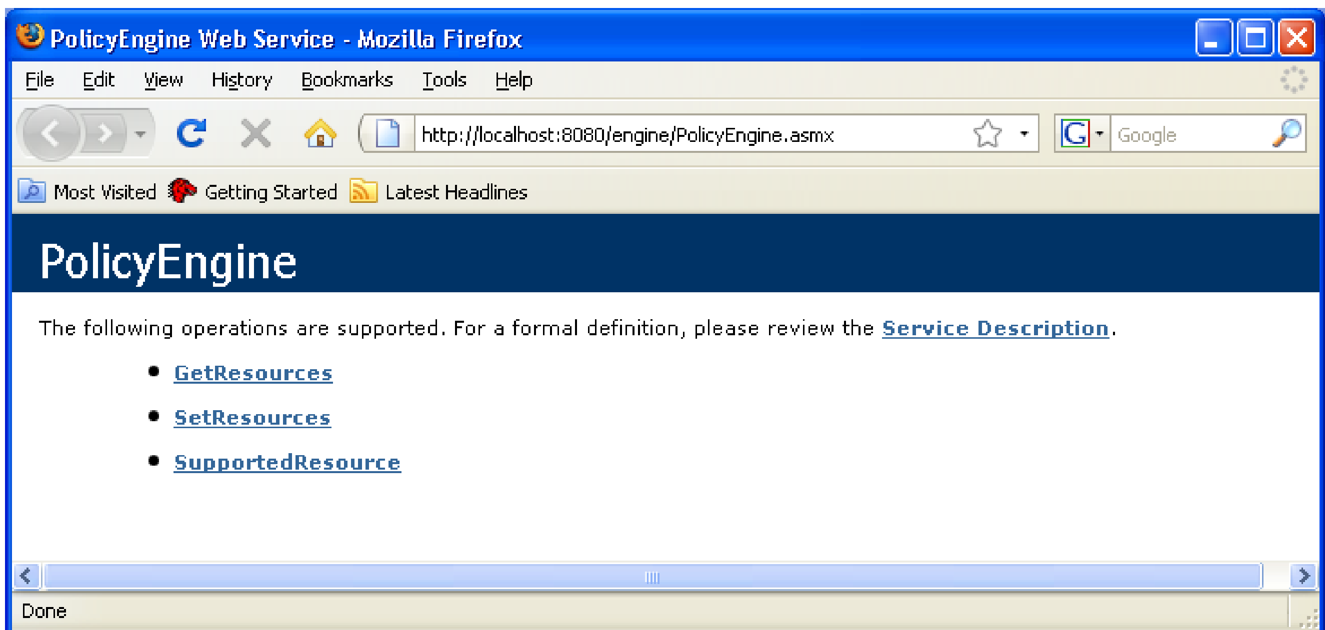


Fig. 9 Policy engine web service

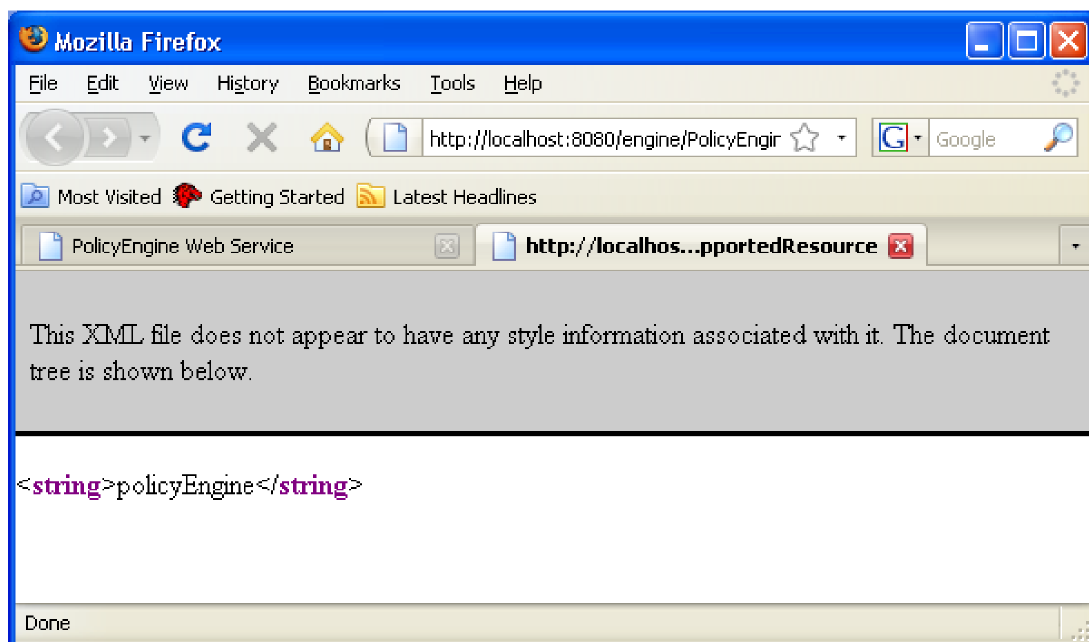


Fig. 10 Testing the `SupportedResource` web method

2.2 Admin tool installation and configuration

The install procedure for the admin tool from the `client/` directory in the GPAC distribution is similar to that for the policy engine, described in the previous section.

1. Start by copying the admin tool in `client/` to the IIS "Default Web Site" location on the install server (typically `C:\Inetpub\wwwroot\`, but this may be different on your server).

Note: You may want to change the name of the copied folder (i.e., `engine/`) to something suitable for your application, as this name will appear in the policy engine URL.

2. Open the `client\Web.Config` configuration file for the copied admin tool in an XML or text editor, and make sure the line

```
<add key="PolicyEngine.PolicyEngine"
     value="http://localhost:8080/engine/PolicyEngine.asmx"/>
```

specifies the actual URL of your policy engine. Note that if the admin tool and the policy engine are installed on different servers, the two servers must be able to communicate with each other, and the firewall configuration on the two servers must allow communication over the IIS port.

To complete the installation, set up the admin tool as an IIS application as described in steps 6 to 8 from **Section 2.1**, "*Policy engine installation and configuration*".

To test the installation, open the admin tool in a web browser — you should see the web page in Fig. 11.

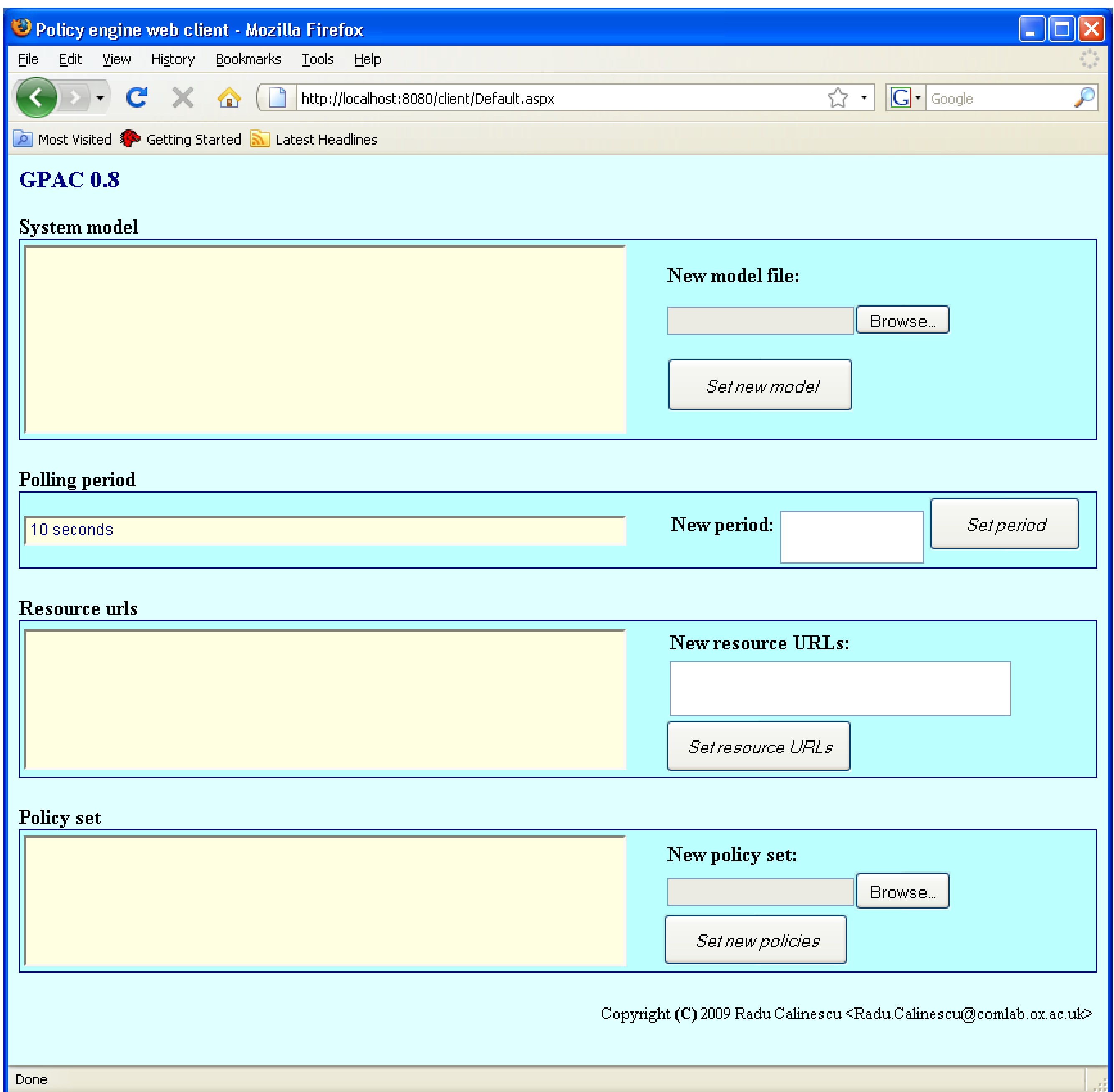


Fig. 11 Admin tool displaying the state of a newly installed policy engine

3. Policy engine parameters

As shown in Fig. 11, the policy engine comprises four configuration parameters. These parameters specify **what**, **when**, **where** and **how** the policy engine should do:

1. The *system model* parameter describes **what** the system resources within the autonomic computing application "look like". This parameter defines formally all types of resources in the system, with their *state* (i.e., read-only) and *configuration* (i.e., read-write) parameters ─ the resource parameters are also termed *properties*. System models are XML documents that are instances of the `managedSystem.xsd` XML schema from the `metamodel\` directory in the GPAC distribution.

2. The *polling period* (or simply the *period*) parameters tells the policy engine **when** to sample the state of the system and to evaluate the user-specified autonomic computing frequency. The parameter specifies the length of the time interval between two successive policy evaluation steps, in seconds. Note that policy engine is intended to also evaluate policies when *notified* that the state of the system changed. However, this functionality is not included in the policy engine described by this user guide.

3. The *resource URLs* specify **where** the manageability adaptors from the architecture in Fig. 1 are located.

4. The *policy set* parameter represents the set of autonomic computing policies that specify **how** the policy engine should manage the system resources. The types of policies supported by the current version of the engine are described later in the guide, but it is worth noting that each policy comprises the following elements:

- A *scope* that specifies which of the system resources the policy refers to.
- A *trigger* (also termed a *condition*) that specifies when the policy should be enforced.
- An *action* that describes what the policy engine should do about the resources within the *scope* of the policy when the policy *condition* is `true`. Note that the *action* element of a GPAC policy should not be confused with an "action autonomic computing policy", which is a special type of policy. GPAC policy *actions* can also be used to specify the "goal" and "utility-function" autonomic computing policies described for instance in [1].

4. Application development with the GPAC framework

The development of an autonomic computing application with GPAC (Fig. 11) comprises three stages that correspond to the sets of steps to be performed by three different user roles:[1]

1) The manageability adaptor required to organise an existing IT system into an autonomic system is devised by the *system developer* during the **generation stage**;

2) In the **deployment stage**, this adaptor is deployed, and the policy engine is configured by the *system administrator*;

3) Policies expressing the high-level system objectives are specified by the *end user* in the **exploitation stage**.

The three stages are described below and illustrated through referring to the sample application from the `examples\simpleCluster\` directory in the GPAC distribution (Fig. 13). In this sample

[1] Note that the same person may be responsible for two or all of these roles.

application, the policy engine is configured to monitor the OS processes running on a "cluster" of servers, and to stop processes as required by user-specified policies.

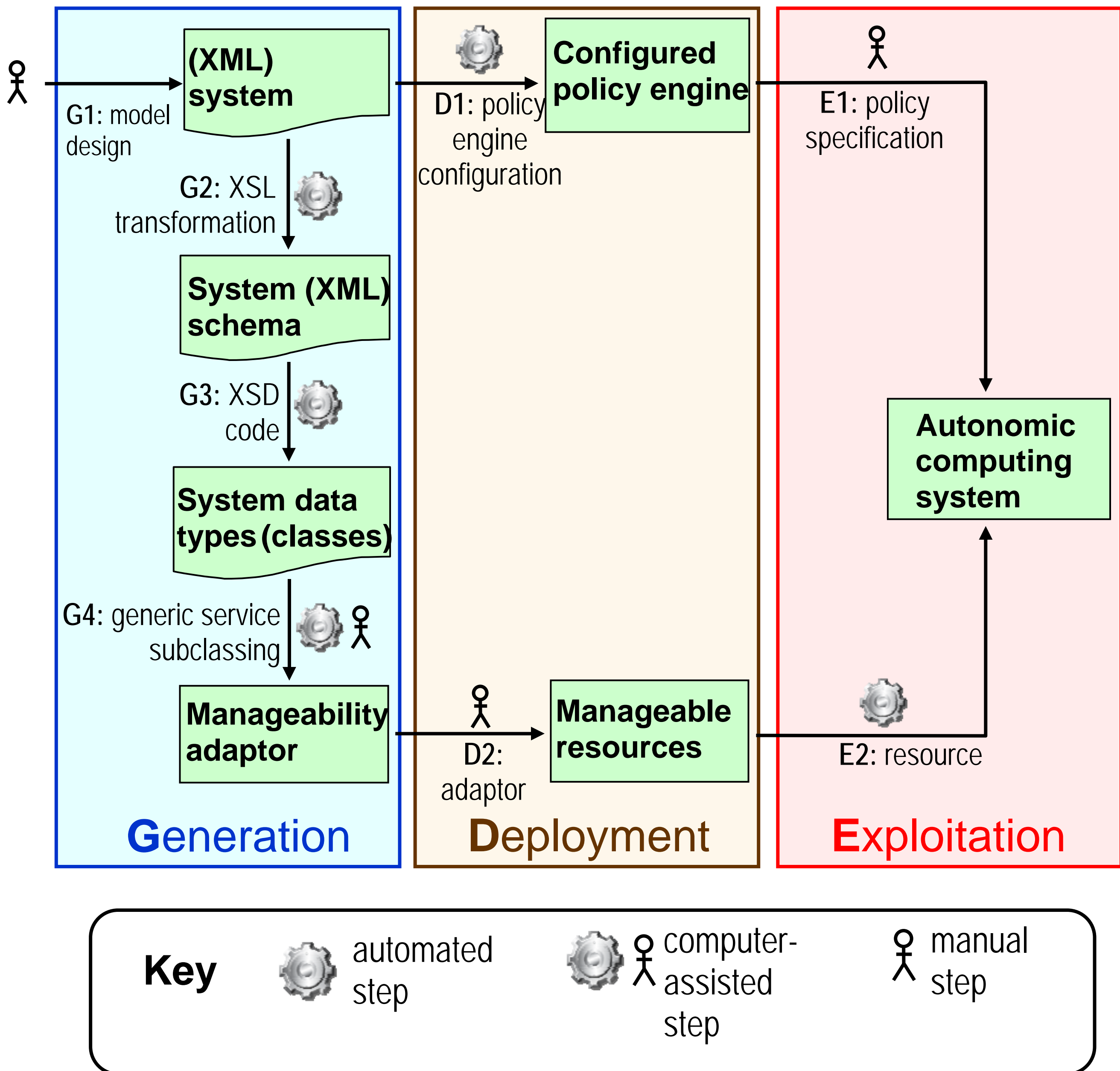


Fig. 12 Application development with GPAC

4.1 Generation

This stage comprises four steps, **G1** to **G4**. As indicated in Fig. 12, **Steps G1** to **G3** are fully automated, and **Step G4** is computer-assisted. To take advantage of this automation, use the Manageability Adaptor Generator tool included in the GPAC distribution under `tools/AdaptorGenerator/` and described at the end of this section.

In **Step G1**, an XML model of the system resources is designed by the system developer. The use of an existing, off-the-shelf XML editor such as the Oxygen XML editor (http://www.oxygenxml.com/) for this purpose is highly recommended (though not mandatory) for this. Make sure that when you create a

new XML document for the system model you specify that the new document is an instance of the XML schema `metamodel\managedSystem.xsd` from the GPAC distribution, and the editor will help with the generation of a valid instance of the GPAC meta-model. Please consult the documentation for the XML editor you use to find out how to do this.

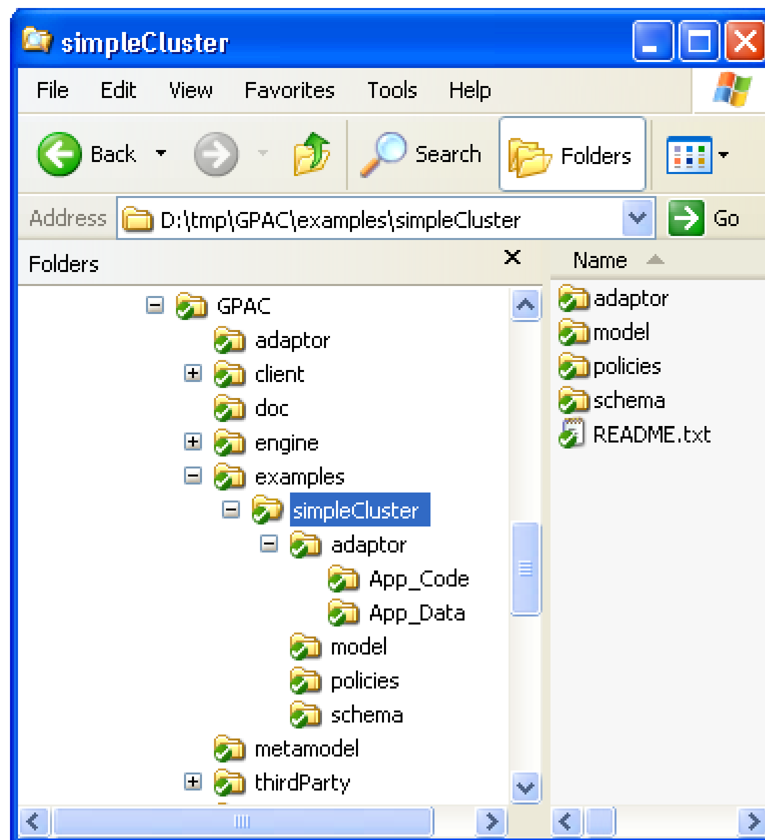


Fig. 13 Sample application used to illustrate the GPAC development process

The model for the sample application is available as `simpleCluster\model\cluster.xml` in the GPAC distribution, and partly shown in Fig. 14. The only resources of interest for the sample application are OS processes, and the model defines five properties of a process:

- *serverName*, the name of the server on which the process runs;
- *pid*, the OS process ID;
- *name*, the name of the process;
- *cpuTime*, the amount of CPU time consumed by the process;
- *stop*, a `boolean`-valued parameter that can be set to `true` to kill the process.

The first four properties of a process are read-only, i.e., *state properties*, while the *stop* property represents the only *configuration property* of a process resource.

The *serverName* and *pid* properties of a process form the *primary key* of a process resource, i.e., they uniquely identify a process within the 'cluster' system.

For a comprehensive description of all the elements of a system model, please see [2]. However, note that the mutability and subscribe-ability characteristics of a resource property are not supported by the GPAC framework version described in this guide.

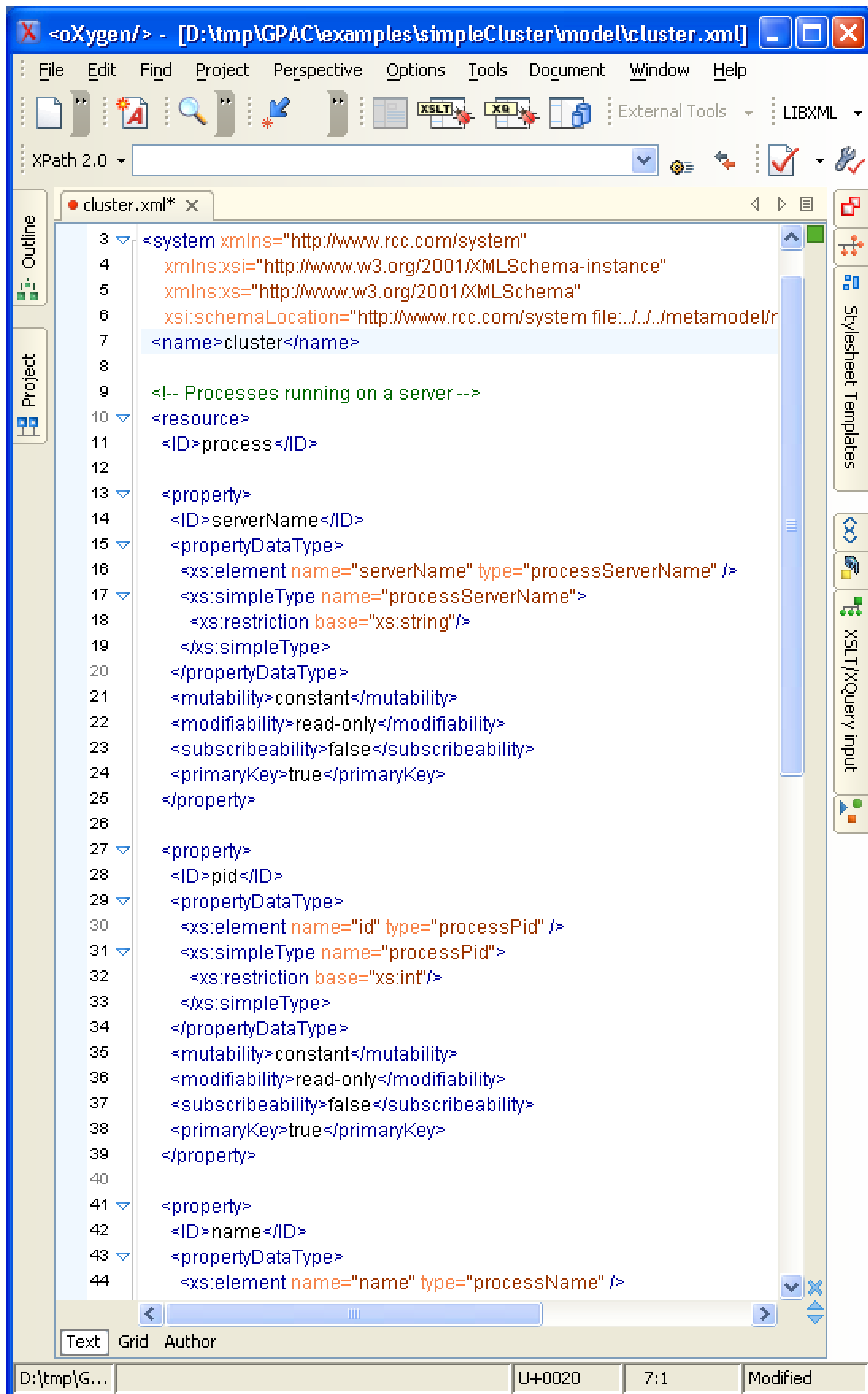


Fig. 14 XML model of a cluster for the sample application

In **Step G2** of the generation phase, the XML model derived in **Step G1** is used to generate an XML schema describing the data types associated with the system resources.

The XML schema `simpleCluster\schema\clusterTypes.xsd` is the XML schema for the sample application – Fig. 15.

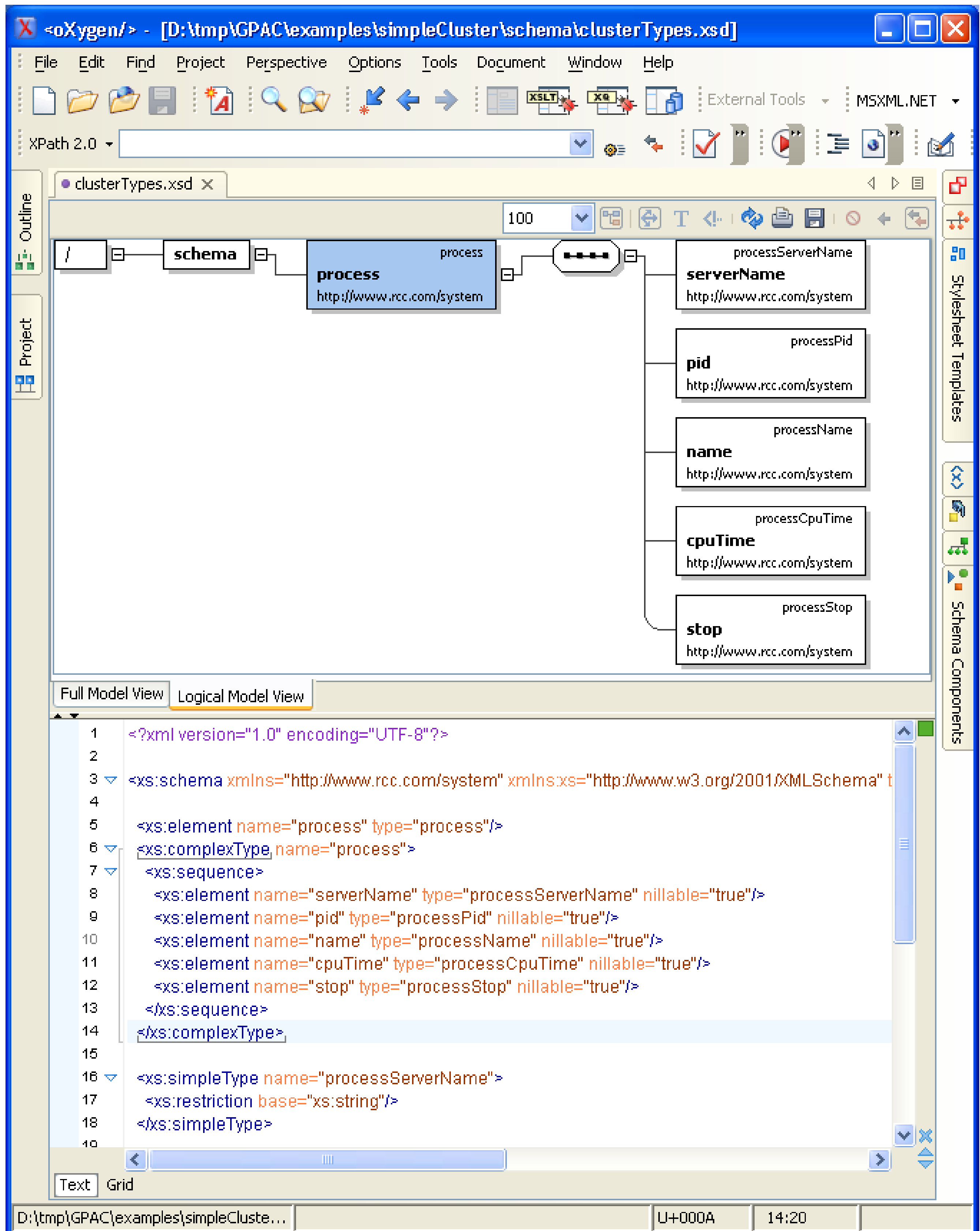


Fig. 15 XML schema for the sample application

In **Step G3** of the generation phase, the classes corresponding to the data types from the XML schema are generated automatically. The classes for the sample application were generated using the xsd.exe tool from Microsoft Visual Studio 2005, and are available in the GPAC distribution as

```
examples\simpleCluster\adaptor\App_Code\clusterTypes.cs
```

In **Step G4**, a manageability adaptor stubs is generated for each type of resource in the system, and the system developer is required to augment these stubs with code in which the actual resource APIs are used to access the state and configuration resource properties described in the system model from **Step G1**.

The elements of the "process" manageability adaptor stub generated for the sample application is shown in Fig. 16, and the file to which the developer needs to add the code mentioned above is shown in Fig. 17 – note that the name of the file in this figure should read

App_Code/ProcessManageabilityAdaptor.cs

instead of

App_Code/ManagedProcess.cs

(This inconsistency is due to file renaming.)

A completed version of the manageability adaptor stub for the application is available from the GPAC distribution – please see `examples\simpleCluster\adaptor\`.

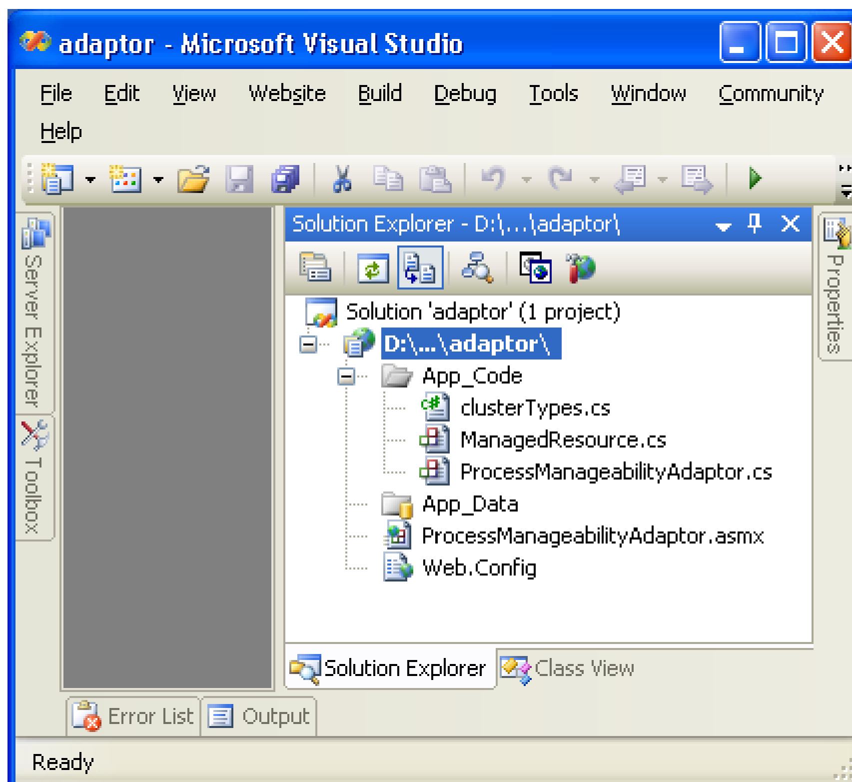


Fig. 16 The process manageability adaptor for the sample application

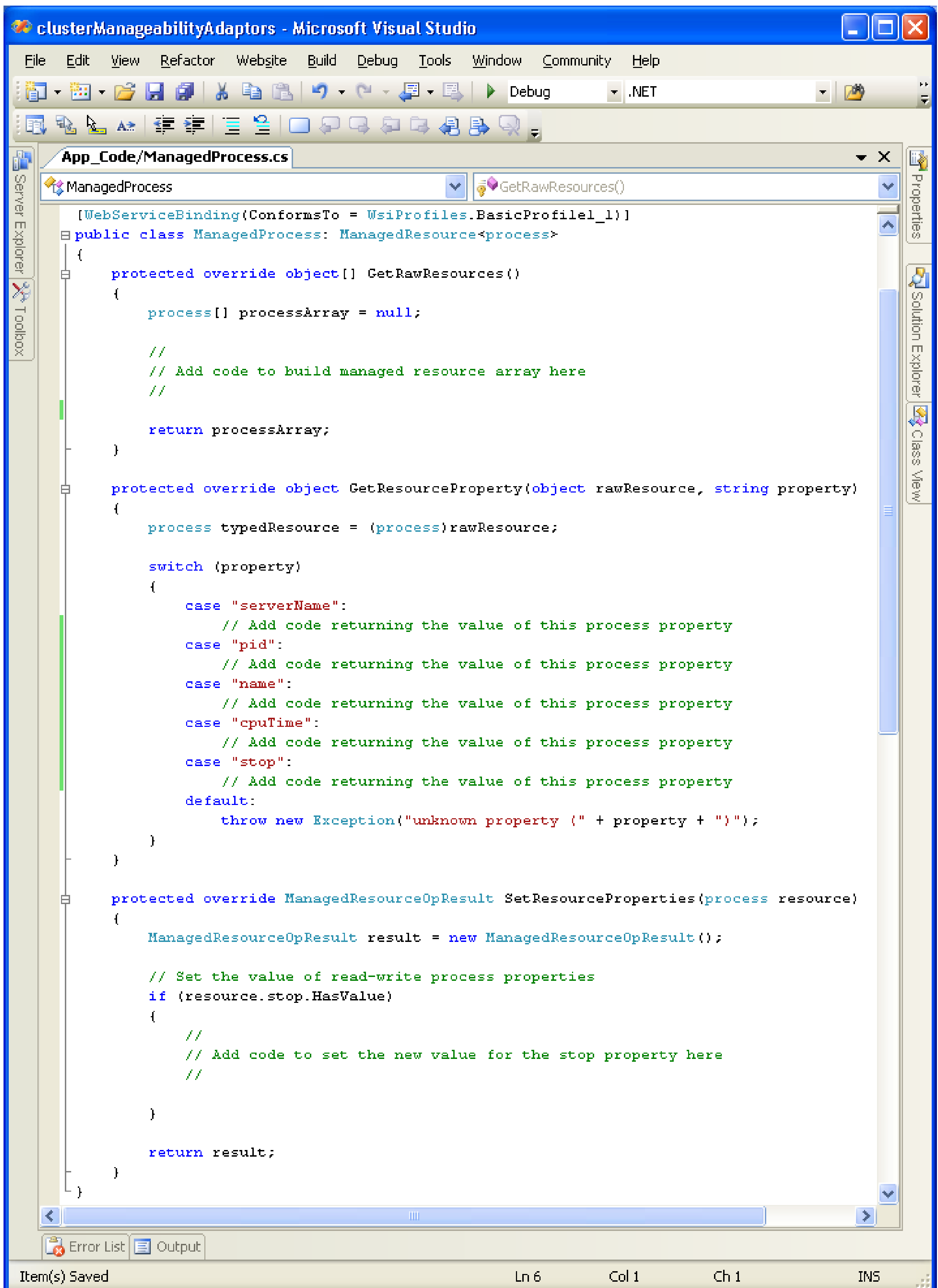


Fig. 17 The developer needs to complete the implementation of the ManagedProcess.cs stub

The Manageability Adaptor Generator tool included in the GPAC distribution should be used to take advantage of the automated generation of manageability adaptor stubs supported by the framework.

To use this tool, run the Windows application

```
tools\AdaptorGenerator\AdaptorGenerator\bin\Release\AdaptorGenerator.exe
```

from the GPAC distribution, specify the system model you want to use and an output directory, then press the 'Generate' button – Fig. 18.

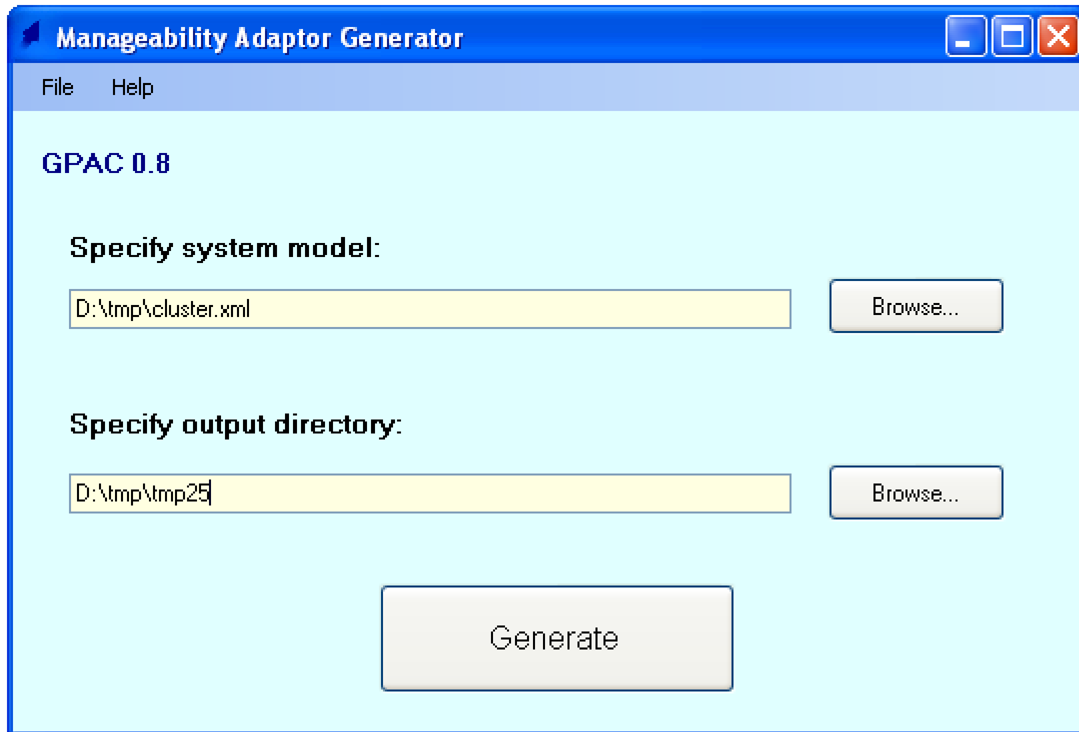


Fig. 18 The GPAC Manageability Adaptor Generator tool

The manageability adaptor generator stub is generated in the specified output directory, after which the tool could be used to generate more adaptors if required (Fig. 19).

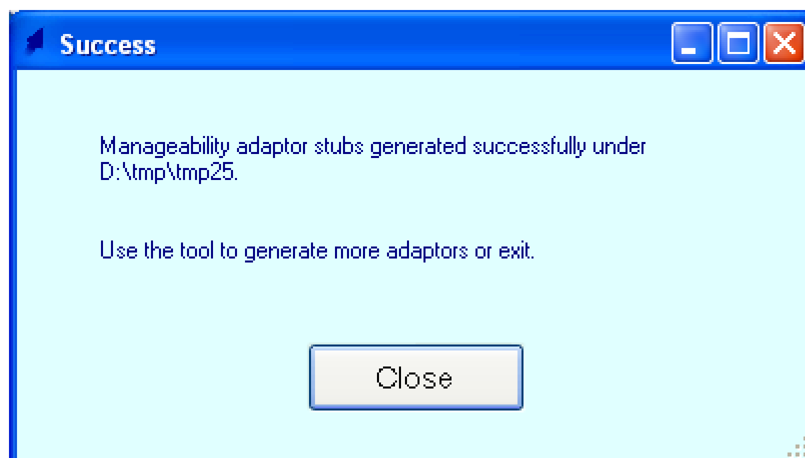


Fig. 19 Confirmation of successful adaptor stub generation

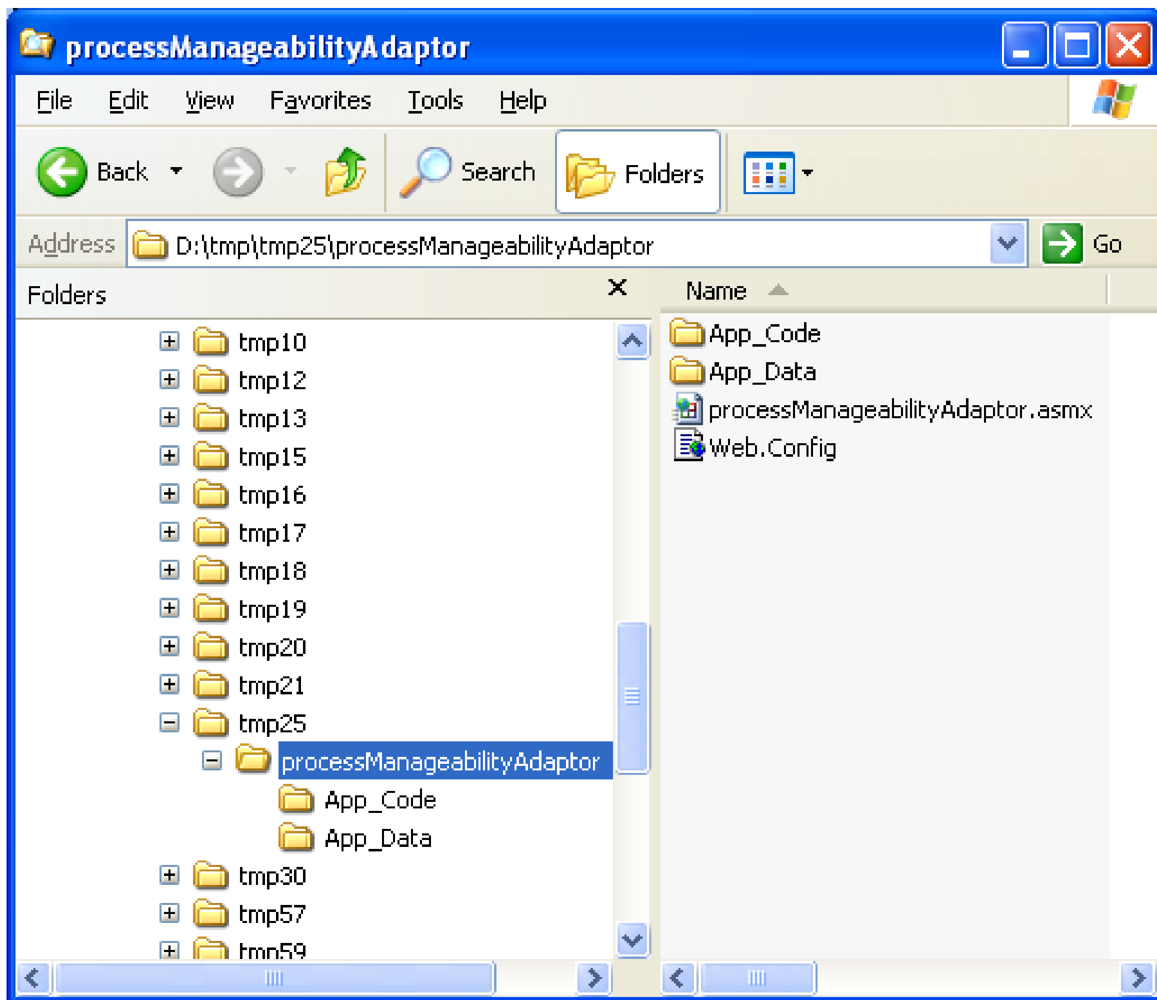


Fig. 20 Generated manageability adaptor stub for the sample application

4.2 Deployment

In **Step D1** of the deployment phase, the admin tool from Fig. 11 is used to configure the policy engine, and thus to set it up for integration in the planned autonomic computing application. To do this, select a new model file in the "System model" section of the admin tool, and press the 'Set new model' button. The result after setting the XML system model for the sample application is shown in Fig. 21.

In **Step D2** of the deployment phase, the manageability adaptors whose generation is described in the previous section are installed as IIS web services on servers from which they can access the APIs of the managed system resources. The process is similar to the one described for the policy engine in an earlier section of this guide, except that no changes to the Web.Config manageability adaptor configuration files and no granting of directory access permissions are required.

It is a good idea to use a web browser to test that the deployed manageability adaptors are operational before moving on to the next development step – Fig. 22.

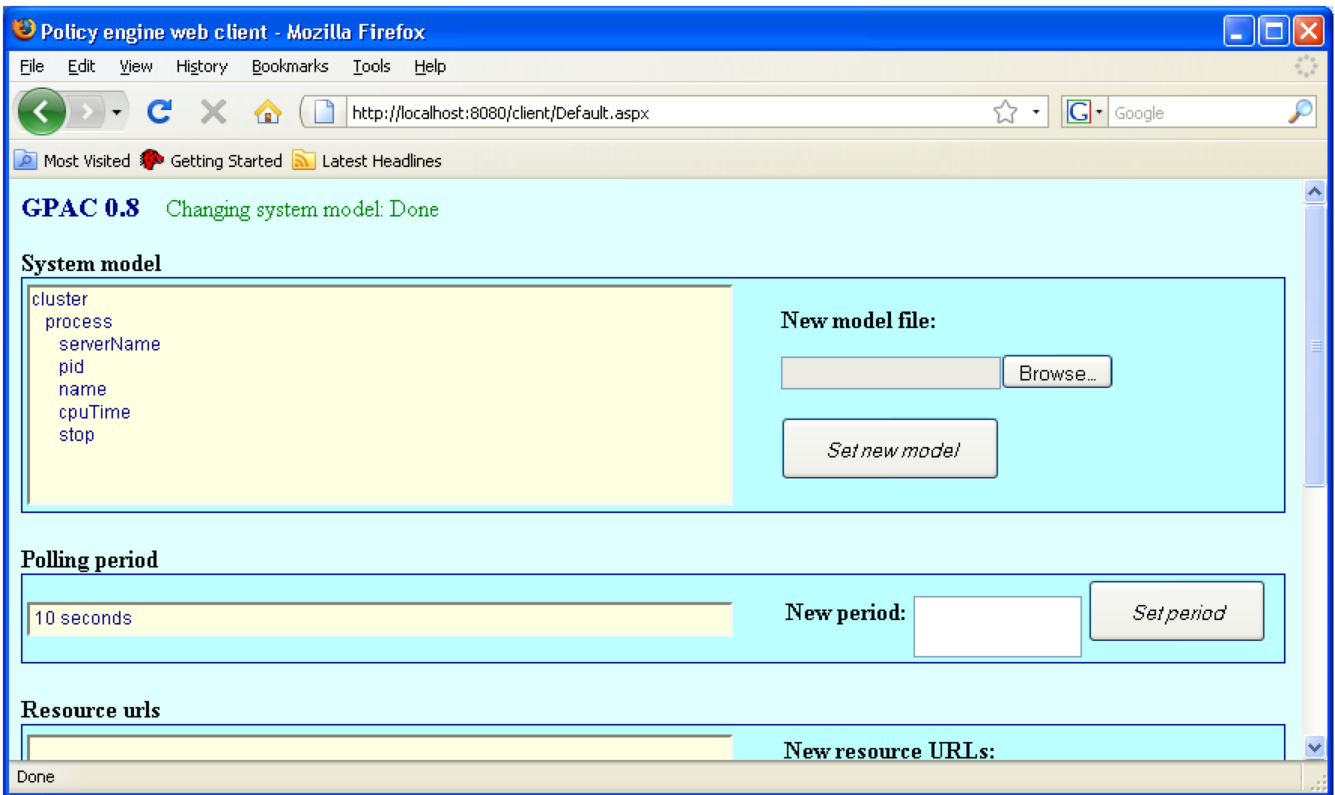


Fig. 21 Configured policy engine for the sample application

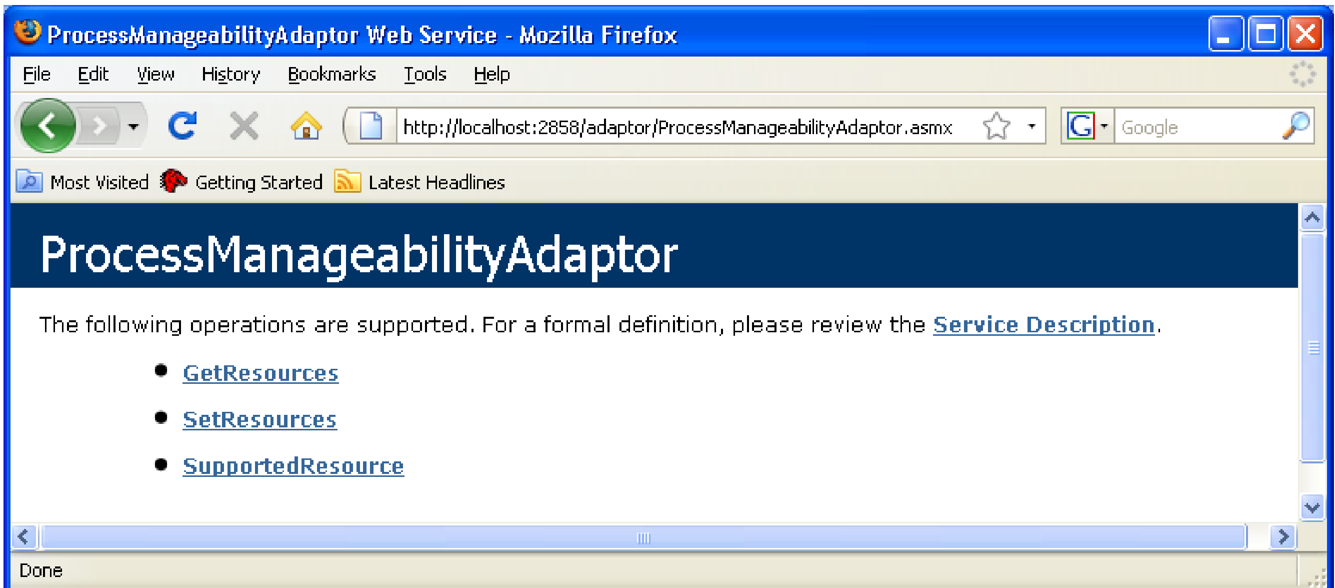


Fig. 22 Deployed manageability adaptor for the sample application

4.3 Exploitation

In **Step E1** of this development stage, autonomic computing policies are specified that encode the high-level objectives of the system. The GPAC distribution provides two simple policy sets for the sample application under `examples/simpleCluster/policies/` and one of these is shown in Fig. 23.

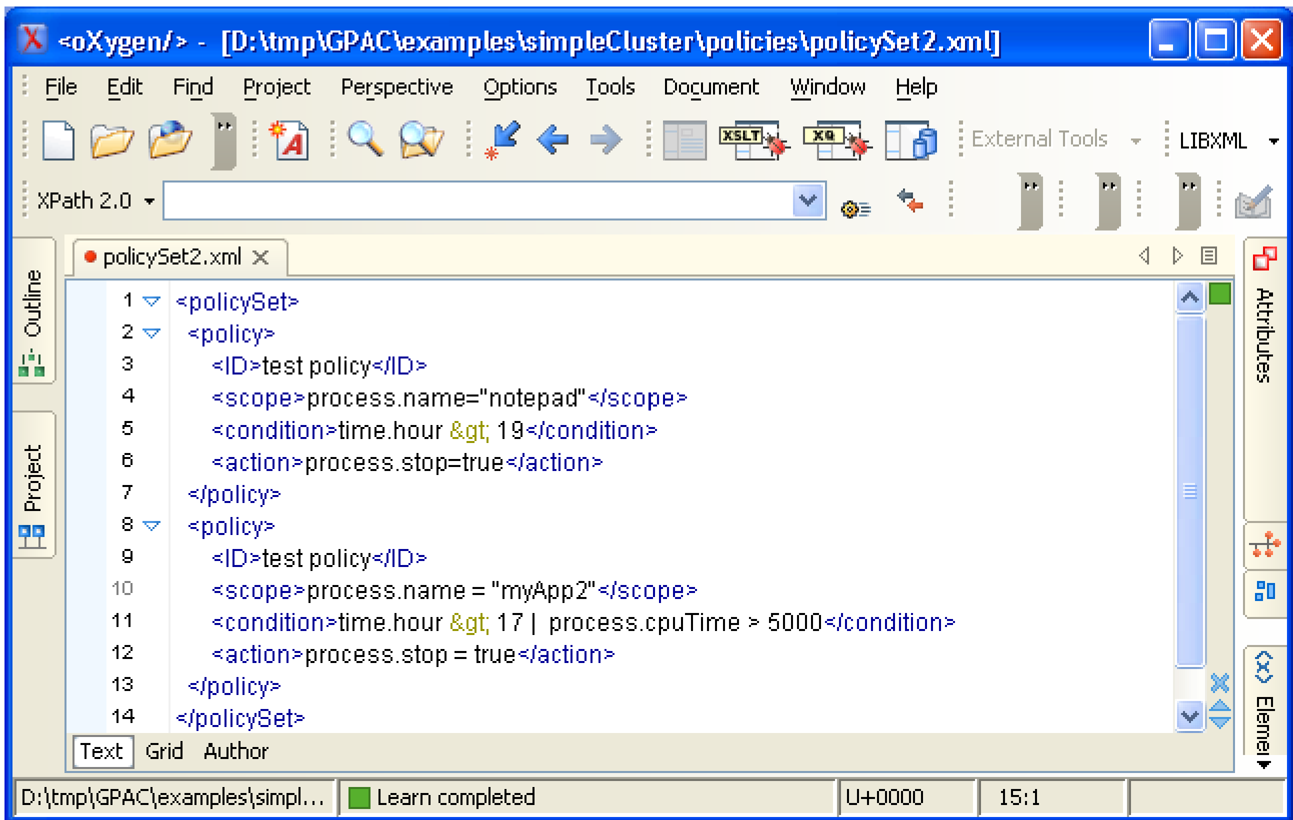


Fig. 23 Policy set for the sample application

The second of the two policies specifies that:

- "processes whose name is 'myApp2'" (i.e., the policy *scope*)
- should be stopped (i.e., the policy *action*)
- when the current time is later than 5pm or the process has consumed over 5000 units of CPU (i.e., the policy *trigger* or *condition*).

Again, you can use the admin tool to set up these policies – a snapshot of the tool after the policies were set is shown in Fig. 24.

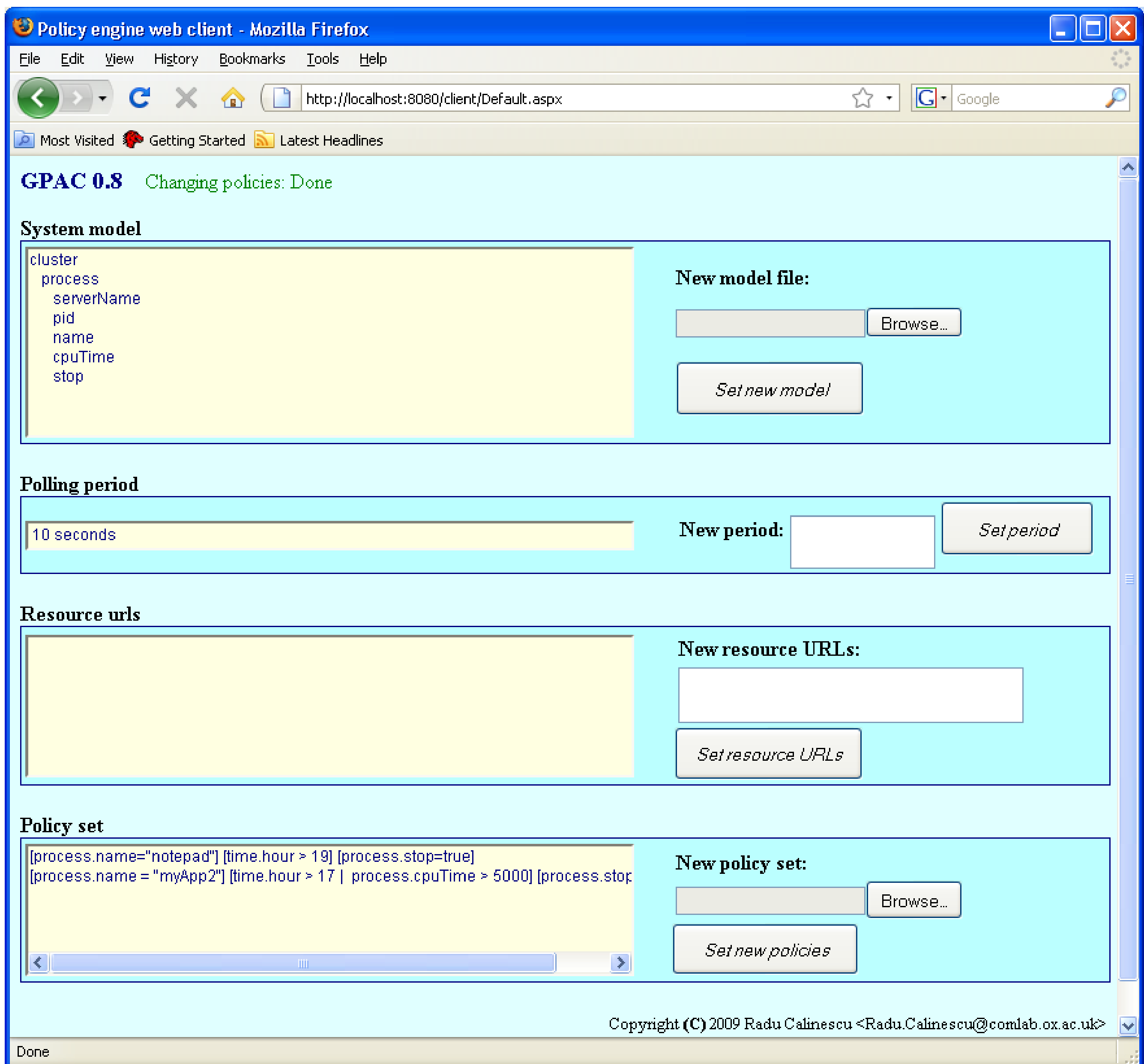


Fig. 24 Snapshot of the admin tool after setting a policy set for the sample application

Finally, in **Step E2** of the exploitation phase, the URLs of all the manageability adaptors in the system are supplied to the policy engine using the admin tool.

Before you specify the manageability adaptor URLs for the sample application: *Make sure that the servers on which you installed these adaptors contain no processes that you do not want to loose and which fall into the scope of any of the policies* ***and*** *satisfy the trigger of such a policy.*

Fig. 25 shows a snapshot of the admin tool after the URL of an installed instance of the process manageability adaptor for the sample application and the invalid string http://invalidURL were supplied to the policy engine (as a space-separated list of URLs typed into the "New resource URLs" area of the admin tool). Notice in Fig. 25 that the policy engine identified that the first URL corresponds to a "process" manageability adaptor, whereas the latter URL was invalid.

Note: Under this policy set, the 'notepad' processes running on the same server as the manageability adaptors will be killed if the time-dependent policy trigger is fulfilled.

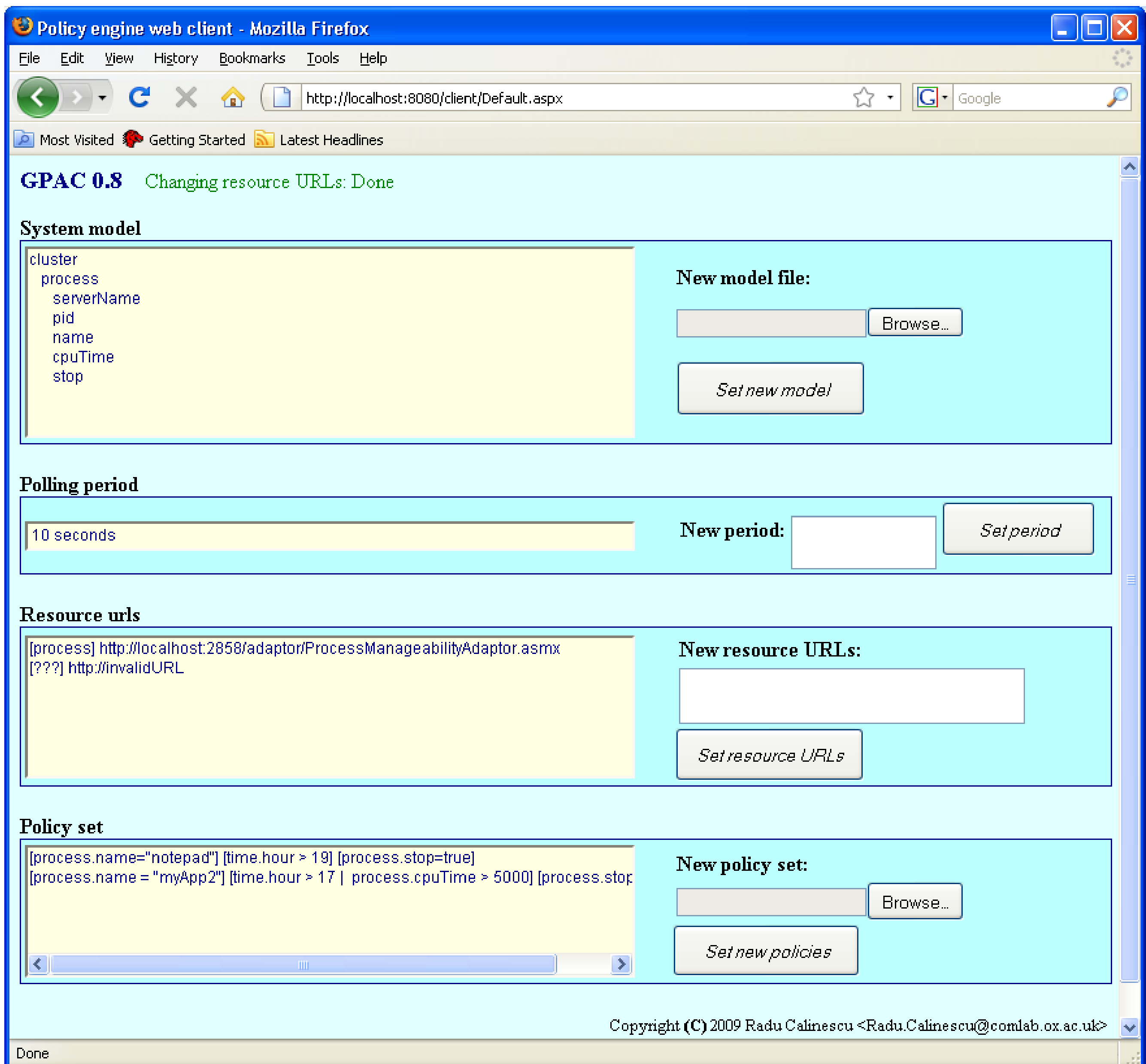


Fig. 25 Snapshot of the admin tool after manageability adaptor URLs were given by the user

5. Audit trail

The GPAC logging mechanism can be configured dynamically to generate a policy-evaluation audit trail, which is useful during both policy development and autonomic system operation. This logging is done using a hierarchy of log4net loggers with the structure in Figure 5.1.

Logger	**Generated entries**
`PolicyEvaluator`	policy evaluator start/end log entries
`PolicyEvaluator.Info`	start/end log entries for the policy evaluation steps
`PolicyEvaluator.Resources`	summary of system resources examined
`PolicyEvaluator.Resources.Details`	details of each resource
`PolicyEvaluator.Policy`	start/end log entries about individual policy evaluations
`PolicyEvaluator.Policy.Scope`	summary of system resources in the policy scope
`PolicyEvaluator.Policy.Scope.Details`	details of each resource identified to be in the policy scope
`PolicyEvaluator.Policy.Trigger`	log entry on the policy trigger
`PolicyEvaluator.Policy.Action`	log entries on read/write resource properties that were set

Fig. 5.1 Logger hierarchy for the policy evaluator

All audit-trail log messages are generated at level 'info', so the log4net configuration file `engine\log4net.Config` needs to be changed to turn on the logging of these messages. Several useful configurations are presented below – for a comprehensive descriptions of all options available, please see the log4net documentation [4]. Note that the log4net configuration file can be modified as many times as required during the operation of the autonomic manager; new configurations will take effect immediately.

5.1 Complete audit trail

Append the XML fragment below to the log4net configuration file

```
<logger name="PolicyEvaluator">
  <level value="INFO" />
</logger>
```

to obtain a full audit trail. The log4net configuration file must not contain other entries referring to the loggers in Fig. 5.1.

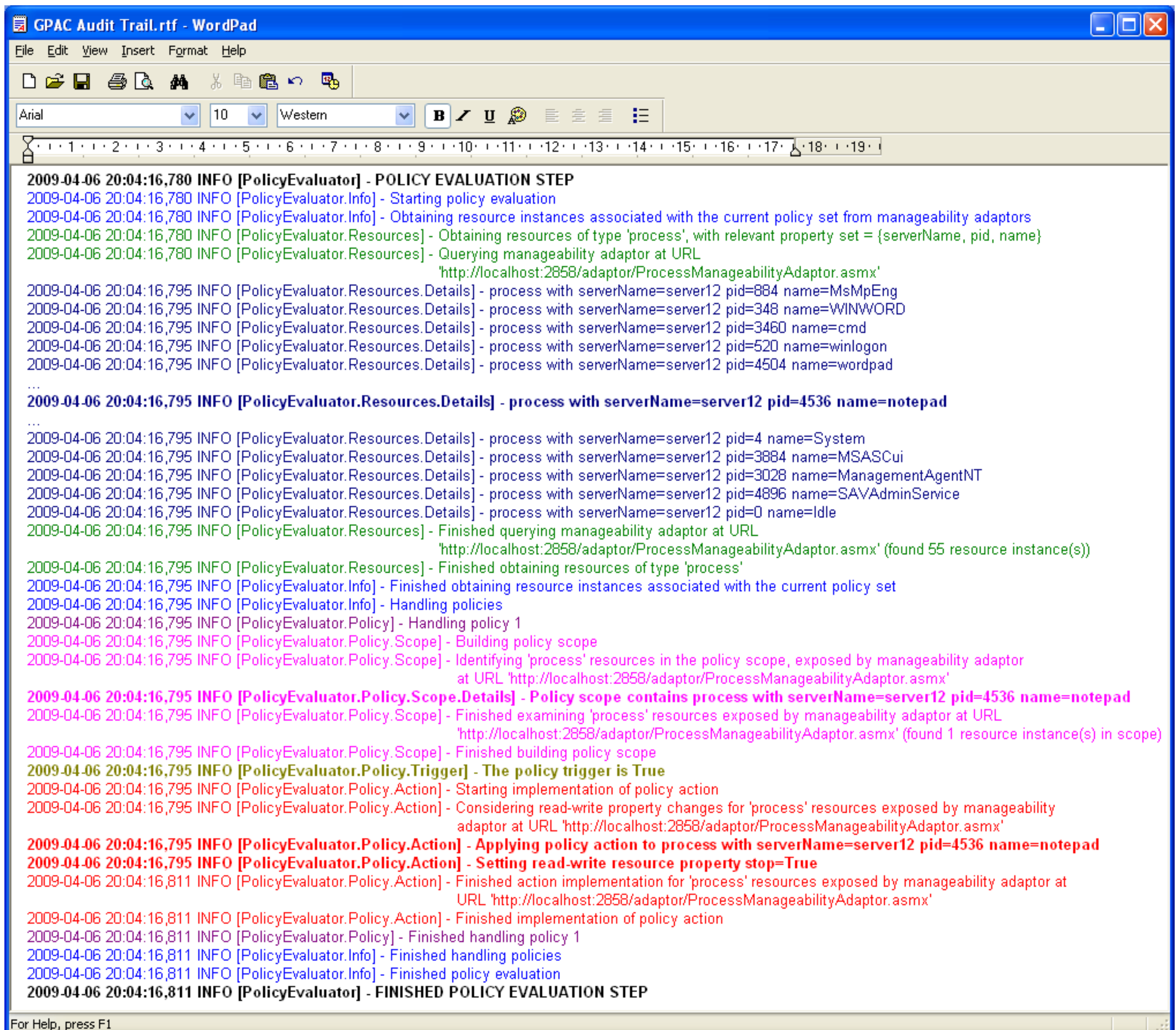
A fragment of the audit-trail log generated when this log4net configuration is used for the sample application is shown in Fig. 5.2.

5.2 Full policy details

Full details about the evaluation of all policies are obtained when the following XML fragment is included in the log4net configuration file:

```
<logger name="PolicyEvaluator.Policy">
  <level value="INFO" />
</logger>
```

The log4net configuration file must not contain other entries referring to the loggers in Fig. 5.1.



```
2009-04-06 20:04:16,780 INFO [PolicyEvaluator] - POLICY EVALUATION STEP
2009-04-06 20:04:16,780 INFO [PolicyEvaluator.Info] - Starting policy evaluation
2009-04-06 20:04:16,780 INFO [PolicyEvaluator.Info] - Obtaining resource instances associated with the current policy set from manageability adaptors
2009-04-06 20:04:16,780 INFO [PolicyEvaluator.Resources] - Obtaining resources of type 'process', with relevant property set = {serverName, pid, name}
2009-04-06 20:04:16,780 INFO [PolicyEvaluator.Resources] - Querying manageability adaptor at URL
                    'http://localhost:2858/adaptor/ProcessManageabilityAdaptor.asmx'
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources.Details] - process with serverName=server12 pid=884 name=MsMpEng
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources.Details] - process with serverName=server12 pid=348 name=WINWORD
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources.Details] - process with serverName=server12 pid=3460 name=cmd
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources.Details] - process with serverName=server12 pid=520 name=winlogon
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources.Details] - process with serverName=server12 pid=4504 name=wordpad
...
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources.Details] - process with serverName=server12 pid=4536 name=notepad
...
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources.Details] - process with serverName=server12 pid=4 name=System
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources.Details] - process with serverName=server12 pid=3884 name=MSASCui
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources.Details] - process with serverName=server12 pid=3028 name=ManagementAgentNT
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources.Details] - process with serverName=server12 pid=4896 name=SAVAdminService
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources.Details] - process with serverName=server12 pid=0 name=Idle
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources] - Finished querying manageability adaptor at URL
                    'http://localhost:2858/adaptor/ProcessManageabilityAdaptor.asmx' (found 55 resource instance(s))
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Resources] - Finished obtaining resources of type 'process'
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Info] - Finished obtaining resource instances associated with the current policy set
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Info] - Handling policies
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Policy] - Handling policy 1
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Policy.Scope] - Building policy scope
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Policy.Scope] - Identifying 'process' resources in the policy scope, exposed by manageability adaptor
                    at URL 'http://localhost:2858/adaptor/ProcessManageabilityAdaptor.asmx'
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Policy.Scope.Details] - Policy scope contains process with serverName=server12 pid=4536 name=notepad
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Policy.Scope] - Finished examining 'process' resources exposed by manageability adaptor at URL
                    'http://localhost:2858/adaptor/ProcessManageabilityAdaptor.asmx' (found 1 resource instance(s) in scope)
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Policy.Scope] - Finished building policy scope
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Policy.Trigger] - The policy trigger is True
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Policy.Action] - Starting implementation of policy action
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Policy.Action] - Considering read-write property changes for 'process' resources exposed by manageability
                    adaptor at URL 'http://localhost:2858/adaptor/ProcessManageabilityAdaptor.asmx'
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Policy.Action] - Applying policy action to process with serverName=server12 pid=4536 name=notepad
2009-04-06 20:04:16,795 INFO [PolicyEvaluator.Policy.Action] - Setting read-write resource property stop=True
2009-04-06 20:04:16,811 INFO [PolicyEvaluator.Policy.Action] - Finished action implementation for 'process' resources exposed by manageability adaptor at
                    URL 'http://localhost:2858/adaptor/ProcessManageabilityAdaptor.asmx'
2009-04-06 20:04:16,811 INFO [PolicyEvaluator.Policy.Action] - Finished implementation of policy action
2009-04-06 20:04:16,811 INFO [PolicyEvaluator.Policy] - Finished handling policy 1
2009-04-06 20:04:16,811 INFO [PolicyEvaluator.Info] - Finished handling policies
2009-04-06 20:04:16,811 INFO [PolicyEvaluator.Info] - Finished policy evaluation
2009-04-06 20:04:16,811 INFO [PolicyEvaluator] - FINISHED POLICY EVALUATION STEP
```

Fig. 5.2 Fragment of the full audit trail for the sample application, with colour-coded entries for the different loggers in the audit-trail logger hierarchy. The entries referring to the notepad 'process' resource in the policy scope are highlighted in bold font.

5.3 Policy evaluation steps and policy actions

The inclusion of the XML fragment

```
<logger name="PolicyEvaluator">
  <level value="INFO" />
</logger>
<logger name="PolicyEvaluator.Info">
  <level value="WARN" />
</logger>
<logger name="PolicyEvaluator.Resources">
  <level value="WARN" />
</logger>
<logger name="PolicyEvaluator.Policy">
  <level value="WARN" />
</logger>
<logger name="PolicyEvaluator.Policy.Action">
  <level value="INFO" />
</logger>
```

in the log4net configuration file will produce an audit trail that records the timed start/end of each policy evaluation step, and the actions enforced on the managed system. Note how the logging level for three of the loggers from Fig. 5.1 is raised to "WARN" (i.e., warning) to ensure that these loggers do not contribute ("INFO" log messages) to the audit trail.

Again, no other part of the configuration file should refer to the audit-trail loggers.

Note: One issue with the log4net version integrated in GPAC is that changing the audit trail configuration from

```
<logger name="PolicyEvaluator">
  <level value="INFO" />
</logger>

<logger name="PolicyEvaluator.Resources">
  <level value="WARN" />
</logger>
```

(i.e., generate all log entries except for those associated with the PolicyEvaluator.Resources logger and its descendents) to

```
<logger name="PolicyEvaluator">
  <level value="INFO" />
</logger>
```

(i.e., generate the full audit trail) does not work as expected. What happens is that the logging threshold for the PolicyEvaluator.Resources logger and its descendents remains WARN-ing instead of being changed to that of their ancestor logger PolicyEvaluator (i.e., INFO-rmation). To work around this issue, specify the new logging level for the adjusted logger explicitly:

```
<logger name="PolicyEvaluator">
  <level value="INFO" />
</logger>

<logger name="PolicyEvaluator.Resources">
  <level value="INFO" />
</logger>
```

6. Types of autonomic computing policies

Several policy types are typically used in autonomic computing systems:

- *action policies* provide a low-level specification of how the system configuration should be changed to match its state;
- *goal policies* specify precise constraints that should be met by varying the system configuration;
- *utility-function policies* specify a "measure of success" that the self-managing system should optimise by appropriately varying its configuration;
- *resource-definition policies* specify how the autonomic manager at the core of the autonomic system should expose the system to its environment.

The current version of GPAC supports action and utility-function policies, as described in the remainder of this section.

Notes:

1) The scope and condition/trigger components of all types of policies supported by GPAC use the same syntax and have the same semantics (described in Appendix B). The policy component that differs from one policy type to another is the policy action.

2) For historical reasons (in the early days of autonomic computing, action policies were the only type of autonomic computing policies), the term 'action' is semantically overloaded – it is used to denote a component of autonomic computing policies, as well as a type such policy. The sense it is used in should be obvious from the context.

6.1. Action policies in GPAC

GPAC action policies specify new values that read-write or write-only properties of resources instances in the scope of the policy should be set to if the policy condition holds. The running example of an autonomic computing application used in the previous sections of this user guide employs action policies (Figure 25).

6.2. Utility-function policies in GPAC

GPAC utility-function policies require that the autonomic manager decides the values of certain read-write or write-only properties of resources instances in the scope of the policy such as to maximise an expression that reflects the utility of the system.

The scope and condition/trigger components of a utility-function policy are similar to those of an action policy. The action component of a utility-function policy has the form in Table 6.1 below.

Table 6.1 Generic form of a utility-function policy action

MAXIMISE(	– keyword; tells that this is a utility-function policy action
set-comprehension,	– selects resources whose properties will be set (subset of policy scope)
arithmetic-expression,	– the utility function to maximise
config-property-list,	– list of resource properties to set to values that maximise system utility
bool-expression,	– constraint(s) that the new system configuration must satisfy
operational-model-expression	– operational model that he autonomic manager must use in its planning
)	

Note: The *arithmetic-expression* can be:

1. A *set-arithm-function*[2] such as "SUM(service, service.priority*service.throughput)", in which case the policy is termed a *global utility-function policy* and the autonomic manager is required to maximise the utility function across all resources in the policy scope. The sample application described in this section uses a global utility-function policy.
2. An *arithmetic-expression* that is not a *set-arithm-function*, e.g., "service.throughput – 0.2*service.allocatedCpu", in which case the policy is termed a *local utility-function policy* and the autonomic manager is required to maximise the utility function individually for each resource in the policy scope. The sample application presented in Section 6.3 employs a local utility-function policy.

We will illustrate the use of utility policies for the concrete example application from the `examples\server\` directory in the GPAC distribution.

Consider a server that runs several services of different priorities and variable workloads. Each service handles end-user requests that are received with different, variable inter-arrival time, and has a response time that depends on the amount of server resources dedicated to it. For illustration purposes, we will take this amount to be the percentage of CPU time allocated to the service, and assume that the server supports such precise partitioning of the CPU among services.

The only type of resource in the system described above is 'service' and its properties are:

Property	Description	Characteristics
name	Unique identifier of service	read only; primary key
priority	Numerical priority	read only
interArrivalTime	Request inter-arrival time (e.g., averaged over past 120s)	read only
cpuAllocation	Percentage of server CPU allocated to the service	read write
responseTime	Current response time in ms (e.g., averaged over past 120s)	read only

Further assume that the utility of each service depends on its response time and its priority as shown by the diagram in Figure 6.1.[3]

When all services running on the server are considered, the utility function to maximise is:

$$\sum_{service} service.priority \times \min(1000, \max(0, 2000 - service.responseTime))$$

The autonomic manager is required to maximise this utility function by deciding appropriate values for the 'cpuAllocation' property of all services, subject to the constraint:

$$\sum_{service} service.cpuAllocation \leq 100$$

i.e., the autonomic manager must not allocate more than 100(%) of the server CPU across all services. Figure 6.2 shows a graphical representation of this utility function when our server is running two services: a "premium" service of priority 100 and a "standard" service of priority 10.

---

[2] See **Appendix B.2** for further details.

[3] A concrete shape for the utility function and concrete values such as 1000ms and 2000ms were chosen for illustration purposes. This does not limit the generality of the approach.

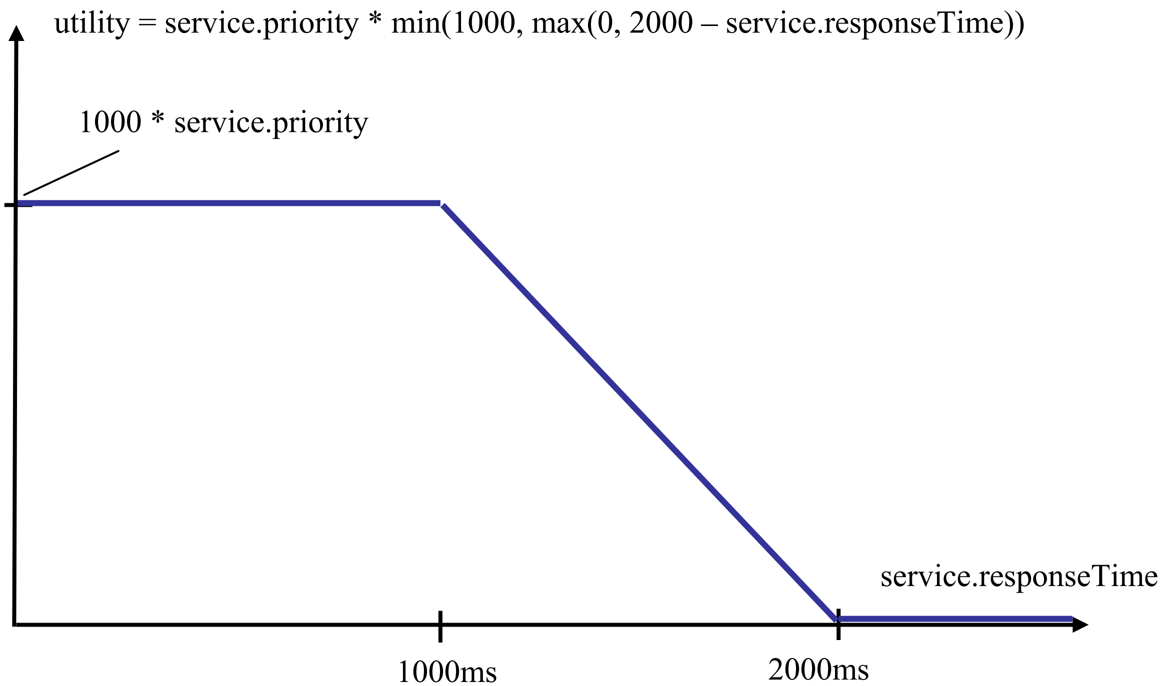


Fig. 6.1 The utility of a service is: (a) equal to its priority * 1000 if its response time is under 1000ms; (b) zero, if its response time is above 2000ms; (c) a linearly decreasing value for response times between 1000ms and 2000ms.

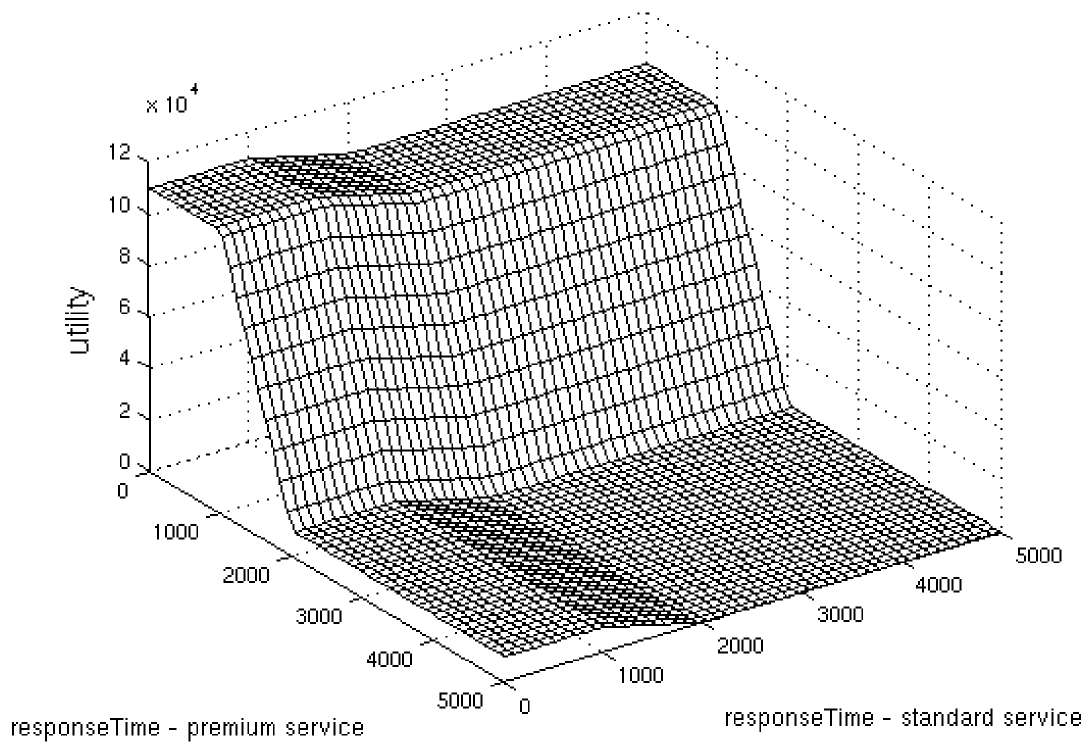


Fig. 6.2 Utility function for server running a premium service of priority 100 and a standard service of priority 10, shown for response times between 0 and 5000ms.

Based on the description of our example application so far, we can write its utility-function policy action as

```
MAXIMISE(
    service,                                                                      resources to manage
    SUM(service,service.priority*min(1000,max(0,2000-service.responseTime))),     utility function
    (service.cpuAllocation),                                                      resource property to set
    SUM(service,service.cpuAllocation)<=100,                                      constraint
    operational-model-expression                                                  operational model (see below)
)
```

The remainder of this section explains what an operational model of a resource is; how it is supplied to the autonomic manager; and how the autonomic manager uses such operational models to realise utility-function policies. The outstanding argument *operational-model-expression* of the policy action is explained at the end of Section 6.2.2.

6.2.1 Operational models

An operational model of a resource specifies how the state (i.e., the read-only properties) of the resource changes when its configuration (i.e., read-write and write-only properties) are modified (e.g., by the autonomic manager).

Consider a generic resource with n>0 state properties $(s_1, \ s_2, \ ..., \ s_n)$ and m>0 configuration properties $(c_1, \ c_2, \ ..., \ c_n)$. An operational model for the resource is a function f such that, given any current state of the resource $(s_1^0, s_2^0, \ ..., \ s_n^0)$ and any possible configuration $(c_1, \ c_2, \ ..., \ c_m)$ for the resource,

$$(s'_1, \ s'_2, \ ..., \ s'_n) = f(s_1^0, \ s_2^0, \ ..., \ s_n^0, \ c_1, \ c_2, \ ..., \ c_m) \qquad (6.1)$$

represents (an approximation of) the next state of the resource.

GPAC works with partial operational models of resources, i.e., with finite sets of points from the mapping f presented above. Such a partial operational model for a 'service' resource from our example application specify the expected response time of the service for a number of request inter-arrival time values and a number of possible values for the cpuAllocation configuration property of the service, e.g.:

$s'_1 =$	$f(s_2^0,$	c_1)
service.responseTime (i.e., s'_1)	service.interArrivalTime (i.e., s_2^0)	service.cpuAllocation (i.e., c_1)
…	…	…
2500ms	3000ms	5%
2200ms	3000ms	10%
…	…	…
500ms	3000ms	100%
2400ms	3500ms	5%
2350ms	3500ms	10%
…	…	…

The actual operational model approximation used for the example application was obtained by running multiple tests in which the service workload (i.e., request inter-arrival times) and CPU allocations were

varied across the entire range of possible values for these properties of a service. The resulting model is shown in Figure 6.3.

Given the current requested inter-arrival time of a service, the autonomic manager uses the (partial) operational model of the service to estimate the response times associated with various values it can assign to the cpuAllocation property of the service, and thus to decide which of these values to use. This is explained in more detail below.

6.2.2. Supplying an operational model to the autonomic manager

Partial operational models can be supplied to the autonomic manager through the manageability adaptors for the relevant resource instances. This requires that one of the read-only properties of the associated resource type is an array whose elements are tuples of the form

$$(s'_1,\ s'_2,\ \ldots,\ s'_n,\ s^0_1,\ s^0_2,\ \ldots,\ s^0_n,\ c_1,\ c_2,\ \ldots,\ c_m)$$

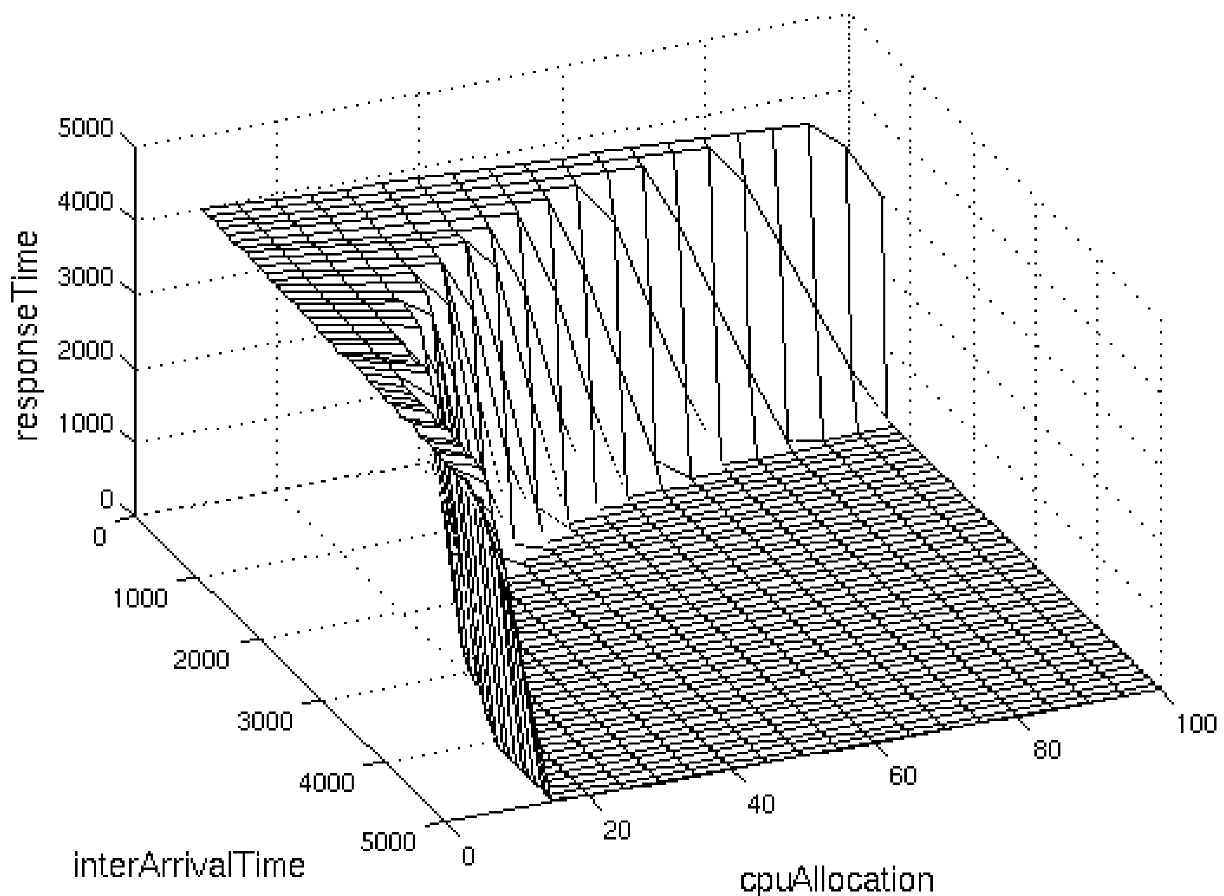


Fig. 6.3 Operational model of a service, obtained by averaging the results of multiple experiments

Returning to our example of a server running multiple services, the 'service' resource needs to have an additional '**operationalModel**' property. The specification of this property is illustrated by the abridged version of the system model `examples/server/model/server.xml` from the GPAC distribution shown below:

```
<system …>
 <name>server</name>

 <!-- Services running within a server -->
 <resource>
  <ID>service</ID>

  <property>
   <ID>name</ID>
   …
   <primaryKey>true</primaryKey>
  </property>

  <property>
   <ID>priority</ID>
   …
  </property>

  <property>
   <ID>cpuAllocation</ID>
   …
  </property>

  <property>
   <ID>interArrivalTime</ID>
   …
  </property>

  <property>
   <ID>responseTime</ID>
   …
  </property>

  <property>
   <ID>operationalModel</ID>
   <propertyDataType>
    <xs:element name="operationalModel" type="serviceOperationalModel" />
    <xs:complexType name="serviceOperationalModel">
     <xs:sequence>
      <xs:element name="modelElement" type="serviceModelElement" maxOccurs="unbounded" />
     </xs:sequence>
    </xs:complexType>
    <xs:complexType name="serviceModelElement">
     <xs:sequence>
      <xs:element name="responseTime" type="serviceResponseTime" />
      <xs:element name="interArrivalTime" type="serviceInterArrivalTime" />
      <xs:element name="cpuAllocation" type="serviceCpuAllocation" />
     </xs:sequence>
    </xs:complexType>
   </propertyDataType>
   <mutability>constant</mutability>
   <modifiability>read-only</modifiability>
   <subscribeability>false</subscribeability>
   <primaryKey>false</primaryKey>
  </property>

 </resource>

</system>
```

As indicated above, the name of the additional property must be spelt as shown above (i.e., ‘operationalModel’), and its type must be a unbounded sequence of complex-type elements comprising fields whose names and types are those of other resource properties.

The actual elements of the operational model must be supplied by the manageability adaptor for the resource. For our example application, the model shown in Figure 6.3 is stored as a list of comma-separated coordinates in the file `examples\server\adaptor\App_Data\model.txt` from the GPAC distribution:

```
500, 5, 4487
500, 10, 4458
500, 15, 4436
500, 20, 4415
500, 25, 4392
500, 30, 4373
…
```

On initialisation, the manageability adaptor code in `examples\server\adaptor\Global.asax` reads and parses the contents of this file, so that the manageability adaptor can supply it to the autonomic manager when requested to provide the 'operationalModel' property of a 'service' resource – the code can be examined in `examples\server\adaptor\App_Code\serviceManageabilityAdaptor.cs`.

Note: To run this example application successfully, the entry

```
<add key="operationalModelFile"
     value="C:\GPAC\examples\server\adaptor\App_Data\model.txt"/>
```

from the configuration file `examples\server\adaptor\Web.Config` for the manageability adaptor will need to be changed to reflect the actual deployment of the web service.

We are now ready to explain how the last argument of the MAXIMISE() expression for an utility-function policy action is used to specify to the autonomic manager the way in which the elements of the operational model fit the pattern given by equation (6.1) given at the beginning of Section 6.2.1. For the general case from equation (6.1) this is achieved by setting the *operational-model-expression* argument of the MAXIMISE() expression to

$$(s'_1,\ s'_2,\ \ldots,\ s'_n)\ (s_1^0,\ s_2^0,\ \ldots,\ s_n^0,\ c_1,\ c_2,\ \ldots,\ c_m),$$

which, for our example application, is:

```
(service.responseTime)(service.interArrivalTime,service.cpuAllocation)
```

This specifies that the response time of a service is a depends on its (request) inter-arrival time and the amount of CPU allocated to the service, so the complete policy action for the example application is given by

```
MAXIMISE(
    service,                                                          resources to manage
    SUM(service,service.priority*min(1000,max(0,2000-service.responseTime))),  utility function
    (service.cpuAllocation),                                          resource property to set
    SUM(service,service.cpuAllocation)<=100,                          Constraint
    (service.responseTime)(service.interArrivalTime,service.cpuAllocation)    operational model
)
```

This policy action asks the autonomic manager to decide the 'cpuAllocation' for a set of services such as to maximise the given utility function. The autonomic manager achieves this objective by using the operational model:

a) to assess their expected 'responseTime' (because 'responseTime' depends on the other properties)

b) for their current 'interArrivalTime' (because 'interArrivalTime' appears in the second term of the *operational-model-expression* **and** is not a "resource property to be set")
c) and for all possible values of their 'cpuAllocation' (because this is a "resource property to be set").

The policy presented above is included in the policy file `examples\server\policies\policy1.xml` from the GPAC distribution.

Also included in this example application from the GPAC distribution is a server simulator under `examples\server\simulator\`. This is a Windows application – the complete code is included as well as a binary `examples\server\simulator\bin\Debug\AppServerSimulator.exe`. The interaction between this simulator and the manageability adaptor consists in simulator calls to two additional web methods of the manageability adaptor – one to inform the adaptor about the current average request inter-arrival times and one to read the latest cpuAllocation values decided by the autonomic manager. In order to enable this interaction, the entry

```
<applicationSettings>
  <AppServerSimulator.Properties.Settings>
    <setting
        name="AppServerSimulator_ConfigurationService_serviceManageabilityAdaptor"
        serializeAs="String">
      <value>http://localhost:4027/adaptor/serviceManageabilityAdaptor.asmx</value>
    </setting>
  </AppServerSimulator.Properties.Settings>
 </applicationSettings>
```

from the configuration file `examples\server\simulator\app.config` for the simulator must be changed to reflect the actual URL of the manageability adaptor.

Figures 6.4 and 6.5 illustrate the overall architecture of the example application described so far and a sample run of the server simulator, respectively.

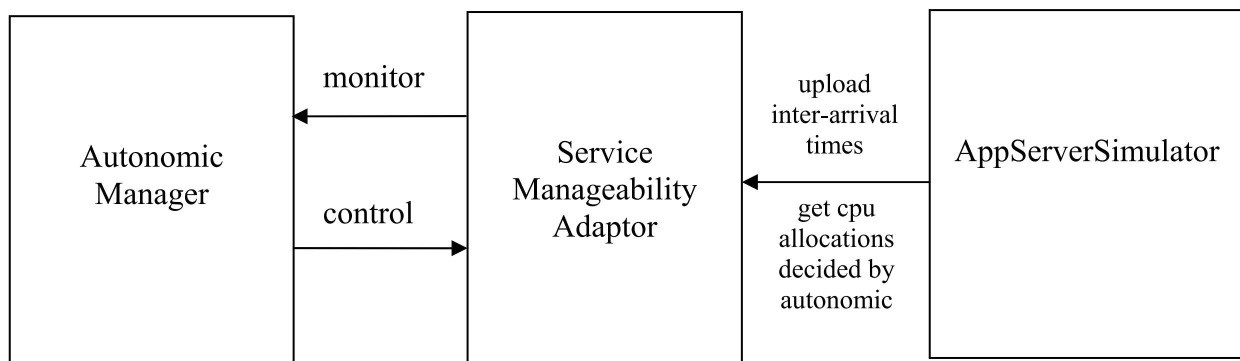


Fig. 6.4 The examples\server\ application from the GPAC distribution, with a summary of the three steps required to run the application.

Note: It is possible to have read-write and write-only resource properties specified in the *config-property-list* argument of a utility-function policy even when they are not involved in the operational model of the associated resource. The range of values to examine for such properties must be specified explicitly in the policy because the autonomic manager cannot use the operational modal to decide the value(s) for such properties. For instance, in the utility-function policy action

MAXIMISE(
set-comprehension,
arithmetic-expression,
(server.cfgPropertyA, server.cfgPropertyB(10.0 : 0.5 : 20.0)),
bool-expression,
(server.stateProperty)(server.cfgPropertyA)
)

server.cfgPropertyB is not part of the operational model. Therefore, the policy specifies that in order to decide the value for this property, the autonomic manager should analyse all the values that are 0.5 units apart between 10.0 and 20.0 (i.e., 10.0, 10.5, 11.0, …, 19.5 and 20.0).

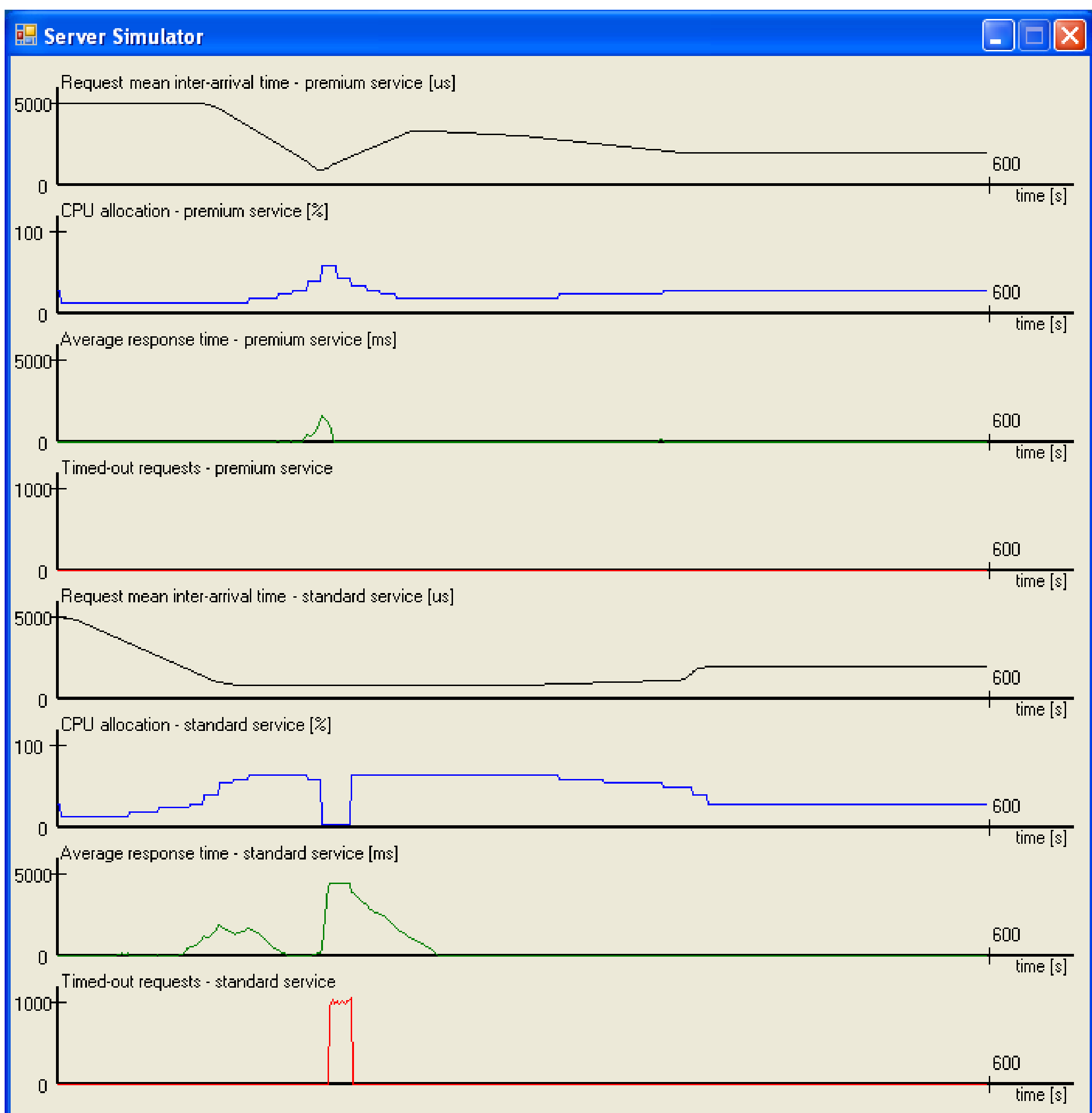


Fig. 6.5 Sample run of the server simulator – autonomic manager polling period 5s

6.3 Utility policies using PRISM quantitative analysis

PRISM [5] is a probabilistic model checker/quantitative analysis tool developed by the University of Oxford's Quantitative Analysis and Verification Group. The tool is used for the analysis of probabilistic models including discrete- and continuous-time Markov chains (DTMCs and CTMCs) expressed in the PRISM high-level, state-based language. Cost/reward-augmented versions of probabilistic computational tree logic (PCTL) and continuous stochastic logic (CSL) are used to specify the quantitative properties to analyse for DTMC and CTMC models, respectively.

The GPAC autonomic manager can use PRISM to support its implementation of utility-function policies as described in detail in [6]. The steps required to use this capability of GPAC are summarised below:

1. The probabilistic model checker PRISM needs to be set up on the server running the autonomic manager. The process involved is described in Appendix C.
2. A PRISM operational model describing the behaviour of the resources involved in the utility-function policy needs to be developed. This step is described below.
3. Like for ordinary utility-function policies (described in Section 6.2), the resource involved in the policy needs to have a read-only 'operationalModel' property, but the property must be of type 'string'. The value supplied by the manageability adaptor for this property must be the PRISM operational model from step 2 above.
4. The policy action of a PRISM-based utility-function policy has the same form as for an ordinary utility-function policy (see Table 6.1), except that the keyword 'PRISM_MAXIMISE' is used instead of 'MAXIMISE' at the beginning of the policy action expression.

Like before, the second argument of the PRISM-MAXIMISE policy action is the utility function, and it can make reference to:

- Read-write resource properties that do not appear in the *config-property-list*, and read-only resource properties. The values obtained from the manageability adaptors will be used wherever such properties are mentioned.
- Derived resource properties (i.e., resources whose *modifiability* is specified as "derived" in the system model). The values of these properties cannot be read or written through the manageability adaptor. Instead, these values are obtained through the PRISM analysis of the operational model for the resource. This is explained below.
- Read-write resource properties that appear in the *config-property-list*. Each property in the list must be followed by a range specifier of the form '(*start* **:** *step* **:** *stop*)' as explained in the note at the end of the previous section in this document. Note that when PRISM is used in the autonomic manager analysis step, such ranges must be specified for all resource properties in the *config-property-list*, whether or not they are part of the operational model.

The GPAC distribution includes a PRISM-based autonomic application that involves the dynamic, adaptive power management of a disk drive. This application – located in `examples\diskDrive\` and described in detail in [6] – ensures that the probability of the disk drive being taken into a low-power `sleep' state is permanently adjusted in line with the inter-arrival time for the I/O requests that the disk is handling. Thus, energy can be saved when the disk drive is lightly loaded, without compromising performance when the disk drive request inter-arrival time decreases.

To try this application, supply to the autonomic manager:

(a) the system model `examples\diskDrive\model\diskDrive.xml`
(b) the policy `examples\diskDrive\policies\policy1.xml`

(c) the URL of a running instance of the manageability adaptor `examples\diskDrive\adaptor`

* make sure that the Web.Config entry

```
<add key="operationalModelFile"
  value="D:\Users\Radu\GPAC\examples\diskDrive\adaptor\App_Data\diskDrive.sm"/>
```

is updated to specify the actual location of the PRISM operational model diskDrive.sm on your server.

Also, set the autonomic manager polling period to 5 seconds. Running the disk-drive simulator at examples\diskDrive\simulator after ensuring that the entry

```
<setting
    name="AppServerSimulator_ManageabilityAdaptor_diskDriveManageabilityAdaptor"
    serializeAs="String">
  <value>http://localhost:2358/adaptor/diskDriveManageabilityAdaptor.asmx</value>
</setting>
```

in the app.config configuration file for the simulator is updated to reflect the URL of the manageability adaptor on your system will then yield the results in Figure 6.6.

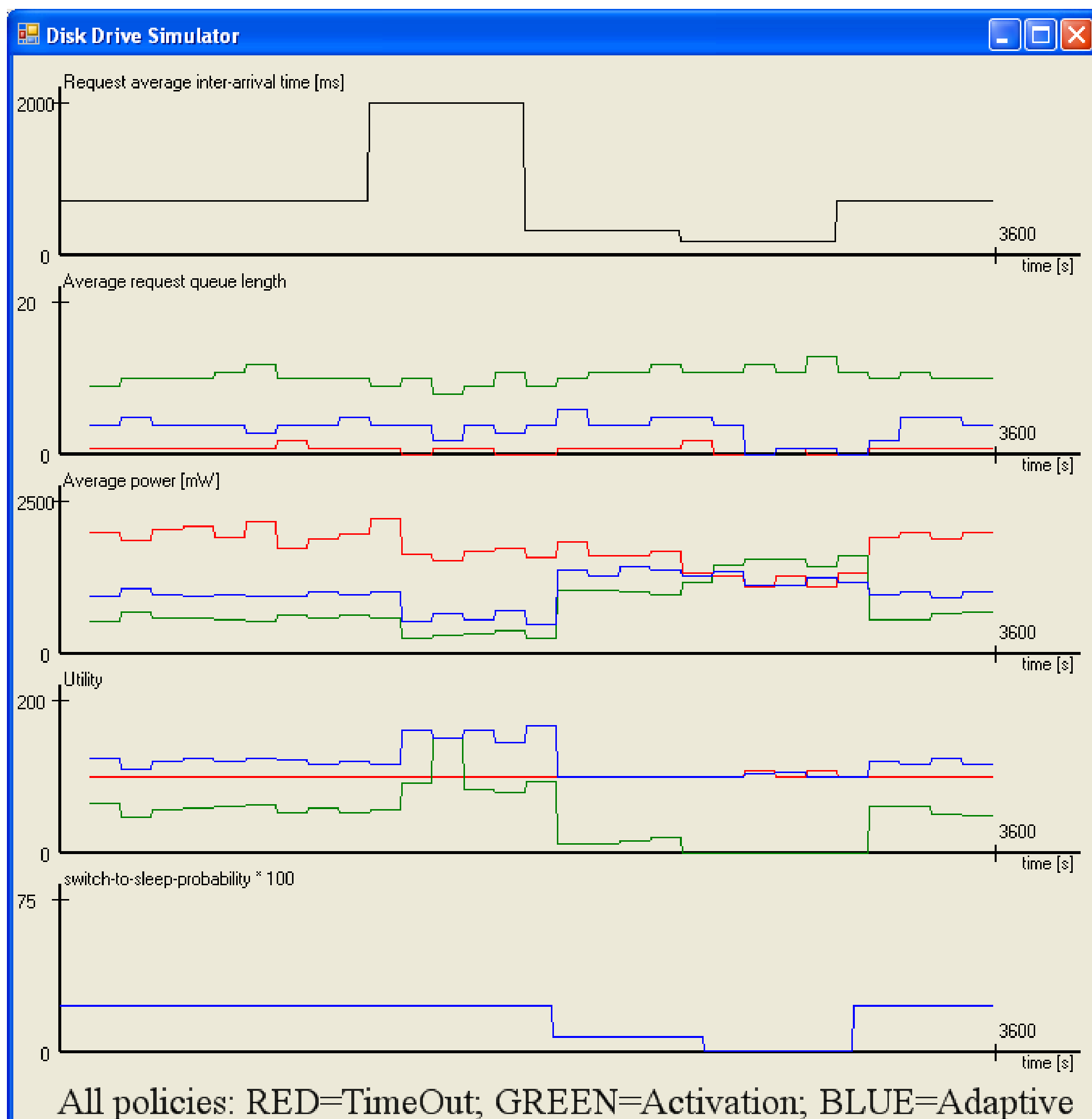


Fig. 6.6 Dynamic power management of disk drive using PRISM-based quantitative analysis for the implementation of utility-function autonomic computing policies

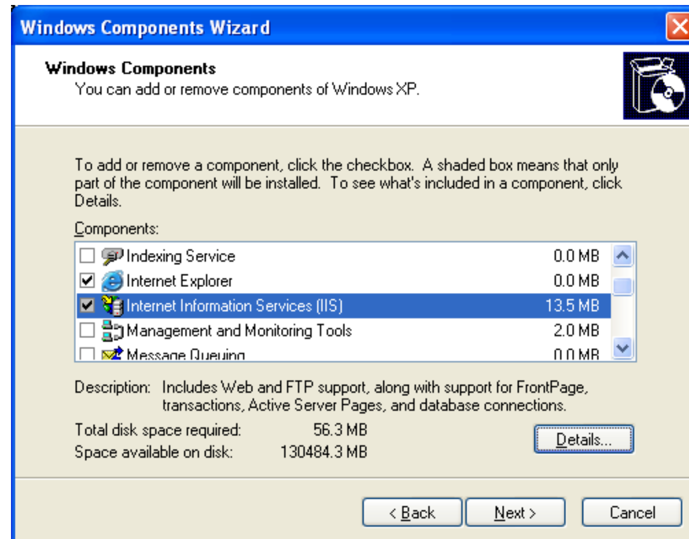
References

[1] W.E. Walsh et al. Utility functions in autonomic systems. In: *Proceedings of the 1st IEEE International Conference on Autonomic Computing*, pp. 70–77, 2004.
[2] R. Calinescu – General-Purpose Autonomic Computing. In: M. Denko *et al* (editors), *Autonomic Computing and Networking*, Springer, New York, 2009, pp. 3–30.
[3] Steve Litt – Perl Regular Expressions, http://www.troubleshooters.com/codecorn/littperl/perlreg.htm.
[4] Apache log4net, http://logging.apache.org/log4net/index.html.
[5] PRISM probabilistic model checker, http://www.prismmodelchecker.org/.
[6] R. Calinescu and M. Kwiatkowska – Using Quantitative Analysis to Implement Autonomic IT Systems. Proceedings of the 31st International Conference on Software Engineering, 2009. Available from http://qav.comlab.ox.ac.uk/bibitem.php?key=CK09.

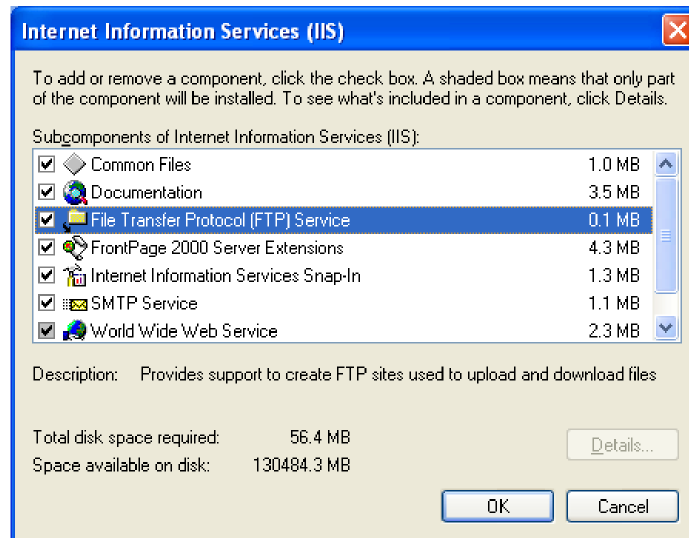
Appendix A. Installing IIS and .NET on a server

Install IIS (May need Windows XP CD)

1. Start → Control Panel → Add or Remove Programs → Add/Remove Windows Components



2. Click "Details" and tick everything (Documentation, FTP, and SMTP are optional)



3. OK → Next → Finish

Install .NET Framework 2.0 Beta 2

1. Download and install .NET framework
   - http://msdn.microsoft.com/netframework/downloads/updates/default.aspx
2. Register .NET with IIS
   - Open command prompt and navigate to C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxxxx\
3. run "aspnet_regiis.exe –i" to register .NET into IIS

Appendix B. GPAC autonomic computing policies

This appendix describes the syntax and semantics of the policies supported by the GPAC reconfigurable policy engine. Note, however, that this appendix is work under development.

B.1 Overview

The GPAC version described in this guide supports policies comprising three components:

1. Policy scope. This is a semicolon-separated list of simple set comprehension expressions

    scope `::=` *set-comprehension* ; *set-comprehension* ; … ; *set-comprehension*

    where the generic form of a set comprehension expression is

    set-comprehension `::=` *resource-type* | *boolean-expression*

    Either term (but not both terms) can be missing from a set comprehension expression, i.e., the policy scope

    server ; process.name = "foo"

    is equivalent to

    server | `true` ; process | process.name = "foo"

    Boolean expressions are conjunctions (i.e., '&') of boolean terms, which are disjunction (i.e., '|') of potentially negated (i.e., '!') boolean-typed resource properties (e.g., 'process.isActive'), boolean constants (i.e., `true` or `false`), relational expressions (described below), paranthesised boolean expressions (i.e., '(*boolean-expression*)') or regular expressions of the form

    resource-property `=~` *"pattern"*

    or

    resource-property `!~` *"pattern"*

    The first regular expression returns true if the value of the resource property matches the pattern and the second regular expression returns true if the value does not match the pattern provided – see [3] for more information on using regular expressions.

    Relational expressions can be used to compare any types of expressions, including arithmetic expressions involving resource properties.

2. Policy trigger (aka 'policy condition') This is a boolean expression that refers to resource properties and/or properties of *built-in policy engine variables*. The only built-in variable supported by the GPAC framework version described in this guide is '`time`', with the properties: year, month, day, hour, minute, second.

3. Policy actions. The GPAC version described in this guide supports semicolon-separated list of assignments, where the left-hand-side term of the assignment is a resource property and the right-hand-side term is an expression of the appropriate type.

B.2 Policy syntax

policy	::=	*policy-scope policy-trigger policy-action*
policy-scope	::=	*set-comprehension* | *set-comprehension* **;** *policy-scope*
set-comprehension	::=	*resource-type* | *resource-type* **|** *bool-expression* | *bool-expression*
bool-expression	::=	*bool-term* | *bool-term* **|** *bool-expr*
bool-term	::=	*bool-factor* | **!** *bool-factor* | *bool-factor* **&** *bool-term* **!** *bool-factor* **&** *bool-term*
bool-factor	::=	*bool-constant* | *resource-property* | *selected-res-property* | **(** *bool-expression* **)** | *relational-expression*
bool-constant	::=	**true** | **false**
resource-property	::=	*resource-type* **.** *property-id*
resource-type	::=	`ID of resource defined in the system model`
property-id	::=	`ID of resource property from the system model`
selected-res-property	::=	**ARGMAX(** *set-comprehension* **,** *arithmetic-expression* **).** *property-id* | **ARGMIN(** *set-comprehension* **,** *arithmetic-expression* **).** *property-id*
relational-expression	::=	*arithmetic-expression arithm-rel-operator arithmetic-expression* | *bool-expression bool-rel-operator bool-expression* | *string-expression string-rel-operator string-expression*
arithmetic-expression	::=	*arithmetic-term* | *arithmetic-term* **+** *arithmetic-expression* | *arithmetic-term* **–** *arithmetic-expression*

arithmetic-term	::=	*arithmetic-factor* | *arithmetic-factor* `*` *arithmetic-term* | *arithmetic-factor* `/` *arithmetic-term*
arithmetic-factor	::=	*numerical-constant* | *resource-property* | *selected-res-property* | *built-in-property* | `(` *arithmetic-expression* `)` | *function*
numerical-constant	::=	`0` | `[1-9][0-9]*` | `.[0-9]+` | `0.[0-9]+` | `[1-9][0-9]*.[0-9]+`
build-in-property	::=	**`time.year`** | **`time.month`** | **`time.day`** | **`time.hour`** | **`time.minute`** | **`time.second`**
function	::=	*math-function* `(` *arithmetic-expr-list* `)` | *set-arithm-function* `(` *set-comprehension* `,` *arithmetic-expression* `)` | **COUNT** `(` *set-comprehension* `)`
math-function	::=	**`min`** | **`max`**
arithmetic-expr-list	::=	*arithmetic-expression* `,` *arithmetic-expression* | *arithmetic-expression* `,` *arithmetic-expr-list*
set-arithm-function	::=	**SUM** | **PROD** | **MAX** | **MIN** | **MEAN**
arithm-rel-operator	::=	`=` | `!=` | `<` | `<=` | `>` | `>=`
bool-rel-operator	::=	`=` | `!=`
string-expression	::=	*string-constant* | *resource-property*
string-constant	::=	*`"-delimited string of ASCII characters`*
string-rel-operator	::=	`=` | `!=` | `=~` | `!~`
policy-trigger	::=	*bool-expression*
policy-action	::=	*assignment-action* | *utility-function-action*
assignment-action	::=	*assignment-expression* | *assignment-expression* `;` *assignment-action*
assignment-action	::=	*resource-property* `=` *expression*
expression	::=	*arithmetic-expression* | *bool-expression* | *string-expression*

utility-function-action ::= *utility-function-type* (*set-comprehension*, *arithmetic-expression*, (*config-property-list*), *bool-expression*, *op-model-expr*)

utility-function-type ::= **MAXIMISE | PRISM_MAXIMISE**

config-property-list ::= *config-property-spec* | *config-property-spec*, *config-property-list*

config-property-spec ::= *resource-property* | *resource-property*(*numerical-constant* **:** *numerical-constant* **:** *numerical-constant)*

op-model-expr ::= (*resource-property-list*) (*resource-property-list*)

resource-property-list ::= *resource-property* | *resource-property*, *resource-property-list*

B.3 Policy semantics

B.3.1 Policy scope

The scope of a policy is a semicolon-separated list of set comprehension expressions, each of which selects a set of system resources. Given that a policy can involve resources of different types, these set comprehension expressions can refer to different types of resources, as indicated by the diagram in Figure B.1.

The resources involved in the policy are those that belong to one of the three sets selected by the set comprehension expressions in the policy scope:

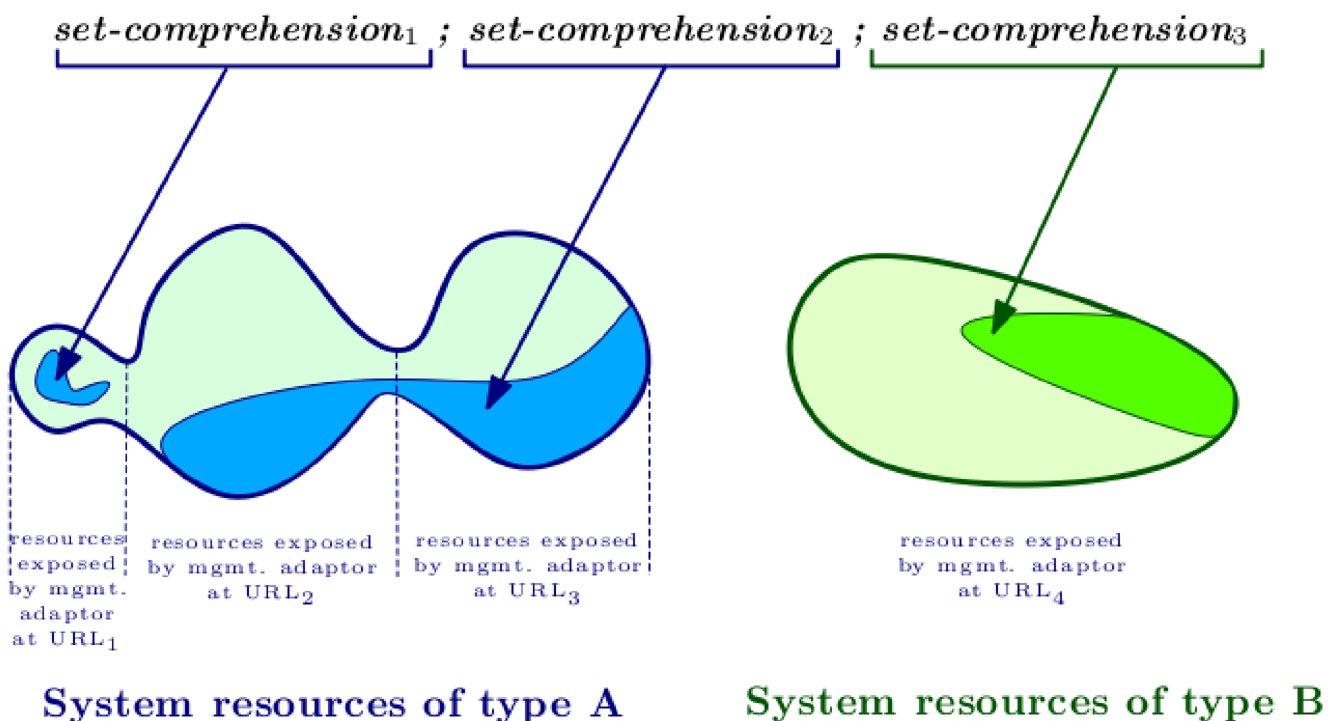


Fig. B.1 The scope of a policy for a system comprising two types of resources, A and B. Four manageability adaptors expose the resources of the systems – those at URLs 1 to 3 expose resource instances of type A, while the adaptor at the fourth URL exposes resource instances of type B. The policy scope comprises three set comprehension expressions, the first two of which select different subsets of system resources of type A; the last set comprehension expression selects some of the type B resource instances exposed by the manageability adaptor located at URL_4.

B.3.2 Policy condition

Given the set of resource instances in the scope of a policy, the policy condition is a **global boolean expression** over all these resource instances. The policy action is implemented *if and only if* the policy condition evaluates to `true`.

Several examples of policy conditions are presented below:

Policy condition	Description
$time.hour \geq 17 \quad \& \quad time.minute \geq 30$	Implement policy action if the current time is 5:30pm or later
$SUM(process, process.cpuUtilisation) \geq 60$	Implement policy action if the aggregated CPU utilisation of all processes **in the policy scope** is greater than or equal to 60%
$COUNT(harddisk) > 0$	Implement policy action if the policy scope contains more than 0 'harddisk' resource instances
true	Always implement policy action

B.3.3 Policy action

Policy actions are sequences of semicolon-separated assignments to read-write or write-only resource properties. All resource instances in the policy scope that match the resource type on the left-hand side of a policy action assignment expression have their respective properties set to the value of the expression on the right. For instance, for the sample policy action

process.stop = `true`

all processes in the policy scope will have their 'stop' property set to `true`.

Note:

Some resource instances in the policy scope may not appear on the left-hand side of the assignment, e.g., in the policy action

report.cpuUsage = SUM(process, process.cpuUsage)

resource instances of type process are used to obtain the value for a read-write property of another type of resources. Similarly, some of the resources in the policy scope may only contribute to the policy condition expression, without appearing in the policy action expression.

Appendix C. Setting up PRISM within GPAC

1. Follow the instructions at http://www.prismmodelchecker.org/download.php to download and install PRISM on the same server on which you are running the GPAC autonomic manager.
2. In the Web.Config configuration file for the autonomic manager, update the configuration entry

   ```
   <add key="prismPath"
        value="D:\Personal Program Files\prism-3.1.1.r542\bin\prism"/>
   ```

   so that it specifies the actual location of the PRISM deployment on your server.

Appendix D. Known limitations

Support for goal policies is not available in version 0.13 of the GPAC framework, but will be reintroduced in future versions of the framework, as described in [2]. However, note that goal policies can be simulated with utility-function policies, by using a utility function that takes only two values – 0 for argument values for which the goal function would have been false, and 1 otherwise.

Setting a new system model automatically removes the policy set in place, which is the right thing to do if the model refers to new types of resources, but may be undesirable if the system model is just a fine-tuned version of the previous model.

Manageability adaptors URLs are checked only when first supplied to the autonomic manager. If the manageability adaptors are not switched on when the URLs are specified to the autonomic manager, they will be marked as invalid, and not used once they are started. The workaround is to re-specify the same URLs after the manageability adaptors are started.