Project N°:	**FP7-610582**
Project Acronym:	**ENVISAGE**
Project Title:	**Engineering Virtualized Services**
Instrument:	**Collaborative Project**
Scheme:	**Information & Communication Technologies**

Deliverable D4.4.3
Assurance of the ENG Case Study

Date of document: T34

Start date of the project:	**1st October 2013**
Duration:	**36 months**
Organisation name of lead contractor for this deliverable:	**ENG**

Final version

STREP Project supported by the 7th Framework Programme of the EC		
Dissemination Level		
PU	Public	✓
PP	Restricted to other programme participants (including Commission Services)	
RE	Restricted to a group specified by the consortium (including Commission Services)	
CO	Confidential, only for members of the consortium (including Commission Services)	

Executive Summary:
Assurance of the ENG Case Study

This document constitutes deliverable D4.4.3 of project FP7-610582 (Envisage[1]), and addresses *quality assurance* for the Engineering case study. Details of the case study have already been presented in deliverables D4.4.1 (overview & objectives) and D4.4.2 (resource modelling). This deliverable reports just on the results of applying the Envisage toolset to the ABS model developed for the case study. The specific tools applied were: *Model Simulation* (i.e. execution in the Erlang Backend for ABS); *Deadlock Analysis*; *Resource Analysis* (specifically *w.r.t. computational cost*); and *Automatic Java Code Generation*. Various versions of the tools were tested, and the overall results were mixed. In sum, using the tools, we were able to satisfy approximately half of the case study's *quality assurance* objectives.

List of Authors

Keven T. Kearney (ENG)
Domenico Presenza (ENG)

List of Contributors

Stefania D'Agostini (ENG)
Elvira Albert (UCM)
Francesco Iadanza (ENG)
Enrique Martín (UCM)
Antonio Flores Montoya (TUD)
Claudia Pandolfo (ENG)
Rudolf Schlatte (UIO)
Vlad Serbanescu (CWI)
Francesco Torelli (ENG)

[1] Envisage is a Collaborative Project supported by the 7th Framework Programme of the EC within the Information & Communication Technologies scheme. Full information is available online at http://www.envisage-project.eu.

1. Introduction

This deliverable focuses on quality assurance for the Engineering case study through the application of the analysis techniques developed in other work packages. The case study itself has already been described in deliverables D.4.4.1 *Initial Modelling of the ENG Case Study* and D4.4.2 *Resource Aware Modelling the ENG Case Study*, and we assume that readers are familiar with both these deliverables. By way of brief recap:

The Engineering case study concerns "ETICS", an online code build and test service for software developers, that is able to dynamically exploit on-demand, distributed computational resources (virtual machines, or VMs) to 'optimally' satisfy service requests. The elastic pool of VMs is managed, in real-time, by an automated Resource Pool Manager (RPM), which analyses incoming service requests, to determine the best set of VMs to deploy, and the best allocation of request processing tasks to these VMs. The term 'optimal' here denotes 'most profitable', and is formally defined over (monetary) cost & penalty terms specified in ETICS consumer- and VM provider-facing SLAs. As detailed in D4.4.2, the RPM uses a distributed genetic algorithm (DGA) to achieve this task, hence 'optimal' is also intended in an heuristic sense.

As also reported in D4.4.2 the ENG case study aims at applying the ABS analytic tools to meet the following three key objectives:

1. **To ensure (if possible) that the DGA is deadlock free:**

   In parallel with the development of the ABS model for ETICS, we also built a working prototype of the ETICS service simulator in Swift. On most runs, however, this prototype stalled, with output frozen and CPU usage dropping to zero. Application of the Envisage deadlock analysis tools to the ABS model may help to show whether the stalling of the Swift prototype is due to logical/scheduling problems in the DGA, or to other factors (e.g. a quirk of the Swift execution).

2. **To ensure that the DGA scales:**

   The *distributed* algorithm (DGA) has only been tested in simulation on a *serial* machine, and with only small numbers (up to 1000) of requests and VMs. Testing on a larger scale on a single machine is impractical (the CPU & memory requirements are too high), and over multiple machines too costly. The intention, therefore, is to use formal ABS analytic methods (if possible) to assess the scalability of the DGA under high/extreme loads.

3. **Automatic generation of Java code:**

   The ETICS service is implemented in Java. The goal of using the Envisage Java code generation tools is to provide a *proof of concept* assessment of the feasibility of adopting ABS in the service production lifecycle. In principle, elements of the service stack could be developed in ABS (with guaranteed properties - *cf.* previous bullets), then automatically translated into executable Java components (that verifiably conform to the guaranteed properties of the ABS model), which could then be deployed as part of the live service implementation.

This deliverable reports on progress towards these objectives, and is structured as follows:

Section 2 - Code Simulation (Erlang)
: Summarises the results of compiling and executing the ABS model for the ETICS service using the Erlang backend for ABS;

Section 3 - Deadlock Analysis
: Describes the results of applying the Envisage *deadlock analysis* tools to the ABS model;

Section 4 - Cost Analysis (for Scalability)
: Describes the results of applying the Envisage *resource analysis* tools to the ABS model;

Section 5 - Java Code Generation
: Describes the results of applying the Envisage *code generation* (and code conformance verification) tools to the ABS model;

Section 6 - Summary & Conclusion

The following subsections very briefly report on the current status of the ABS Model for ETICS, and on Engineering's general experience *w.r.t.* installation and use of the ABS tools.

1.1. Status of the ABS Model for ETICS

The ABS model referred to in this deliverable is a significantly revised version of the ABS model submitted with, and described in, deliverable D4.4.2. At a high-level, the architecture and operation of the model remains unchanged, but there have been many modifications at the detailed code level. Of particular note:

- At the time D4.4.2 was submitted, it was not possible to execute the original ETICS model due to critical ABS language features not being supported by the available ABS backends[2]. These features have since been implemented, and the model can now be executed. The initial execution trials revealed various bugs in the original model that have been resolved in the revised version.
- The ABS language specification and core library[3] have been under continuous development since D4.4.2, and the ABS model has been updated to reflect these changes.
  - In particular, we note that Engineering's previous suggestions for improving the language (*see* D4.4.2, §4.3) have been accepted and adopted.
  - At the time of writing, there is no official release schedule for the ABS language & tools. The trials presented in sections §2 to §5 are based on the state of the language and analysis tools towards the end of July 2016.
- Finally, we learned that the Java code generation tools are not able to handle **`case`** statements in functional data types. Accordingly, these statements have been removed from the original code and replaced with suitable alternatives.

A handful of tool-specific changes to the ABS model will be described in subsequent sections.

1.2. General Experience *w.r.t.* ABS Tools

Engineering encountered several practical problems in installing and using the ABS tools - primarily:

- Inaccurate and out-of-date documentation (we are aware that the developers are currently taking steps to rectify this).
- Problems deploying the local (Vagrant[4]/VirtualBox[5] hosted) version of the Collaboratory. Despite great effort it proved impossible to deploy the Collaboratory in Engineering's office environment (Windows PCs with proxy mediated internet access). Installation was *only* possible working from home (without a proxy) on employee's personal (non-Windows) machines - entirely at their own risk. We were, however, able to use the online version of the Collaboratory at http://ei.abs-models.org:8082/clients/web/.
- Unavailability of the Java Code Generation tools (see §5) - which, at the time of writing, have not been integrated into the ABS tool chain. The tool developers, however, have access to the ABS model for ETICS, and we are currently awaiting feedback on their progress in translating the model to Java.

All the ABS language and tool developers, however, have been extremely responsive to the practical issues we encountered, and have spent considerable effort trying to resolve them.

In the remainder of this deliverable we report just the substantial results of using the ABS tools.

[2] Most significantly, the object method **`thisDC`**, to access the **`DeploymentComponent`** on which the object resides.

[3] Available from the GitHub repository at https://github.com/abstools/abstools.git.

[4] See: https://vagrantcloud.com.

[5] See: https://www.virtualbox.org.

2. Code Simulation (Erlang)

As explained in previous deliverables (D4.4.1 & D4.4.2), the ABS model for Engineering's ETICS case study is a simulation: abstracting from technical details of the ETICS service, and serving as a tool for ETICS business managers to fine tune customer-facing SLAs. Using the tool, managers can assess the performance of the service - under different SLA contingencies - with randomly generated service requests. In the ABS version, all the SLA parameters, and request generation policies, are hard-coded.

The complete revised code for the ABS model is given in Appendix A. The model comprises around 1600 lines of code, and (we are informed) is by far the largest and most complex ABS model thus far implemented. Execution of the model was tested using the Erlang[6] backend for ABS, invoked from the Terminal application in Mac OS 10.11. The remainder of this section describes the results.

For small test runs (simulating only a small number of requests), the code compiles and executes successfully. Table 2-*a* below, for example, shows the console output (with explanations), for a test run of just *one* simulated request to the ETICS service. This output demonstrates that the code, at least in general outline, is working as intended.

Table 2-a: Console Output for a Single Request Test Run

Console Output	*Explanation*
`START`	*Start of the main simulation loop - which generates random requests at random times, and posts them to the RPM.*
`RECEIVED:RQ(id=1,size=117071231/5000000,pri=1/2)`	*The RPM receives the first request, with the 'size' and 'priority' values shown.*
`END`	*End of the simulation - meaning just that all simulated requests have been generated.*
`LAUNCHED:VM(id=1,power=74/5,disk=32)`	*The RPM (which continues running on a separate COG)* ***launches*** *a new VM, with the 'power' and hard 'disk' size values shown.*
`DEPLOYED:id=1`	*The VM has been successfully deployed, and is ready to process requests.*
`ENQUEUED:VM(id=1),RQ(id=1),D(dur=-1)`	*The RPM has assigned the request to the VM.*
`SETTING_UP:VM(id=1),RQ(id=1),D(dur=1287783541/1480000000)`	*The VM is performing setup operations before processing the request (the value 'dur' is the time required to complete the setup process).*
`EXECUTING:VM(id=1),RQ(id=1),D(dur=1287783541/740000000)`	*The VM is processing the request.*

`INCOME=351213693/40000000`
`UTILITY=351213693/40000000`
	Updated values of the global total 'income' and total 'utility' variables.

`CLEANING_UP:VM(id=1),RQ(id=1),D(dur=1287783541/1480000000)`	*The VM has finished processing the request (some clean-up time is required before the VM can process another request).*
`FINISHED:VM(id=1),RQ(id=1)`	*The VM has completely finished with the request and is ready to process another.*

`VM_COST=1104831/125000`
`UTILITY=-2332227/40000000`
	Updated values of the global total 'VM cost' and total 'utility' variables.

`KILLED:id=1`	*The RPM kills the VM since it is no longer needed.*

As a slightly more complex illustration, the following console output is from an execution run with four simulated requests (line numbers have been added for reference). Points to note are:

- The RPM launches only *three* VMs (see lines 7, 11 and 19), to handle the *four* requests - and also note that the *second* VM takes longer to deploy than the *third* (lines 16 & 28);

[6] Specifically: Erlang 'emulator version' 7.3 - available from http://www.erlang.org.

- The requests are not assigned to VMs in the order in which they arrived (see lines 9, 17, 29 & 30) - namely:
  - Request 1 is first assigned to VM 1;
  - Request 3 is then assigned to VM 3;
  - Requests 4 & 2 (in that order) are then assigned to VM 2.

  In short, the RPM is both prioritising the order of execution of requests, and distributing them to different VMs for processing.
- A completion-time penalty (line 24) of around 1 unit (= 176761942682583/176000000000000) was incurred (this arises when a request is not processed within the SLA defined 'completion-time limit').
- Despite the penalty, the final total utility (line 50) is around +18.7 (= 3273851592917417/176000000000000) - i.e. the net result of the RPM's activity is that the service operates at a profit.

```
 1  START
 2  RECEIVED:RQ(id=1,size=117071231/5000000,pri=1/2)
 3  RECEIVED:RQ(id=2,size=146926711/5000000,pri=1)
 4  RECEIVED:RQ(id=3,size=14985807/2000000,pri=1)
 5  END
 6  RECEIVED:RQ(id=4,size=3798911/250000,pri=1)
 7  LAUNCHED:VM(id=1,power=74/5,disk=32)
 8  DEPLOYED:id=1
 9  ENQUEUED:VM(id=1),RQ(id=1),D(dur=-1)
10  SETTING_UP:VM(id=1),RQ(id=1),D(dur=1287783541/1480000000)
11  LAUNCHED:VM(id=3,power=44/5,disk=8)
12  EXECUTING:VM(id=1),RQ(id=1),D(dur=1287783541/740000000)
13  INCOME=351213693/40000000
14  UTILITY=351213693/40000000
15  CLEANING_UP:VM(id=1),RQ(id=1),D(dur=1287783541/1480000000)
16  DEPLOYED:id=3
17  ENQUEUED:VM(id=3),RQ(id=3),D(dur=-1)
18  SETTING_UP:VM(id=3),RQ(id=3),D(dur=14985807/32000000)
19  LAUNCHED:VM(id=2,power=72/5,disk=32)
20  EXECUTING:VM(id=3),RQ(id=3),D(dur=14985807/16000000)
21  FINISHED:VM(id=1),RQ(id=1)
22  INCOME=501071763/40000000
23  UTILITY=501071763/40000000
24  CT_PENALTIES=176761942682583/176000000000000
25  UTILITY=2027953814517417/176000000000000
26  CLEANING_UP:VM(id=3),RQ(id=3),D(dur=14985807/32000000)
27  FINISHED:VM(id=3),RQ(id=3)
28  DEPLOYED:id=2
29  ENQUEUED:VM(id=2),RQ(id=4),D(dur=-1)
30  ENQUEUED:VM(id=2),RQ(id=2),D(dur=20059265831/480000000)
31  SETTING_UP:VM(id=2),RQ(id=4),D(dur=41788021/72000000)
32  EXECUTING:VM(id=2),RQ(id=4),D(dur=41788021/36000000)
33  INCOME=804984643/40000000
34  UTILITY=3365170486517417/176000000000000
35  CLEANING_UP:VM(id=2),RQ(id=4),D(dur=41788021/72000000)
36  FINISHED:VM(id=2),RQ(id=4)
37  SETTING_UP:VM(id=2),RQ(id=2),D(dur=1616193821/1440000000)
38  EXECUTING:VM(id=2),RQ(id=2),D(dur=1616193821/720000000)
39  INCOME=1686544909/40000000
40  UTILITY=7244035656917417/176000000000000
41  CLEANING_UP:VM(id=2),RQ(id=2),D(dur=1616193821/1440000000)
42  FINISHED:VM(id=2),RQ(id=2)
43  VM_COST=1104831/125000
44  UTILITY=5688433608917417/176000000000000
45  KILLED:id=1
46  VM_COST=1523781/125000
47  UTILITY=5098552008917417/176000000000000
48  KILLED:id=3
49  VM_COST=2819733/125000
50  UTILITY=3273851592917417/176000000000000
51  KILLED:id=2
```

Unfortunately, *four* requests is the maximum we were able to simulate using the Erlang backend. At five requests or more, the execution hangs.

For the deadlock analysis (reported in §3 below) we also developed a 'skeleton' version of the model, abstracting away the details of the DGA to focus exclusively on the distributed method invocations - essentially mimicking the concurrent control flow of the full version, but without any complex data types or data processing. This reduced version of the model - listed in full in Appendix B - reliably executes up to 100 simulated requests. Intermittent problems (hanging) occur for request numbers up to 125, however, and stalling is pretty much guaranteed for more than 125 requests.

3. Deadlock Analysis

This section reports the results of applying the Envisage *deadlock analysis* tools to the ABS model for ETICS. As stated earlier (§1), the original Swift prototype of the ETICS simulator tends to stall, with output frozen and CPU usage dropping to zero. The goal of applying Envisage deadlock analysis techniques to the ABS model is just to determine whether or not the model could potentially exhibit deadlocks. Contingent on how accurately the ABS model represents the Swift code, a *negative* result (finding potential deadlocks) would:

- Point to deadlocks as a *likely* cause of stalling in the Swift prototype;
- Identify the precise points in the code where deadlock could occur.

The deadlock analysis trials were conducted on *both* local and online versions of the Collaboratory, using both SACO and DSA deadlock analysis tools, on two distinct code samples:

- The *complete* ABS model for ETICS (as listed in Appendix A);
- A skeleton version of the ABS model for ETICS (introduced in §2, listed in Appendix B).

The following subsections present the results for each of these code samples (§3.1, §3.2), together with a final summary and discussion (§3.3).

3.1. Results: Complete ABS Model

The SACO deadlock analysis tool successfully reported the model to be free of deadlocks (tool settings were left at default values[7], and the results were the same for both local and online versions of the Collaboratory) - with the following output:

```
Pointsto analysis performed in 24 ms.
LMhp analysis performed in 68 ms.
Mhp graph created in 5 ms.
Closure time 40 ms.
Discarded 0 cycles with freshness analysis.
The program is deadlock free
Complete analysis performed in 149 ms.
```

In constrast, applying the DSA deadlock analysis tool to the ABS model gave the following results:

- Using the *local* version of the Collaboratory, the tool produced no output at all.
- With the *online* version, the tool output a long list of "`Checking with: ...`" notifications, but then failed to give any final conclusion.

Accordingly, we can draw no conclusions from the DSA tool *w.r.t.* the complete ABS model. From various (unreported) tests, however, we know that it does work on smaller models - suggesting that the sheer size and complexity of the ETICS model may be a problem. To try to ease the pressure on the tool, therefore, we developed a stripped down 'skeleton' version of the ABS model - the results for which are given in the next section.

[7] Namely: Debug information = 2; Points-to analysis precision = 2; Ignore MHP information in deadlock analysis = no

3.2. Results: Skeleton ABS Model

As noted earlier, the skeleton version of the ABS model for ETICS abstracts away all the detailed data processing of the DGA to focus just on the asynchronous interactions between the distributed DGA components. The code is explained further, and listed in full in Appendix B. The deadlock analysis results for this skeleton model are essentially identical to the results for the complete model (§3.1) - namely:

- both local & online versions of the SACO tool report the skeleton code to be deadlock free;
- the local version of the DSA tool produces no output;
- the online version of the DSA tool outputs a long list of "**`Checking with: ...`**" notifications, but then fails to give any final conclusion.

The skeleton model runs to just 333 lines of code (compared to the 1600 or so lines of the complete model), and can not be simplified further without compromising the model. Through trial and error, however (i.e. by variously commenting out different lines of code) we did discover that two lines in particular were problematic for the DSA deadlock analysis tool: those indicated by **`****`** comments in the following code snippet (which shows just the initialiser & **`receive()`** methods of the class **`RPM`**):

```
class RPM() implements RPM{
    ...
    { // INIT
        solver = new Solver(this); // ****
        cloudProvider = new CloudProvider("name of a cloud provider");
    }
    ...
    Unit receive(Request request){
        println("RPM received request " + request);
        pendingRequests = appendright(pendingRequests, request);
        if (!active){
            this!activate(); // ****
        }
    }
    ...
}
```

With these two lines of code commented out, the (online) DSA tool works and successfully reports the model to be deadlock free. While these lines of code are essential to the model (since without them the RPM would never deploy any VMs), removing them never-the-less leaves the essential framework for asynchronous DGA communications intact. We can thus cautiously conclude that the DSA tool also reports the DGA to be effectively free of deadlocks.

3.3. Summary & Discussion

Table 3.3-*a* gives a summary of the results of applying the SACO and DSA deadlock analysis tools (both local & online versions) to both the *complete* and *skeleton* ABS models for ETICS.

Table 3.3-a: Summary of Deadlock Analysis Results

Model	*Collaboratory Version*	**SACO**	**DSA**
Complete Model			
	Local	***Deadlock Free ✓***	*no output*
	Online	***Deadlock Free ✓***	*no conclusion*
Skeleton Model			
	Local	***Deadlock Free ✓***	*no output*
	Online	***Deadlock Free ✓***	*no conclusion*
Skeleton Model, with problematic lines of code (see main text) commented out.			
	Local	***Deadlock Free ✓***	*no output*
	Online	***Deadlock Free ✓***	***Deadlock Free ✓***

On the whole, we are satisfied with these results:

- The SACO tool consistently reports the complete model to be deadlock free.
- The DSA tool at least reports the essential, skeleton structure of asynchronous DGA interactions to be deadlock free.

Accordingly, we feel confident in concluding that the ABS model for ETICS is in fact deadlock free. This conclusion is further corroborated by the following observations:

- It turns out that ABS models can *only* exhibit deadlock when they contain one or more **get** statements (to retrieve the value of a future) that are not immediately preceded by an **await** call - i.e. as follows (where the calling process must block until the future variable is instantiated):

  ```
  Fut<X> fx = o!foo();
  X x = fx.get;
  ```

  In contrast, the following the code *does not* block, and cannot lead to a deadlock: inserting an **await** call before the **get** suspends the current task, allowing other tasks (in the same COG) to be processed (until the future variable is instantiated):

  ```
  Fut<X> fx = o!m();
  await fx?;
  X x = fx.get;
  ```

  In ABS this latter, 'non-blocking' form of using futures can also be abbreviated as follows:

  ```
  X x = await o!m();
  ```

- The ABS model for ETICS (Appendix A) never uses the blocking form of future **get** statements: it only uses the (abbreviated) **await** version. As such, the model is *a-priori* guaranteed to be free of any deadlocks[8].

With respect to the motivating problem, therefore, we conclude that: *assuming the ABS model for ETICS accurately represents the original Swift prototype*, then deadlocks are not the reason for the Swift version stalling. The question as to just how accurately (in formal terms) the ABS model represents the Swift prototype is somewhat involved, and will not be addressed here.

4. Cost Analysis (for Scalability)

This section reports the results of applying the Envisage resource analysis tools to the ABS model for ETICS. As stated earlier (§1), the DGA has only been tested in simulation on a *serial* machine, with only small numbers of requests and VMs. The goal of resource analysis in the Engineering case study is just to assess the scalability of the DGA under under high/extreme loads.

Envisage offers two resource analysis tools, SACO and SRA, but only the SACO tool is relevant to our goal. The SACO tool, in turn, comes in two flavours - one using PUBS and the other CoFloCo as backend. The initial results (not reported here) of applying the SACO tool to the ABS model for ETICS were not good, with both tool versions variously throwing exceptions, stalling, or failing to give valid cost expressions for the majority of code samples tested. Based on this feedback, however, the developers were able to implement changes to the tool such that, with some small modifications to the ABS model, we were eventually able to get a complete set of useful results.

The following subsections first explain the rationale behind the resource analysis (§4.1 - stating which aspects of the model were analysed/measured, and why), provide some insight into the code to be tested (§4.2), and then report the final results (§4.3) with a short summary and discussion (§4.4).

[8] Recent changes to ABS semantics do introduce *implicit* blocking **get** statements under certain conditions - but these conditions do not arise in the ABS model for ETICS.

4.1. Rationale

As explained in D4.4.2, the number of different ways that the **RPM** can assign incoming requests to VMs for processing rises exponentially with the number of requests and the number of VMs. The **RPM** operates in a continuous cycle with a fixed duration, attempting - via a *distributed genetic algorithm* (DGA) - in each cycle, to find the best assignments for the requests received on the previous cycle. The longer the cycle duration, the better, at least in principle, can be the quality of the results[9] - but the harder it is to locate the best solutions due to the exponential increase in the size of the search space. The underlying goal of using Envisage resource analysis on the DGA is to help identify an optimal duration for the cycle.

In brief, we need to identify a cycle duration that will guarantee good "coverage" of the search space for high numbers of requests. We can formulate the problem more precisely as follows:

- Let $\boldsymbol{n}$ be the number of requests received by the **RPM** during a given cycle;
- Let $\boldsymbol{m} = \boldsymbol{a} + \boldsymbol{n}$ be the *maximum* number of VMs available to process these requests[10] - where $\boldsymbol{a}$ is the number of VMs available at the start of the cycle;
- Let $\boldsymbol{S}$ be the total number of *possible* solutions (assignments), which, as stated above, is some exponential function over $\boldsymbol{n}$ and $\boldsymbol{m}$. Due to stochastic factors[11] it is impossible to calculate an exact value for $\boldsymbol{S}$, but the upper bound on $\boldsymbol{S}$ is given by the following recursive function, $f(\boldsymbol{n},\boldsymbol{m})$[12]:

$$f(\boldsymbol{n},\boldsymbol{m}) = \begin{cases} 1 & \text{if } \boldsymbol{n} = 0 \text{ or } \boldsymbol{m} = 0 \\ \sum_{x=0}^{n} f(\boldsymbol{n}-\boldsymbol{x},\boldsymbol{m}-1)\boldsymbol{n}^{\underline{x}} & \text{otherwise} \end{cases} \qquad \textbf{Expr.1}$$

where $\boldsymbol{n}^{\underline{x}}$ is the $\boldsymbol{x}^{\text{th}}$ falling factorial power of $\boldsymbol{n}$ (equivalent to the Pochhammer symbol $(\mathbf{n})_x$)

  - ‣ The statistical mean for $\boldsymbol{S}$ will tend towards around half this upper-bound.
- Finally, let $\boldsymbol{s}$ be the number of solutions generated and tested by the DGA throughout the cycle;
- The "coverage" of the DGA is then the ratio, $\boldsymbol{s}/\boldsymbol{S}$, of the number of generated & tested solutions to the total number of possible solutions;

The DGA *scales* just to the extent that its coverage is both *high* and, to a reasonable approximation, *linear* with respect to $\boldsymbol{n}$ & $\boldsymbol{a}$. Accordingly, to assess the scalability, since *coverage* $= \boldsymbol{s}/\boldsymbol{S}$ and we already know $\boldsymbol{S}$ (above), we just need to find an expression for $\boldsymbol{s}$. We derive this expression as follows:

- First, recall that the DGA is implemented as a set of concurrent processes, each running a single genetic algorithm (GA) instance. In a given cycle, there are exactly $\boldsymbol{a}+1$ GA instances: one for the **RPM**, and one for each available (deployed) VM.
- Each GA instance is executed once per **RPM** cycle, and runs for the entire duration of the cycle. It sequentially creates successive *generation*s of solutions - keeping track of the best solution. At the end of the cycle, the best solution found across *all* GA instances is put into effect by the **RPM**.
- If the duration of the cycle is $\boldsymbol{T}$ seconds, and if it takes $\boldsymbol{t}(\boldsymbol{v})$ seconds for a GA instance running on a particular VM, $\boldsymbol{v}$, to create & test a *single* generation, then the number of generations created by that GA instance is $\boldsymbol{T}/\boldsymbol{t}(\boldsymbol{v})$.

[9] E.g. suppose the **RPM** receives 2 requests, *a* followed by *b*, with the optimal solution being to assign first *b* then *a* to the same VM (for serial processing). A *short* cycle duration, in which *a* and *b* are received during different iterations, would have no chance of finding this optimal solution.

[10] As detailed in D4.4.2, the **RPM** may destroy existing VMs and/or launch at most one new VM for each new request. Hence the *maximum* number of VMs available, on the *next* cycle, is the number of current VMs plus the number of requests.

[11] There are systematic preconditions on which kinds of requests can be assigned to which kinds of VMS, but since requests are generated randomly, the effect of these constraints cannot be formally accounted for (see D4.4.2 for details).

[12] In brief: the recursive iteration covers each of the different ways to distribute $\boldsymbol{x}$ ($0 \le \boldsymbol{x} \le \boldsymbol{n}$) requests among $\boldsymbol{m}$ machines. For each possibility we need also consider the different ways in which $\boldsymbol{x}$ requests can be selected from $\boldsymbol{n}$ requests ($\boldsymbol{n}!/(\boldsymbol{n}-\boldsymbol{x})\boldsymbol{x}!$) and their different possible orderings ($\boldsymbol{x}!$). The product of these factors is $\boldsymbol{x}!\boldsymbol{n}!/(\boldsymbol{n}-\boldsymbol{x})\boldsymbol{x}! = \boldsymbol{n}!/(\boldsymbol{n}-\boldsymbol{x}) = \boldsymbol{n}^{\underline{x}}$.

- The *total* number of generations created per cycle for the *whole* DGA is therefore:

$$\sum_{v \in V} T/t(v)$$

where V is the set of all machines running GA instances (i.e. $|V| = a + 1$)

- The number of individual solutions created in *each* generation is fixed, across all GA instances, at some constant p.
- Ostensibly, therefore, the total number, s, of solutions tested per cycle is $p\sum_{v \in V} T/t(v)$. There is again, however, a stochastic element to creating solutions which will result in some number of duplicates - hence the best we can do is define the upper-bound on s:

$$UB(s) = p\sum_{v \in V} T/t(v) \qquad \textbf{\textit{Expr.2}}$$

- Now since p, V and T are all given quantities, the only unknown in ***Expr.2*** is the value $t(v)$ - *the time it takes for a GA instance running on VM* v *to create & test one generation*. If the number of computational steps required for this process is g, and the time it takes v to perform a single computational step is $o(v)$, then we have $t(v) = o(v)g$. The value $o(v)$ is again a pre-given, so it just remains to determine g, and for this we can use the Envisage resource analysis tools. The specific ABS code to be analysed is described in the next section.

4.2. The Relevant DGA Code

As explained in the previous section, the goal of applying the Envisage resource analysis tools to the ABS model for ETICS is to determine the number, g, of computational steps required by the DGA to create and test a single generation of solutions. Henceforth, we will just use the term 'cost' rather than 'the number of computational steps'. In the ABS model, each generation is created and tested by the method **`createNextGeneration(..)`** of the class **`Solver`**. So the goal is to determine the cost of this method - the complete implementation of which is as follows (from Appendix A):

```
List<Solution> createNextGeneration(List<Solution> previous, Int pop_count, Map<VMId, VM> pool){
      List<VM> vms = values(pool);
      Int vm_count = length( vms );
       Int top_count = ceiling(pop_count / 20);
      List<Solution> next_generation = Nil;
      Int i = 0;
      while (!cancelled && i < pop_count){
              Solution solution = nth(previous, random(top_count));
              Rat f = randomf();
              if (f < 1/4 || vm_count == 0){
                      solution = this.mutate(solution); // **** MUTATE ****
              }else if (f < 1/2){
                      Solution another = nth(previous, random(top_count));
                      solution = this.crossover(solution, another); // **** CROSSOVER ****
              }else if (f < 3/4){
                      // note: vm_count > 0 (see 1st case above)
                      VM vm = nth(vms, random(vm_count));
                      if (vm != null){
                            solution = await vm!bestSolution(); // **** VM - BEST SOLUTION ****
                      }
              } // else{ **** LEAVE THE SOLUTION UNCHANGED **** }
              next_generation = Cons(solution, next_generation);
              i = i + 1;
      }
      return next_generation;
}
```

In brief, the method **`createNextGeneration(..)`** takes three input parameters:

- A list, **`previous`**, of solutions from the previous generation;
- The number, **`pop_count`**, of solutions per generation (equal to the length of **`previous`**);
- A dictionary listing all currently deployed VMs (indexed by their IDs).

The method creates a list of **`pop_count`** new solutions - each of which, α, is constructed as follows:

- First, an existing solution, call it β, is chosen at random from the **`previous`** list;
- Then, α is randomly (with equal probability) assigned to one of the following values (indicated in the code snippet above by the `****` comments):
  - a mutated version of β (produced by calling **`this.mutate(..)`**);
  - the result of applying the *genetic crossover function* to β and a second solution also chosen at random from the **`previous`** list (the call to **`this.crossover(..)`**);
  - the best solution found thus far by a randomly chosen GA instance, which is accessed through the VM that hosts the GA (the asynchronous call to **`vm!bestSolution(..)`**);
  - β, i.e. the previous solution is just carried over, without modification, to the new generation.

With respect to resource analysis, there are four significant points to note:

- The cost of the **`createNextGeneration`** method is partially determined by the cost of *all* the methods that it invokes - i.e. **`mutate`**, **`crossover`** and **`bestSolution`** - as well as any further methods invoked by these, and so on recursively. The complete method call hierarchy for **`createNextGeneration`** is shown in Fig. 4.2-*a*.

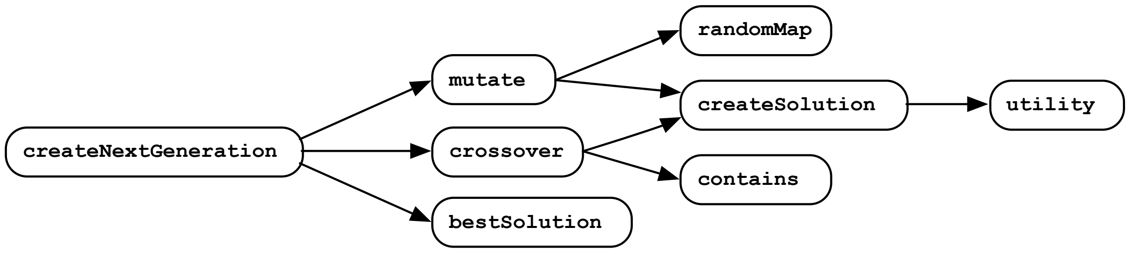

Figure 4.2-a: Call hierarchy for the **`createNextGeneration`** *method*

- For any given invocation of **`createNextGeneration`**, however, the *precise* number of calls to external methods (**`mutate`**, **`crossover`** and **`bestSolutions`**) is *a priori* unpredictable - since the calls are made *at random*. To overcome this, we could either:
  - Take a statistical approach - e.g. from the code, it is clear that the **`mutate`**, **`crossover`** and **`bestSolution`** methods are each called, *on average*, one quarter of the time;
  - Calculate an upper-bound on the cost, based on the worst case scenario - e.g. if the cost for the **`mutate`** method is greater than that of either **`crossover`** or **`bestSolution`**, then we assume that **`mutate`** is always called.
- Moreover, several of the methods in the call hierarchy (Fig. **4.2-*a***) have similarly stochastic, and problematic, computational features.
- Finally, and perhaps most problematically, the call to **`vm!bestSolution(..)`** is *asynchronous*. So even if the cost of the **`bestSolution`** method is known, this cost may bear little relation to the time required to retrieve the best solution - which would depend on communications latency, and on what other tasks the remote **`vm`** is busy with when it receives the request.
  - In short: this invalidates the assumption, from §4.1, that $t(v) = o(v)g$. There are ways & means to mitigate this issue, but we will not discuss them here.

In summary, for all the reasons noted above, we believe that the **`createNextGeneration`** method constitutes an *extreme* test-case for the resource analysis tools - the results of which are presented in the next section.

4.3. Results: SACO Resource Analysis

As stated earlier, with a few modifications to the ABS model, we were able to retrieve a complete set of results from the SACO resource analysis tool. The code modifications are detailed in Appendix C.1, and the full list of all results obtained is given in Appendix C.2. Here we report just the results of applying the tool to calculate the cost of the **createNextGeneration** method. Settings for the tool were left at default values except for the following:

- *Size abstraction for terms* = **TypedNorms**
- *backend* = **CoFloCo**

The result is shown in Table 4.3-*a*.

Table 4.3-a: Results for SACO Resource Analysis of the* createNextGeneration *method
where: $nat(x) = x$ if $x \geq 0$, and $nat(x) = 0$ if $x < 0$

Method Signature	`Solver.createNextGeneration(`
	`    List<Solution> previous,`
	`    Int pop_count,`
	`    Map<VMId, VM> pool`
	`)`

Cost Expression (CE)	`UB(A,B,C,D,E,F) = max([16*F,nat(A)*28800*nat(D)*nat(E)`
	`+nat(A)*58848*nat(E)+nat(B)*21120*nat(D)*nat(E)+nat(B)*47616*nat(E)`
	`+nat(C)*1026*nat(E)+nat(D)*100160*nat(D)*nat(E)+nat(D)*723360*nat(E)`
	`+nat(E)*1988474+nat(nat(A)+ -2)*1176*nat(E)+nat(nat(A)+`
	`-1)*21600*nat(D)*nat(E)+nat(nat(A)+ -1)*46128*nat(E)+nat(A`
	`+D)*1792064*nat(E)+nat(A+D)*869504*nat(A+D)*nat(E)`
	`+max([nat(D)*128256*nat(A+D)*nat(E)+nat(nat(A)+ -2)*1176*nat(E)+nat(A`
	`+D)*1515712*nat(E)+nat(A+D)*1143936*nat(A`
	`+D)*nat(E),nat(A)*57600*nat(D)*nat(E)+nat(A)*117456*nat(E)`
	`+nat(B)*42240*nat(D)*nat(E)+nat(B)*86016*nat(E)+nat(C)*3222*nat(E)`
	`+nat(D)*214640*nat(D)*nat(E)+nat(D)*1261024*nat(E)+nat(D)*628992*nat(A`
	`+2*D)*nat(E)+nat(E)*4692040+48*F+1260*F*nat(E)+nat(nat(A)+`
	`-1)*43200*nat(D)*nat(E)+nat(nat(A)+ -1)*86232*nat(E)+nat(A`
	`+2*D)*9021920*nat(E)+nat(A+2*D)*4932928*nat(A+2*D)*nat(E)])])+25`

Parameters Used in CE	`A = size of previous wrt. List<VMInfo>`
	`B = size of previous wrt. Map<RequestId, Task>`
	`C = size of previous wrt. List<Solution>`
	`D = size of previous wrt. List<Pair<RequestId, Rat>>`
	`E = size of pop_count wrt. Rat`
	`F = size of pool wrt. Map<VMId, VM>`

Complexity of Method	`O(n^3)`
Time To Calculate CE	`893510 ms (≃ 15 mins)`

It is straightforward to convert the cost expression in Table 4.3-*a* into executable code (an example is given in Appendix C.3). Doing so, and plugging in unit parameters, gives the following result:

```
UB(1,1,1,1,1,1) = 89856207
```

If we suppose, *very* roughly, that a 2GHz CPU can perform 10^9 machine instructions per second, then the result indicates that it would take at least $^1/_{10}$ of a second to construct a single, *minimal* solution[13]. The original Swift prototype of the RPM generates and tests around 40,000 (*non-minimal*) solutions per second, or 1 solution every $^1/_{40,000}$ of a second - which suggests that only around 25,000 instructions[14] are required per solution. These are extremely rough estimates, but the cost expression calculated by the SACO tool does seem to miss the mark by some orders of magnitude.

[13] i.e. to create a generation containing 1 solution (parameter **E** = 1) which at most assigns 1 request (**B** = 1) to 1 VM (**F** = 1).

[14] 10^9 / 40,000 = 25,000

4.4. Summary & Discussion

Overall the results from the SACO (CoFloCo) resource analysis tool are mixed. The tool *did* succeed in generating a valid cost expression for the target method. However:

- this required modifications to the original code (Appendix C.1) - which is to say that there are restrictions on the tool's use: it does not yet support the full range of valid ABS expressions;
- the resulting cost expression seems to over-estimate, by some orders of magnitude, the actual cost of the method.

With respect to the motivating objective, assessing the scalability of the DGA, we can never-the-less apply these results to get get an idea of the coverage of the algorithm (as described in §4.1). For present purposes, a single example will suffice ('UB' = upper-bound):

- Let $\boldsymbol{p} = 100$ (the number of solutions per generation);
- Let $\boldsymbol{n} = 5$ (the number of requests received in a given cycle);
- Let $\boldsymbol{a} = 5$ (the number of VMs available to run GA instances);
- So:
  - $\boldsymbol{m} = \boldsymbol{a} + \boldsymbol{n} = 10$ (the number of different VMs to which the requests could be assigned);
    - For simplicity, we will assume that all these VMs have the same computing power;
  - $\mathit{UB}(\boldsymbol{S}) = 340391$ (the UB on the number of different possible assignments - from ***Expr.1*** in §4.1)
- Given $\boldsymbol{p}$, $\boldsymbol{n}$ & $\boldsymbol{m}$, and the cost expression generated by the SACO tool (Table 4.3.2-*a*), we get:
  - $\boldsymbol{g} = \mathit{UB}_{createNextGeneration}(1,5,1,1,100,10) = 9065544705$ (the computational cost of creating and testing a single generation containing $\boldsymbol{p}$ solutions);
- Assuming that a 2GHz CPU performs 10^9 machine instructions per second, we also have:
  - $\boldsymbol{o}(v) = 1/10^9$ seconds (the time taken to perform a single operation)
- Hence:
  - $\boldsymbol{t}(v) = \boldsymbol{o}(v)\boldsymbol{g} = 9065544705/10^9 \simeq 9$ seconds (the time required to create & test a generation);
- Now, arbitrarily, let $\boldsymbol{T} = 60$ seconds (the duration of a single RPM cycle), hence:
  - $\boldsymbol{T}/\boldsymbol{t}(v) = 60/9 \simeq 6.7$ (the number of generations created & tested by a GA instance);
- Which allows us to calculate the upper-bound on $\boldsymbol{s}$, the number of solutions created & tested by the distributed DGA (comprising *all* the GA instances) during one cycle:
  - The number of GA instances is $\boldsymbol{a} + 1 = 6$ (one for each available VM, and one for the RPM), so from ***Expr.2*** (§4.1) we have:
    - $\mathit{UB}(\boldsymbol{s}) = \boldsymbol{p}\sum_{v \in V} \boldsymbol{T}/\boldsymbol{t}(v) = 100 \times 6 \times 6.7 \simeq 4020$
- The coverage in this case (the ratio of the solutions tested, $\boldsymbol{s}$, to all possible solutions, $\boldsymbol{S}$) is thus:
  - $\boldsymbol{s}/\boldsymbol{S}\ (\simeq \mathit{UB}(\boldsymbol{s})/\mathit{UB}(\boldsymbol{S})) = 4020/340391 \simeq 0.012 = \mathbf{1.2\%}$

This means, in summary, that 6 machines, each working continuously for 1 minute, will only manage to generate and test around $^1/_{100}$ of all the possible assignments of *only* 5 requests to 10 VMs. Under these conditions we would be hoping instead for 80% coverage or higher, so this is a very poor result for the DGA. The results, moreover, will be even worse for larger numbers of requests and VMs. So based on these results, we must conclude that the DGA does not scale.

As noted in the previous section, the SACO results are inconsistent with the original Swift prototype for the RPM, which achieved performance levels (measured with standard code profiling techniques) of at least 3 orders of magnitude greater. In mitigation, we did state earlier (§4.2) that we consider the **`createNextGeneration`** method to constitute an *extreme* test-case for the resource analysis tools. The mere fact that the tool produced a valid cost expression is thus worthy of note.

5. Java Code Generation

This section reports the results of applying the Envisage Java code generation tools to the ABS model for ETICS. Engineering's objective here, as noted earlier (§1), is to provide a *proof of concept* assessment of the feasibility of adopting ABS in the production lifecycle for Java-based services (such as ETICS). Put simply we wish to ascertain:

- If executable Java code can in fact be automatically generated from the ABS model - and if so:
  - Whether the code is functionally valid (does it do what it is intended to do);
  - Whether the code performs well in terms of speed and memory usage.

As noted at the outset (§1.2) the code generation tool is not as yet available as part of the ABS tool chain. Recently, however, we have been given access to the source files for the tool, have successfully compiled them, and used the tool to generate and run Java code from a few small ABS model test cases. An illustration is provided in Appendix D - which presents an ABS model encapsulating the formula in ***Expr.1*** (§4.1). The generated Java code for this model is listed in full in Appendix D.2. The generated code *as is* does not compile, but the errors are minor and easy to rectify, and once fixed, the code performs exactly as expected. It is also significant that the generated Java code has essentially the same (class/interface/method) structure as the source ABS model - such that the relation between the two is clear and predictable.

When applied to the complete ABS model for ETICS (Appendix A), however, the tool reports that the model is *syntactically* invalid (we know from the Erlang simulator runs in §2 that this is not the case). It turns out that the tool is based on a version of the ABS syntax that is significantly older than the version used by the rest of the ABS tool chain. In particular, this version of the syntax does not support the following features - all of which are present in the ETICS ABS model:

- Use of the '!' symbol for logical negation (the older syntax used the tilde '~');
- Nested function calls - e.g. of the form **`foo(goo())`** (see also our previously reported comments *w.r.t.* syntactic issues in §3.1 of D4.4.1);
- Use of '//' style comments;
- Use of the built-in object method **`thisDC()`** to retrieve the deployment component on which the object is running/hosted (note that this is the same 'critical language feature', reported above in §1.1, that had previously prevented us executing the ABS model for ETICS);

The first three features are trivial to resolve by modifying the ABS model, but we could not find any alternative for the **`thisDC()`** method - which is required for the RPM to decommission VMs - and so we are currently on hold waiting for the tool developers to implement **`thisDC()`** support.

In conclusion, although the Java code generation tool works, it employs an out-of-date version of the ABS syntax, and hence cannot translate our ABS model, which can only be expressed in the current ABS syntax.

6. Overall Summary & Conclusion

In general terms, it should be underlined that *all* of the Envisage tools do in fact work (as can be verified in the online Collaboratory using the pre-installed examples[15]). For the *particular* case of the ABS model for ETICS, however, the results are mixed. Two of the six tools that we used worked without critical issue - namely: SACO deadlock analysis and SACO resource analysis using the CoFloCo backend. The Erlang simulator and DSA deadlock analysis can be considered as partial successes, but we were unable to satisfactorily apply the other two tools: SACO resource analysis with the PUBS backend and Java code generation.

More specifically, as reported in this deliverable, our experience with the Envisage tools highlighted the following issues:

- The Erlang simulator successfully compiles and executes the ABS model for ETICS (as presented in Appendix A, without any modifications) - *but*:
  - The tool only works for small (≤4) numbers of requests, and stalls for higher numbers.
- The SACO deadlock analysis tool consistently reported (all versions of) the ETICS ABS model to be free of deadlocks - *but*:
  - The DSA deadlock analysis tool did not work at all on the complete model, and only worked on the skeleton ABS model after problematic lines of code had been commented out;
  - In any case, the ABS model is *a priori* guaranteed to be free of deadlocks, for the simple reason that it does not contain any blocking **`get`** calls;
- The SACO resource analysis tool successfully produced valid cost expressions for the target **`createNextGeneration`** method (as well as for all methods in the target's call hierarchy) - *but*:
  - It only produced these results following (minor) modifications to the model;
  - The resulting cost expression (we believe) over-estimates the cost by orders of magnitude;
- The Java code generation tool is not part of the ABS tool chain and uses an older version of the ABS syntax. The tool works for simple test cases - *but*:
  - Since the ETICS model is not compatible with the older syntax, the tool rejects the model, and is unable to translate it into Java.

Mitigating factors for these results may include the sheer size and complexity of the ABS model for ETICS, which contains several *deeply* recursive object methods (perhaps the root cause of the Erlang problems), and employs stochastic control flow (which cannot be trivial for formal resource analysis). In light of this complexity, the results look more promising, and we hope that the tool developers rise to the challenge of this case study by further refining and improving their algorithms.

With respect to the case study objectives, we draw the following conclusions:

1. **To ensure (if possible) that the DGA is deadlock free:**

   As reported in §3, we feel confident that the ABS model for ETICS is deadlock free. But we are less confident that this also holds for the original Swift version. Specifically: it remains possible that the Swift version does in fact hang because of deadlocks, but that (due to errors/over-sights in porting Swift to ABS) the critical features of the Swift version leading to deadlock are simply not represented in the ABS version. To ensure against this kind of problem, we need systematic rules for translating existing code into ABS (e.g. based on a comparison of their underlying concurrency models) - which this project has not provided.

   Conversely, since we do now have a version of the DGA, in ABS, that is demonstrably deadlock free, it would be useful to either execute this version directly (*cf.* the Erlang results), or transform it into executable code that is also verifiably deadlock free (*cf.* the code generation results) - neither of which we were able to satisfactorily achieve.

[15] Except for Java code generation which we verified separately.

2. **To ensure that the DGA scales:**

   The basic workflow involved in using the Envisage tools to determine the scalability of the DGA was as follows:

   - create an ABS model of the DGA;
   - apply the resource analysis tools to specific methods in the model;
   - convert the resulting cost expressions into executables, and use these to calculate cost values under a range of input contingencies;
   - do some math with the cost values to estimate the *rate* at which solutions are created and tested - and from this assess the scalability of the DGA.

   There is obviously a lot of work involved in this process. The current practice in Engineering to address issues of resource usage is to employ standard off-the-shelf code profiling tools - but there are doubtless cases where the Envisage approach would be preferable. General guidelines to support the decision about when to use one approach or the other would greatly improve the acceptability of the Envisage methodology within an industrial setting.

3. **Automatic generation of Java code:**

   Engineering's software services are developed in Java. While the ability to build detailed ABS models with formally demonstrable properties (freedom from deadlocks, peak resource costs, etc.) has definite value - this value would be diminished for Engineering if it were to prove necessary to develop and maintain *distinct*, *hand-crafted* ABS and Java versions of the same system. Automating the generation of Java from ABS is therefore essential if Engineering is to adopt the Envisage methodologies. Despite the fact that we were unable to generate Java code for our ABS model, we never-the-less believe this feature to be critical.

Finally, in terms of technological readiness, our experiences throughout the project suggest that all the tools developed by Envisage would greatly benefit from a stronger software engineering practice. More controlled / regulated processes governing the documentation, packaging, release & deployment of the toolset would lead the entire ABS toolchain to higher levels of readiness for a typical industrial setting. In terms of Technology Readiness Level[16] (TRL), we would rank the Envisage tools as ranging between TRL3 and TRL4.

[16] e.g. see: https://en.wikipedia.org/wiki/Technology_readiness_level

Appendices

A. Complete ABS Model for the ETICS Case Study

This appendix lists the *complete* code for the ABS model for Engineering's ETICS case study. This code is a revised version of the code submitted with Deliverable D4.4.2 - in particular:

- All the code has been condensed into a single file, for use in the Collaboratory[17];
- Functional types involving 'case' statements have been replaced with simple constants - since 'case' statements are not supported by the Java code generation tools;
- Various changes were made to keep in line with changes to the ABS core syntax and library;
- Various bugs have been fixed.

The complete code runs to 1672 lines, and is structured into the following sections (the architecture of the model has already been described in D4.4.2):

- Functions
- Global constants & aliases
- Functional data types:
  `ServiceLevel`
  `SLA`
  `RequestPriority`
  `VMData`
  `Request`
  `VMInfo`
  `Task`
  `Problem`
  `Solution`
- Interfaces & classes:
  `Solver`
  `Processor`
  `VM (extends Solver)`
  `RPM`
  `Simulator`
  `Tally`
- The **`main`** method

The ABS model is as follows:

```
//=======================================================================
// ETICS.abs
// THIS IS THE UPDATED (2016) VERSION OF THE ENG "ETICS" MODEL
//=======================================================================

module ETICS;

import * from ABS.DC;

//=======================================================================
// FUNCTIONS
//=======================================================================

def Rat randomf() = random(10000000)/10000000;
def Int ceiling(Rat r) = truncate(r) + 1;

def Int next_power_of_2(Rat r, Int current) =
  if r < current then current else next_power_of_2(r, current * 2);
def Rat sqrt(Rat r) = sqrt_newton(r, r, 1/100);
```

[17] The Collaboratory is unable to save/open files. Instead, a new empty file has to be created, into which the code is then pasted. This is easier if all the code comes from a single source file.

```
//=========================================================================
// CONSTANTS
//=========================================================================

def Bool global_logging() = True; // turn on/off the Tally 'println' logging
def Int global_request_count() = 1000; // total number of requests to generate
def Int global_population_count() = 50; // total number of solutions in a single GA population

def Rat global_xP() = 1; // unit cost per size for requests
def Rat global_kCT() = 1; // completion time constant
def Rat global_xCT() = 1; // unit completion time penalty
def Rat global_xFR() = 1; // unit failure rate penalty
def Rat global_N() = 50; // failure rate modulo
def Rat global_du() = 60; // VM time unit (in minutes)
def Rat global_dAT() = 1; // action time

def Rat global_min_vm_power() = 16/10;
def Rat global_max_vm_power() = 168/10;

def Rat global_min_request_size() = 1;
def Rat global_max_request_size() = 32;

//=====================================
// Aliases
//=====================================

data Price = Price(Rat priceValue);

//=========================================================================
// FUNC TYPES
//=========================================================================

//=========================================================================
// ServiceLevels

def Rat bronzeSL() = 1/2;
def Rat silverSL() = 3/4;
def Rat goldSL() = 1;

data ServiceLevel = ServiceLevel(Rat serviceLevelValue);

//=========================================================================
// SLA

data SLA = SLA(   Int slaId,
                  Int slaTimeZone,
                  Int slaUserCount,
                  ServiceLevel slaServiceLevel);

def Rat maxFR(SLA sla)
      = ceiling(2/serviceLevelValue(slaServiceLevel(sla)));

//=========================================================================
// RequestPriority

def Rat adHocPriority() = 1;
def Rat scheduledPriority() = 1/2;

data RequestPriority = RequestPriority(Rat requestPriorityValue);

//=========================================================================
// VMData

data VMId = VMId(Int vmIdValue);
data VMData = VMData(VMId vmId, Rat vmClock, Int vmCores, Int vmMemory, Int vmDisk);

def Rat vmPower(VMData d)
      = vmPower3(vmClock(d), vmCores(d), vmMemory(d));

def Rat vmPower3(Rat clk, Int cor, Int mem)
      = (clk * cor) + (6 * mem / 10);

def Rat vmSpec(VMData d)
      = vmPower(d) + vmDisk(d);

def Duration vmDeployTime(VMData d)
      = Duration((2/10) * vmSpec(d));

def Price vmUnitCost(VMData d)
      = Price((266/100000) * vmSpec(d));
```

```
//============================================================================
// Request

data RequestId = RequestId(Int requestIdValue);
data Request = NoRequest | Request( RequestId requestId,
                                    Rat requestSize,
                                    Int requestUser,
                                    RequestPriority requestPriority,
                                    Time requestReceiveTime,
                                    SLA requestSLA,
                                    VMData requestDefaultVMData);

def Rat requestPhi(Request r)
      = rawRequestPhi(requestSLA(r), requestPriority(r));

def Rat rawRequestPhi(SLA sla, RequestPriority p)
      = serviceLevelValue(slaServiceLevel(sla)) * requestPriorityValue(p);

def Duration requestMaxCT(Request r)
      = rawRequestMaxCT(requestSize(r), requestPhi(r));

def Duration rawRequestMaxCT(Rat size, Rat phi)
      = Duration((global_kCT() * size) / phi);

def Price requestPrice(Request r)
      = Price(global_xP() * requestSize(r) * requestPhi(r));

def Price requestPenaltyFR(Request r)
      = Price(global_xFR() * serviceLevelValue(slaServiceLevel(requestSLA(r)))
                                         * serviceLevelValue(slaServiceLevel(requestSLA(r))));


//============================================================================
// VMInfo

data VMInfo = VMInfo(   VMData vmInfoVMData,
                        Time vmInfoActionTime,
                        Time vmInfoStartTime,
                        Time vmInfoLaunchTime,
                        Rat vmScore,
                        Bool vmInfoFirstUse);



//============================================================================
// Task

data Task = Task(Request taskRequest, List<VMInfo> taskVMInfoList);



//============================================================================
// Problem

data Problem = Problem( Map<RequestId, Task> problemTaskMap,
                        Map<VMId, VMInfo> problemVMInfoMap);



//============================================================================
// Solution

data Solution = NoSolution | Solution(Problem solutionProblem,
                                       List<Request> solutionRejections,
                                       Map<VMId, Pair<VMInfo, List<Request>>> solutionAssignments,
                                       Price solutionUtility,
                                       List<Pair<RequestId,Int>> solutionMaps);




//============================================================================
// SOLVER
//============================================================================

interface Solver {
    Unit startSolving(Problem problem, Map<VMId, VM> pool);
    Solution stopSolving();
    Solution bestSolution();
}
```

```
class Solver(
      Rat vmPower
) implements Solver{

      //=====================================
      // Properties
      //=====================================

      Bool cancelled = False;
      Solution best = NoSolution;
      List<Solution> solutions = Nil;

      //=====================================
      // Interface Methods
      //=====================================

      Solution stopSolving(){
            cancelled = True;
            return best;
      }

      Solution bestSolution(){
            return best;
      }

    Unit startSolving(Problem problem, Map<VMId, VM> pool){
            cancelled = False;
            best = NoSolution;
            solutions = Nil;

            // generate initial random solutions ..

            Map<RequestId, Task> task_map = problemTaskMap(problem);
            Int n = global_population_count(); //ceiling(vmPower) * size(keys(task_map));
            Int i = 0;
            while (!cancelled && i < n){
                  Solution solution = this.randomSolution(problem);
                  solutions = Cons(solution, solutions);
            i = i + 1;
            }

            // iterate through successive generations ..

            while (!cancelled){
                  solutions = this.qsortByDescendingUtility(solutions);
                  this.updateBestSolution( nth( solutions, 0 ) );
            solutions = this.createNextGeneration(solutions, n, pool);
                  await duration(1, 1);
            }

      }

      List<Solution> createNextGeneration(List<Solution> previous, Int pop_count, Map<VMId, VM>
pool){
            List<VM> vms = values(pool);
            Int vm_count = length( vms );
            Int top_count = ceiling(pop_count / 20);
            List<Solution> next_generation = Nil;
            Int i = 0;
            while (!cancelled && i < pop_count){
                  Solution solution = nth(previous, random(top_count));
                  Rat f = randomf();
                  if (f < 1/4 || vm_count == 0){
                        solution = this.mutate(solution);
                  }else if (f < 1/2){
                        Solution another = nth(previous, random(top_count));
                        solution = this.crossover(solution, another);
                  }else if (f < 3/4){
                  // note: vm_count > 0 (see 1st case above)
                        VM vm = nth(vms, random(vm_count));
                        if (vm != null){
                              solution = await vm!bestSolution();
                 }
                  }// just use 'solution' again
                  next_generation = Cons(solution, next_generation);
                  i = i + 1;
            }
            return next_generation;
    }
```

```
//====================================
// Private
//====================================

//======== Quicksort Solution list by descending utility ========

List<Solution> qsortByDescendingUtility(List<Solution> list){
      List<Solution> res = Nil;
      Solution head = head(list);
      Pair<List<Solution>, List<Solution>> split = this.qsplit( head, tail(list) );
  List<Solution> sorted_small = fst(split);
      if (sorted_small != Nil){
      sorted_small = this.qsortByDescendingUtility( sorted_small );
  }
      List<Solution> sorted_big = snd(split);
      if (sorted_big != Nil){
      sorted_big = this.qsortByDescendingUtility( sorted_big );
  }
      if (sorted_small != Nil){
      if (sorted_big != Nil){
            res = concatenate( sorted_small, Cons( head, sorted_big ) );
      }else{
            res = appendright( sorted_small, head );
      }
  }else if (sorted_big != Nil){
      res = Cons( head, sorted_big);
  }else{
      res = list[ head ];
  }
  return res;
}

Pair<List<Solution>, List<Solution>> qsplit(Solution x, List<Solution> list){
      Rat x_util = priceValue( solutionUtility( x ) );
      List<Solution> smaller = Nil;
      List<Solution> bigger = Nil;
      while(list != Nil){
            Solution h = head(list);
      Rat h_util = priceValue( solutionUtility( h ) );
            if (h_util < x_util ){ // note '<' = DESCENDING
                  bigger = Cons(h, bigger);
            }else{
                  smaller = Cons(h, smaller);
            }
            list = tail(list);
      }
      return Pair(smaller, bigger);
}

//====================================

Unit updateBestSolution(Solution solution){
      if (!cancelled && solution != NoSolution){
            if (best != NoSolution){
                  Rat best_u = priceValue(solutionUtility(best));
                  Rat u = priceValue(solutionUtility(solution));
                  if (u > best_u){
                        best = solution;
                  }
            }else{
                  best = solution;
            }
      }
}

Solution randomSolution(Problem problem){
      Map<RequestId, Task> task_map = problemTaskMap(problem);
      List<Task> task_list = values(task_map);
      List<Pair<RequestId,Int>> maps = Nil;
      while (task_list != Nil){
            Task task = head(task_list);
            Pair<RequestId,Int> map = this.randomMap(task);
            maps = appendright(maps, map);
            task_list = tail(task_list);
      }
      maps = this.randomiseOrder(maps);
      return this.createSolution(problem, maps);
}
```

```
  Pair<RequestId,Int> randomMap(Task task){
        Request request = taskRequest(task);
        RequestId request_uuid = requestId(request);
        List<VMInfo> vm_info_list = taskVMInfoList(task);
        Int n = length(vm_info_list);
        Int i = 0;
        Int selected = -1;
        while (selected < 0 && i < n){
              if (randomf() > (1/4)){
                    selected = i;
              }
              i = i + 1;
        }
        if (selected < 0){
              selected = random(n + 1); // extra '1' is for rejections
        }
    return Pair(request_uuid, selected);
}

Solution createSolution(Problem problem, List<Pair<RequestId,Int>> maps){
    Map<RequestId, Task> task_map = problemTaskMap(problem);
        List<Request> rejected = Nil;
        Map<VMId, Pair<VMInfo, List<Request>>> assigned = EmptyMap;
        Rat utility = 0;
        Int n = length(maps);
        Int i = 0;
        while (i < n){
        Pair<RequestId,Int> map = nth(maps, i);
              RequestId request_id = fst(map);
              Int vmi_index = snd(map);
              Maybe<Task> maybe_task = lookup(task_map, request_id);
              if (maybe_task != Nothing){
                    Task task = fromJust(maybe_task);
                    Request request = taskRequest(task);
                    List<VMInfo> vmis = taskVMInfoList(task);
                    if (vmi_index >= length(vmis)){
                      rejected = Cons(request, rejected);
                      Rat penalty_fr = priceValue(requestPenaltyFR(request));
                      utility = utility - penalty_fr;
                    }else{
                      VMInfo vm_info = nth(vmis, vmi_index);
                      VMId vm_id = vmId(vmInfoVMData(vm_info));
                      Maybe<Pair<VMInfo, List<Request>>> maybe_assigned
                                                        = lookup(assigned, vm_id);
                      List<Request> requests = Nil;
                      if (maybe_assigned != Nothing){
                        Pair<VMInfo, List<Request>> p = fromJust(maybe_assigned);
                        requests = snd(p);
                      }
                      requests = Cons(request, requests);
                      assigned = put(assigned, vm_id, Pair(vm_info, requests));
                    }
              }
              i = i + 1;
        }
        Set<VMId> vm_ids = keys(assigned);
        while (hasNext(vm_ids)){
        VMId vm_id = take(vm_ids);
              Maybe<Pair<VMInfo, List<Request>>> maybe_assigned = lookup(assigned, vm_id);
              if (maybe_assigned != Nothing){
                    Pair<VMInfo, List<Request>> p = fromJust(maybe_assigned);
                    VMInfo vm_info = fst(p);
                    List<Request> requests = snd(p);
                    Triple<Rat, List<Request>, List<Request>> t
                                                  = this.utility(vm_info, requests);
                    utility = utility + fstT(t);
                    List<Request> accepted = sndT(t);
                    List<Request> rejects = trd(t);
                    if (length(rejects) > 0){
                          assigned = put(assigned, vm_id, Pair(vm_info, accepted));
                          while (rejects != Nil){
                                rejected = Cons(head(rejects), rejected);
                                rejects = tail(rejects);
                          }
                    }
              }
        vm_ids = remove(vm_ids, vm_id);
        }
        return Solution(problem, rejected, assigned, Price(utility), maps);
  }
```

```
List<Pair<RequestId,Int>> randomiseOrder(List<Pair<RequestId,Int>> maps){
  List<Pair<RequestId,Int>> input = maps;
      List<Pair<RequestId,Int>> output = Nil;
      while (input != Nil){
      Int n = length(input);
            Int i = random(n);
            Pair<RequestId,Int> map = nth(input, i);
            input = without(input, map);
            output = Cons(map, output);
      }
      return output;
}

Triple<Rat, List<Request>, List<Request>> utility(VMInfo vm_info, List<Request> requests){
      VMData vmData = vmInfoVMData(vm_info);
      Rat utility = 0;
      List<Request> accepted = Nil;
      List<Request> rejected = Nil;
      Rat tACT = timeValue(vmInfoActionTime(vm_info));
      Rat tSTART = timeValue(vmInfoStartTime(vm_info));
      Rat tLAUNCH = timeValue(vmInfoLaunchTime(vm_info));
      Bool used = !vmInfoFirstUse(vm_info);
      List<Request> list = requests;
      while (list != Nil){
            Request request = head(list);
            Rat tRCV = timeValue(requestReceiveTime(request));
            Rat size = requestSize(request);
            Rat pow = vmPower(vmData);
            Rat dXT = ((11/10) * size) / pow;
            Rat tEND = tSTART + (2 * dXT);
            Rat dAT = tACT - tRCV;
            Rat dQT = tSTART - tACT;
            Rat dXT_1_5 = ((3/2) * dXT);

            // price for using VM ..

            Rat uc = priceValue(vmUnitCost(vmData));
            Rat price_at_start = 0;
            if (used){
                  price_at_start = uc * ceiling((tSTART - tLAUNCH) / global_du());
            }
            Rat price_at_end = uc * ceiling((tEND - tLAUNCH) / global_du());
            Rat vm_price = price_at_end - price_at_start;

            // completion-time penalty ..

            Rat dCT = dAT + dQT + dXT_1_5;
            Rat maxCT = durationValue(requestMaxCT(request));
            Rat diffCT = 0;
            if (dCT > maxCT){
                  diffCT = dCT - maxCT;
            }
            Rat phi = requestPhi(request);
            Rat penalty_ct = global_xCT() * diffCT * phi;

            // utility ..

            Rat income = priceValue(requestPrice(request));
            utility = utility + (income - penalty_ct - vm_price);

            // will it survive the time-out?

            Rat penalty_fr = priceValue(requestPenaltyFR(request));
            Rat cut_dCT = maxCT + (global_xCT() * ((income - vm_price + penalty_fr) / phi));
            Rat dTO = cut_dCT - dAT - dXT_1_5;
            if (dTO < dQT){
                  utility = utility - 100000;
                  rejected = Cons(request, rejected);
            }else{
                  accepted = Cons(request, accepted);
            }

            // now for the next request ..

            tSTART = tEND;
            used = True;
            list = tail(list);
      }
      return Triple(utility, accepted, rejected);
```

```
        }


        Solution mutate(Solution solution){
                Problem problem = solutionProblem(solution);
                Map<RequestId, Task> task_map = problemTaskMap(problem);
                List<Pair<RequestId,Int>> old_maps = solutionMaps(solution);
                List<Pair<RequestId,Int>> new_maps = Nil;
                Int i = 0;
                Int replace = random(length(old_maps)); // choose a map to mutate
                while(old_maps != Nil){
                        Pair<RequestId,Int> old_map = head(old_maps);
                        Bool use_old_map = True;
                        if (i == replace){
                                RequestId request_id = fst(old_map);
                                Maybe<Task> maybe_task = lookup(task_map, request_id);
                                if (maybe_task != Nothing){
                                        Task task = fromJust(maybe_task);
                                        Pair<RequestId,Int> new_map = this.randomMap(task);
                                        new_maps = appendright(new_maps, new_map);
                                        use_old_map = False;
                                }
                        }
                        if (use_old_map){
                                new_maps = appendright(new_maps, old_map);
                        }
                        old_maps = tail(old_maps);
                        i = i + 1;
                }
                return this.createSolution(problem, new_maps);
        }

        Solution crossover(Solution s1, Solution s2){
                Problem problem = solutionProblem(s1);
                Map<RequestId, Task> task_map = problemTaskMap(problem);
                List<Pair<RequestId,Int>> old_maps = solutionMaps(s1);
                List<Pair<RequestId,Int>> new_maps = Nil;
                List<RequestId> used = Nil;
                Int i = 0;
                Int n = random(length(old_maps)); // random crossover point
                while (i < n){
                        Pair<RequestId,Int> map = head(old_maps);
                        new_maps = appendright(new_maps, map);
                        used = Cons(fst(map), used);
                        old_maps = tail(old_maps);
                        i = i + 1;
                }
                old_maps = solutionMaps(s2);
                while(old_maps != Nil){
                        Pair<RequestId,Int> map = head(old_maps);
                        Bool already_used = this.contains(used, fst(map));
                        if (!already_used){
                                new_maps = appendright(new_maps, map);
                        }
                        old_maps = tail(old_maps);
                }
                return this.createSolution(problem, new_maps);
        }

        Bool contains(List<RequestId> list, RequestId target){
                Int targetId = requestIdValue(target);
                Bool found = False;
                List<RequestId> l = list;
                while (!found && l != Nil){
                        Int id = requestIdValue(head(l));
                        if (targetId == id){
                                found = True;
                        }
                        l = tail(l);
                }
                return found;
        }

}
```

```
//============================================================================
// Processor
//============================================================================

data Progress = Starting | Processing | Stopping | Stopped;

interface RequestProcessor {
      Unit process(Request request);
      Bool isFree();
}

class RequestProcessor(
      VM vm
) implements RequestProcessor{

      Bool is_free = True;
      Rat vmPower = -1;

      Unit process(Request request){
            if (vmPower < 0){
                  VMData vmData = await vm!vmData();
                  vmPower = vmPower(vmData);
            }
            Rat dXT = (requestSize(request) * (1 + ((randomf() * 2) - (1/10)))) / vmPower;
            Rat dXT_2 = dXT/2;
            vm!notifyProgress(request, Starting);
            duration(dXT_2, dXT_2);
            vm!notifyProgress(request, Processing);
            duration(dXT, dXT);
            vm!notifyProgress(request, Stopping);
            duration(dXT_2, dXT_2);
            vm!notifyProgress(request, Stopped);
      }

      Bool isFree(){ return is_free; }

}




//============================================================================
// VIRTUAL MACHINE
//============================================================================

interface VM extends Solver {
      Bool canKill();
      VMData vmData();
      Unit launch();
      Time launchTime();
      Time startTime();
      Duration executionTime();
      Bool hasPendingRequests();
      Request currentRequest();
      Bool hasCompletedRequests();
      Unit assignRequest(Request request);
      Unit notifyProgress(Request request, Progress progress);
      VMInfo vmInfo(Time actionTime);
      DeploymentComponent dc();
}

class VM(
      VMData vmData,
      Tally tally
) implements VM{

      //====================================
      // Properties
      //====================================

      Time launchTime = Time(-1);
      Time startTime = Time(-1);
      Duration executionTime = Duration(-1);
      Bool hasCompletedRequests = False;
      Solver solver;
      List<Request> queue = Nil;
      Request currentRequest = NoRequest;
```

```
RequestProcessor processor;
DeploymentComponent dc;

//=====================================
// Solver Interface Methods
//=====================================

Unit startSolving(Problem problem, Map<VMId, VM> pool){
      if (solver == null){
            solver = new Solver(vmPower(vmData)); // -> Separate COG on DEPLOYMENT COMP
      }
      Fut<Unit> u = solver!startSolving(problem, pool);
}

Solution stopSolving(){
      Solution solution = NoSolution;
      if (solver != null){
            solution = await solver!stopSolving();
      }
      return solution;
}

Solution bestSolution(){
      Solution solution = NoSolution;
      if (solver != null){
            solution = await solver!bestSolution();
      }
      return solution;
}


//=====================================
// VM Interface Methods
//=====================================

Bool canKill(){
      return currentRequest == NoRequest && length(queue) == 0;
}

VMData vmData(){
      return vmData;
}

DeploymentComponent dc(){
      return dc;
}

Bool isInstance(){
      return True;
}

VMInfo vmInfo(Time actionTime){
      Rat at = timeValue(actionTime);
      Time tNOW = now();
      Rat dQT = this.estimatedQueuingTimeForNewRequests();
      Time tSTART = Time(timeValue(tNOW) + dQT);
      Rat score = this.score();
      return VMInfo(vmData,
                    actionTime,
                    tSTART,
                    launchTime,
                    score,
                    !hasCompletedRequests);
}


Time launchTime(){
      return launchTime;
}

Time startTime(){
      return startTime;
}

Duration executionTime(){ return executionTime; }

Bool hasPendingRequests(){
      return length(queue) > 0;
}

Request currentRequest(){ return currentRequest; }
```

```
Bool hasCompletedRequests(){ return hasCompletedRequests; }


Unit assignRequest(Request request){
      if (processor != null && currentRequest == NoRequest){ // = Available
            Fut<Unit> fu = tally!enqueued(this, request, Duration(-1));
            this.processRequest(request);
      }else{
            Bool used = this.hasCompletedRequests();
            Rat tNOW = timeValue(now());
            Rat tLAUNCH = timeValue(launchTime);

            // machine price …

            Rat dXT = this.expectedExecutionTime(request);
            Rat dPT = 2 * dXT; // expected processing time
            Rat dQT = this.estimatedQueuingTimeForNewRequests();
            Rat tSTART = tNOW + dQT;
            Rat uc = priceValue(vmUnitCost(vmData));
            Rat price_at_start = 0;
            if (used){
                  price_at_start = uc * ceiling((tSTART - tLAUNCH) / global_du());
            }
            Rat tEND = tSTART + dPT;
            Rat price_at_end = uc * ceiling((tEND - tLAUNCH) / global_du());
            Rat vm_price = price_at_end - price_at_start;

            // cut-off completion time ..

            Rat maxCT = durationValue(requestMaxCT(request));
            Rat request_price = priceValue(requestPrice(request));
            Rat penalty_fr = priceValue(requestPenaltyFR(request));
            Rat phi = requestPhi(request);
            Rat dCT = maxCT + (global_xCT()
                        * ((request_price - vm_price + penalty_fr) / phi));

            // time-out
            Rat tRCV = timeValue(requestReceiveTime(request));
            Rat dACT = tNOW - tRCV;
            Rat dTO = dCT - dACT - ((15/10) * dXT);

            if (dTO < dQT){
                  // reject the request
                  Fut<Unit> fu = tally!rejected(request);
            }else{
                  queue = appendright(queue, request);
                  Fut<Unit> fu = tally!enqueued(this, request, Duration(dTO));
                  fu = this!delayedTimeOut(request, dTO);
                  if (processor != null && currentRequest == NoRequest){
                        this.processRequest(request); // Q: is this ever called ?
                  }
            }
      }
}

Unit launch(){
      launchTime = now();
      dc = thisDC();
      Fut<Unit> fu = tally!launched(this);
      Duration dDT = vmDeployTime(vmData);
      Rat d = durationValue(dDT);
      await duration(d, d); // WAIT (non-blocking) for duration d
      processor = new RequestProcessor(this); // —> Separate COG on DEPLOYMENT COMP
      fu = tally!deployed(this);
      Unit u = this.processNextRequestIfAny();
}

Unit notifyProgress( Request request, Progress progress ){
      if (currentRequest == request){
            Rat dXT = this.expectedExecutionTime(request);
            Rat dXT_2 = dXT/2;
            if (progress == Starting){
                  Fut<Unit> fu = tally!settingUp(this, request, Duration(dXT_2));
            }else if (progress == Processing){
                  Fut<Unit> fu = tally!executing(this, request, Duration(dXT));
            }else if (progress == Stopping){
                  Fut<Unit> fu = tally!cleaningUp(this, request, Duration(dXT_2));
            }else{
                  currentRequest = NoRequest;
                  Fut<Unit> fu = tally!finished(this, request);
                  startTime = Time(-1);
                  executionTime = Duration(-1);
```

```
                    Unit u = this.processNextRequestIfAny();
                }
            }
        }
        //=====================================
        // Private
        //=====================================

        Rat score(){
              Rat spec = vmSpec(vmData);
              if (currentRequest != NoRequest){
                    spec = spec * 10; // bias available instances
              }
              Rat dQT = this.estimatedQueuingTimeForNewRequests();
              return (priceValue(vmUnitCost(vmData)) * dQT ) / spec;
        }

        Rat estimatedQueuingTimeForNewRequests(){
              Rat d = 0;
              List<Request> list = queue;
              while (list != Nil){
                    Request request = head(list);
                    Rat dXT = this.expectedExecutionTime(request);
                    d = d + (2 * dXT); // sum the processing times for queued requests
                    list = tail(list);
              }
              Rat tNOW = timeValue(now());
              if (processor == null){ // = not yet deployed
                    Duration dt = vmDeployTime(vmData);
                    d = d + durationValue(dt) - (tNOW - timeValue(launchTime));
              }else if (currentRequest != NoRequest){ // = Busy
                    Rat tSTART = timeValue(startTime);
                    Rat dXT = durationValue(executionTime);
                    if (tSTART >= 0 && dXT >= 0){
                          d = d + (2 * dXT) - (tNOW - tSTART);
                    }
              }
              if (d < 0){ d = 0; }
          return d;
        }

        Unit processNextRequestIfAny(){
              if (length(queue) > 0){
                    Request request = head(queue);
                    queue = without(queue, request);
                    Unit u = this.processRequest(request);
              }
        }

        Unit processRequest(Request request){
              if (processor != null && currentRequest == NoRequest){ // = Available
                    currentRequest = request;
                    startTime = now();
                    Rat dXT = this.expectedExecutionTime(request);
                    executionTime = Duration(dXT);
                    Fut<Unit> u = processor!process(request);
              }
              // TODO - otherwise throw some kind of IllegalStateException?
        }

        Rat expectedExecutionTime(Request request){
              return ((11/10) * requestSize(request)) / vmPower(vmData);
        }

        Unit delayedTimeOut(Request request, Rat dTO){ // called by assignRequest
              await duration(dTO, dTO);
              Bool still_queued = this.enqueued(request);
              if (still_queued){
                    queue = without(queue, request); // reject the request
                    Fut<Unit> fu = tally!timedOut( request );
              }
        }

        Bool enqueued(Request request){
              Int target = requestIdValue(requestId(request));
              List<Request> q = queue;
              Bool found = False;
              while(!found && q != Nil){
                    Int r = requestIdValue(requestId(head(q)));
                    if (r == target){ found = True; }
                    q = tail(q);
              }
            return found;
        }

}

//============================================================================
// RPM
//============================================================================

interface RPM{
        Unit receive(Request request);
}

class RPM(
        Tally tally
) implements RPM{

        List<Request> pendingRequests = Nil;
        Bool active = False;
        Solver solver;
        Map<VMId, VM> resourcePool = EmptyMap;
        Int cycle_count = 0;
        CloudProvider cloudProvider;

        // INIT
        {
              solver = new Solver(global_max_vm_power()); // -> Separate COG
              cloudProvider = new CloudProvider("name of a cloud provider"); // -> Separate COG
        }

        Unit receive(Request request){
              tally!received(request);
              pendingRequests = appendright(pendingRequests, request);
              if (!active){
                    this!activate();
              }
        }

        Unit activate(){

              List<Request> pending_requests = pendingRequests;
              pendingRequests = Nil;
              active = True;

              Rat tNOW = timeValue(now());
              Rat tACT = tNOW + global_dAT();
              Map<VMId, VMInfo> vm_info_map = EmptyMap;

              // **** stop all running solvers & get best solution ..

              Solution best = await solver!stopSolving();
              Rat best_u = -10000000;
              if (best != NoSolution){
                    best_u = priceValue(solutionUtility(best));
              }
              List<VM> vms = values(resourcePool);
              while (vms != Nil){
                    VM vm = head(vms);
                    VMInfo vm_info = await vm!vmInfo(Time(tACT));
                    vm_info_map = put(vm_info_map, vmId(vmInfoVMData(vm_info)), vm_info);
                    Solution vm_best = await vm!stopSolving();
                    Rat vm_best_u = -10000000;
                    if (vm_best != NoSolution){
                          vm_best_u = priceValue(solutionUtility(vm_best));
                    }
                    if (vm_best_u > best_u){
                          best = vm_best;
                          best_u = vm_best_u;
                    }
                    vms = tail(vms);
              }
              if (best == NoSolution){
                    println("WARNING: no best solution");
              }else{
                    List<Pair<VMInfo, List<Request>>> assigned = values(solutionAssignments(best));
                    while (assigned != Nil){
                          Pair<VMInfo, List<Request>> a = head(assigned);
                          this.assign(fst(a),snd(a));
                          assigned = tail(assigned);
                    }
                    List<Request> rejected = solutionRejections(best);
                    while (rejected != Nil){
```

```
                    this.reject(head(rejected));
                    rejected = tail(rejected);
                }
            }

            // **** restart the solvers for the next round ..


            Map<RequestId,Task> task_map = EmptyMap;
            List<Request> temp_pending_requests = pending_requests;

            while(temp_pending_requests != Nil){
            Request request = head(temp_pending_requests);
                VMInfo vm_info = this.vmInfoForDefaultVM(request, Time(tACT));
                vm_info_map = put(vm_info_map, vmId(vmInfoVMData(vm_info)), vm_info);
                temp_pending_requests = tail(temp_pending_requests);
            }
            temp_pending_requests = pending_requests;
            while(temp_pending_requests != Nil){
                Request request = head(temp_pending_requests);
                List<VMInfo> valid_vm_info_list = Nil;
                List<VMInfo> vm_info_list = values(vm_info_map);
                while(vm_info_list != Nil){
                    VMInfo vm_info = head(vm_info_list);
                    VMData vm_data = vmInfoVMData(vm_info);
                    if (vmDisk(vm_data) >= requestSize(request)){
                        valid_vm_info_list = Cons(vm_info, valid_vm_info_list);
                    }
                    vm_info_list = tail(vm_info_list);
                }
                if (length(valid_vm_info_list) > 0){
                    valid_vm_info_list = this.qsortByAscendingScore(valid_vm_info_list);
                    task_map =
                       put(task_map, requestId(request), Task(request, valid_vm_info_list));
                }else{
                    this.reject(request);
                }
                temp_pending_requests = tail(temp_pending_requests);
            }
            if (length(values(task_map)) > 0){
                Problem problem = Problem(task_map, vm_info_map);
                solver!startSolving(problem, resourcePool);
                vms = values(resourcePool);
                while (vms != Nil){
                    VM vm = head(vms);
                    vm!startSolving(problem, resourcePool);
                    vms = tail(vms);
                }
                await duration(global_dAT(), global_dAT());
                this!activate();
            }else{
                active = False;
            }
        }


        // ----------------------------------------
        // Quicksort VMInfo list by ascending score
        // ----------------------------------------

        List<VMInfo> qsortByAscendingScore(List<VMInfo> list){
            List<VMInfo> res = Nil;
            VMInfo head = head(list);
            Pair<List<VMInfo>, List<VMInfo>> split = this.qsplit( head, tail(list) );
            List<VMInfo> sorted_small = fst(split);
            if (sorted_small != Nil){
                sorted_small = this.qsortByAscendingScore( sorted_small );
            }
            List<VMInfo> sorted_big = snd(split);
            if (sorted_big != Nil){
                sorted_big = this.qsortByAscendingScore( sorted_big );
            }
            if (sorted_small != Nil){
                if (sorted_big != Nil){
                    res = concatenate( sorted_small, Cons( head, sorted_big ) );
                }else{
                    res = appendright( sorted_small, head );
                }
            }else if (sorted_big != Nil){
                res = Cons( head, sorted_big);
            }else{
                res = list[ head ];
```

```
      }
      return res;
}


VMInfo vmInfoForDefaultVM(Request r, Time actionTime){
      VMData vm_data = requestDefaultVMData(r);
      Rat tACT = timeValue(actionTime);
      Rat tDT = durationValue(vmDeployTime(vm_data));
      Rat ut = priceValue(vmUnitCost(vm_data));
      Rat score = (ut * tDT) / vmSpec(vm_data);
      return VMInfo(vm_data,actionTime,Time(tACT + tDT),actionTime,score,True);
}

Pair<List<VMInfo>, List<VMInfo>> qsplit(VMInfo x, List<VMInfo> list){
      Rat x_score = vmScore(x);
      List<VMInfo> smaller = Nil;
      List<VMInfo> bigger = Nil;
      while(list != Nil){
            VMInfo h = head(list);
      Rat h_score = vmScore(h);
            if (h_score > x_score ){
                  bigger = Cons(h, bigger);
            }else{
                  smaller = Cons(h, smaller);
            }
            list = tail(list);
      }
      return Pair(smaller, bigger);
}

Unit reject(Request request){
      tally.rejected(request);
}

Unit assign(VMInfo vm_info, List<Request> requests){
      VMData vm_data = vmInfoVMData(vm_info);
      VMId vm_id = vmId(vm_data);
      Maybe<VM> maybe_vm = lookup(resourcePool, vm_id);
      VM vm = null;
      if (maybe_vm == Nothing){
            vm = this.createAndLaunchNewVM(vm_data);
      }else{
            vm = fromJust(maybe_vm);
      }
      List<Request> rs = requests;
      while(rs != Nil){
            vm!assignRequest(head(rs));
            rs = tail(rs);
      }
}

VM createAndLaunchNewVM( VMData vm_data ){
      Map<Resourcetype, Rat> resources = this.resourceMap( vm_data );
      DeploymentComponent dc = cloudProvider.launchInstance( resources );
      [DC: dc] VM vm = new VM( vm_data, tally );
      await vm!launch();
      put( resourcePool, vmId(vm_data), vm ); // add the new vm to the pool
      this!killVMWhenPossible( vm );
      return vm;
}

Map<Resourcetype, Rat> resourceMap( VMData vm_data ){
      Map<Resourcetype, Rat> result = EmptyMap;
      put( result, Cores, vmClock(vm_data) * vmCores(vm_data) );
      put( result, Memory, vmMemory(vm_data) );
      return result;
}

Unit killVMWhenPossible(VM vm){
      Rat du = global_du(); // wait a fixed time
      await duration(du, du);
      Bool canKill = await vm!canKill();
      if (canKill){
            VMData vm_data = await vm!vmData();
            resourcePool = removeKey(resourcePool, vmId(vm_data));
            DeploymentComponent dc = await vm!dc();
            dc.release();
      cloudProvider.shutdownInstance( dc );
            tally.killed(vm);
      }else{
            this!killVMWhenPossible(vm); // keep calling ...
            }
      }

}


//=============================================================================
// VMConfig - used for calculating default VM in Simulator
//=============================================================================

data VMConfig = NoVMConfig | VMConfig(Rat vmcClock, Int vmcCores, Int vmcMemory);

def List<VMConfig> vmConfigs() = list[

      // clk = 1.0

      VMConfig(1, 1, 1),
      VMConfig(1, 1, 2),
      VMConfig(1, 1, 4),
      VMConfig(1, 1, 8),
      VMConfig(1, 2, 1),
      VMConfig(1, 2, 2),
      VMConfig(1, 2, 4),
      VMConfig(1, 2, 8),
      VMConfig(1, 4, 1),
      VMConfig(1, 4, 2),
      VMConfig(1, 4, 4),
      VMConfig(1, 4, 8),
      VMConfig(1, 8, 1),
      VMConfig(1, 8, 2),
      VMConfig(1, 8, 4),
      VMConfig(1, 8, 8),

      // clk = 1.25

      VMConfig(125/100, 1, 1),
      VMConfig(125/100, 1, 2),
      VMConfig(125/100, 1, 4),
      VMConfig(125/100, 1, 8),
      VMConfig(125/100, 2, 1),
      VMConfig(125/100, 2, 2),
      VMConfig(125/100, 2, 4),
      VMConfig(125/100, 2, 8),
      VMConfig(125/100, 4, 1),
      VMConfig(125/100, 4, 2),
      VMConfig(125/100, 4, 4),
      VMConfig(125/100, 4, 8),
      VMConfig(125/100, 8, 1),
      VMConfig(125/100, 8, 2),
      VMConfig(125/100, 8, 4),
      VMConfig(125/100, 8, 8),

      // clk = 1.5

      VMConfig(15/10, 1, 1),
      VMConfig(15/10, 1, 2),
      VMConfig(15/10, 1, 4),
      VMConfig(15/10, 1, 8),
      VMConfig(15/10, 2, 1),
      VMConfig(15/10, 2, 2),
      VMConfig(15/10, 2, 4),
      VMConfig(15/10, 2, 8),
      VMConfig(15/10, 4, 1),
      VMConfig(15/10, 4, 2),
      VMConfig(15/10, 4, 4),
      VMConfig(15/10, 4, 8),
      VMConfig(15/10, 8, 1),
      VMConfig(15/10, 8, 2),
      VMConfig(15/10, 8, 4),
      VMConfig(15/10, 8, 8)

      ];

def VMConfig minVMConfig() = head(vmConfigs());

def VMConfig maxVMConfig() = nth(vmConfigs(), length(vmConfigs()) - 1);

def Rat vm_power(VMConfig cf) = vmPower3(vmcClock(cf), vmcCores(cf), vmcMemory(cf));
```

```
//=============================================================================
// SIMULATOR
//=============================================================================

interface Simulator {
      Unit start();
}

class Simulator(
      List<SLA> slas
)
implements Simulator {

      Unit start(){
            Tally tally = new Tally(); // -> Separate COG
            tally!simulationStarted();
            RPM rpm = new RPM(tally); // -> Separate COG
            Int count = 0;
            Int max = global_request_count();
            while (count < max){
                  count = count + 1;
                  Request request = this.nextRequest(count);
                  Time rcv = requestReceiveTime(request);
                  Rat wake = timeValue(rcv) - timeValue(now());
                  duration(wake, wake);
                  rpm!receive(request); // ******** ASYNC CALL
            }
            tally!simulationEnded();
      }

      Request nextRequest(Int count){
            Rat time = timeValue(now()) + ( randomf() * (5/1000) * (count%200) );
            Rat min = global_min_request_size();
            Rat max = global_max_request_size();
            Rat size = min + ( randomf() * ( max - min ) );
            Rat p = adHocPriority();
            if ( randomf() > 8/10 ){
                  p = scheduledPriority();
            }
            RequestPriority priority = RequestPriority(p);
            SLA sla = nth(slas, random(length(slas)));
            Rat phi = rawRequestPhi(sla, priority);
            VMData defaultVM = this.calculateDefaultVM(VMId(count), size, phi);
            return Request(   RequestId(count),
                              size,
                              random(slaUserCount(sla)),
                              priority,
                              Time(time),
                              sla,
                              defaultVM);
      }

      //========================================
      // Private - Calculations for Default VM
      //========================================

      VMData calculateDefaultVM(VMId vmId, Rat size, Rat phi){
            Rat pow = sqrt((825/100)*size);
            Int disk = next_power_of_2(size, 1); //truncate(power(2,ceiling(log2(size))));
            Pair<VMConfig, VMConfig> closest = this.closestAvailablePowers(pow);
            VMConfig a = fst(closest);
            VMConfig b = snd(closest);
            VMData d1 = VMData(vmId, vmcClock(a), vmcCores(a), vmcMemory(a), disk);
            Rat u1 = this.expectedUtilityForDefaultVM(d1, size, phi);
            if (b != NoVMConfig){
                  VMData d2 = VMData(vmId, vmcClock(b), vmcCores(b), vmcMemory(b), disk);
                  Rat u2 = this.expectedUtilityForDefaultVM(d2, size, phi);
                  if (u2 > u1){
                        d1 = d2;
                  }
            }
            return d1;
      }

      Pair<VMConfig,VMConfig> closestAvailablePowers(Rat targetPower){
            Pair<VMConfig,VMConfig> result = Pair(NoVMConfig, NoVMConfig);
            VMConfig min_config = minVMConfig();
            Rat min_power = vm_power(min_config);
            if (targetPower < min_power){
                  result = Pair(min_config, NoVMConfig);
            }else{
                  VMConfig max_config = maxVMConfig();
                  Rat max_power = vm_power(max_config);
                  if (targetPower > max_power){
                        result = Pair(max_config, NoVMConfig);
                  }else{
                        VMConfig last = NoVMConfig;
                        Bool done = False;
                        List<VMConfig> configs = vmConfigs();
                        while ( !done && configs != Nil ){
                              VMConfig config = head(configs);
                              Rat pow = vm_power(config);
                              if (pow >= targetPower){
                                    result = Pair(config, last);
                                    done = True;
                              }
                              last = config;
                              configs = tail(configs);
                        }
                  }
            }
            return result;
      }

      Rat expectedUtilityForDefaultVM(VMData d, Rat size, Rat phi){
            Rat pow = vmPower(d);
            Rat disk = vmDisk(d);
            Rat b = size / pow;
            Rat f = (2/10) * (pow + disk);
            Rat a = f + ((22/10) * b);
            Rat dCT = f + ((165/100) * b) + global_dAT();
            Rat deltaCT = 0;
            Duration max = rawRequestMaxCT(size, phi);
            Rat mCT = durationValue(max);
            if ( dCT > mCT ){
                  deltaCT = dCT - mCT;
            }
            Price xu = vmUnitCost(d);
            Rat uc = priceValue(vmUnitCost(d));
            return (phi *
                ((global_xP()*size) - (global_xCT()*dCT))) - (uc*ceiling(a/global_du()));
      }

}



//============================================================================
// TALLY
//============================================================================

interface Tally {
      Unit simulationStarted();
      Unit simulationEnded();
      Unit received(Request r);
      Unit enqueued(VM vm, Request r, Duration timeOut);
      Unit settingUp(VM vm, Request r, Duration d);
      Unit executing(VM vm, Request r, Duration d);
      Unit cleaningUp(VM vm, Request r, Duration d);
      Unit finished(VM vm, Request r);
      Unit rejected(Request r);
      Unit timedOut(Request r);
      Unit launched(VM vm);
      Unit deployed(VM vm);
      Unit killed(VM vm);
}

class Tally()
implements Tally {

      //===================================
      // Properties

      Rat total_utility = 0;
      Rat total_Request_income = 0;
      Rat total_CT_penalties = 0;
      Rat total_FR_penalties = 0;
      Rat total_VM_cost = 0;
```

```
// SLA.id -> [ User.id -> ( failure_count, success_count ) ]
Map<Int,Map<Int,Pair<Int,Int>>> histories = EmptyMap;


//====================================
// Interface Methods

Unit simulationStarted(){
      // logging
      if (global_logging()){
            println("START");
      }
}

Unit simulationEnded(){
      // logging
      if (global_logging()){
            println("END");
      }
}

Unit received(Request r){
      // logging
      if (global_logging()){
            String s = this.requestStr(r);
            s = "RECEIVED:" + s;
            println(s);
      }
}

Unit enqueued(VM vm, Request r, Duration timeOut){
      // logging
      if (global_logging()){
            String s = this.vmIdAndRequestIdAndDurationStr(vm, r, timeOut);
            s = "ENQUEUED:" + s;
            println(s);
      }
}

Unit settingUp(VM vm, Request r, Duration d){
      // logging
      if (global_logging()){
            String s = this.vmIdAndRequestIdAndDurationStr(vm, r, d);
            s = "SETTING_UP:" + s;
            println(s);
      }
}

Unit executing(VM vm, Request r, Duration d){
      // logging
      if (global_logging()){
            String s = this.vmIdAndRequestIdAndDurationStr(vm, r, d);
            s = "EXECUTING:" + s;
            println(s);
      }
}

Unit cleaningUp(VM vm, Request r, Duration d){
      this.requestHandled(r, False);
      // logging
      if (global_logging()){
            String s = this.vmIdAndRequestIdAndDurationStr(vm, r, d);
            s = "CLEANING_UP:" + s;
            println(s);
      }
}

Unit finished(VM vm, Request r){
      // logging
      if (global_logging()){
            String s = this.vmIdAndRequestIdStr(vm, r);
            s = "FINISHED:" + s;
            println(s);
      }
}

Unit rejected(Request r){
      this.requestHandled(r, True);
      // logging
      if (global_logging()){
            String s = this.requestIdStr(r);
            s = "REJECTED:" + s;
            println(s);
        }
}

Unit timedOut(Request r){
        this.requestHandled(r, True);
        // logging
        if (global_logging()){
            String s = this.requestIdStr(r);
            s = "TIMED_OUT:" + s;
            println(s);
        }
}

Unit launched(VM vm){
        // logging
        if (global_logging()){
            String s = this.vmStr(vm);
            s = "LAUNCHED:" + s;
            println(s);
        }
}

Unit deployed(VM vm){
        // logging
        if (global_logging()){
            String s = this.vmIdStr(vm);
            s = "DEPLOYED:" + s;
            println(s);
        }
}

Unit killed(VM vm){
        // calculate total cost of using the VM ..
        Time launchTime = await vm!launchTime();
        VMData vm_data = await vm!vmData();
        Rat unitCost = priceValue(vmUnitCost(vm_data));
        Rat t_now = timeValue( now() );
        Rat t_then = timeValue(launchTime)/global_du();
        Rat time_units = ceiling(t_now - t_then)/1;
        Rat vm_cost = unitCost * time_units;
        this.incrementTotalVMCostsBy(vm_cost); // <== VM COST APPLIED
        // logging
        if (global_logging()){
            String s = this.vmIdStr(vm);
            s = "KILLED:" + s;
            println(s);
        }
}

//====================================
// Private - Tallies

Unit incrementTotalRequestIncomeBy(Rat request_price){
        total_Request_income = total_Request_income + request_price;
        total_utility = total_utility + request_price;
        // logging
        this.printTotal("INCOME", total_Request_income);
        this.printTotal("UTILITY", total_utility);
}

Unit incrementTotalCTPenaltiesBy(Rat penalty_ct){
        total_CT_penalties = total_CT_penalties + penalty_ct;
        total_utility = total_utility - penalty_ct;
        // logging
        this.printTotal("CT_PENALTIES", total_CT_penalties);
        this.printTotal("UTILITY", total_utility);
}

Unit incrementTotalFRPenaltiesBy(Rat penalty_fr){
        total_FR_penalties = total_FR_penalties + penalty_fr;
        total_utility = total_utility - penalty_fr;
        // logging
        this.printTotal("FR_PENALTIES", total_FR_penalties);
        this.printTotal("UTILITY", total_utility);
}
```

```
Unit incrementTotalVMCostsBy(Rat vm_cost){
      total_VM_cost = total_VM_cost + vm_cost;
      total_utility = total_utility - vm_cost;
      // logging
      this.printTotal("VM_COST", total_VM_cost);
      this.printTotal("UTILITY", total_utility);
}


Unit printTotal(String key, Rat value){
      // logging
      if (global_logging()){
            String s = toString(value);
            s = key + "=" + s;
            println(s);
      }
}


//====================================
// Private - Histories

Unit requestHandled(Request request, Bool failed){
      SLA sla = requestSLA(request);
      Int sla_id = slaId(sla);
      Maybe<Map<Int,Pair<Int,Int>>> maybe_sla_record = lookup(histories, sla_id);
      Map<Int,Pair<Int,Int>> sla_record = EmptyMap;
      if (maybe_sla_record != Nothing){
            sla_record = fromJust(maybe_sla_record);
      }
      Int user_id = requestUser(request);
      Maybe<Pair<Int,Int>> maybe_user_record = lookup(sla_record, user_id);
      Pair<Int,Int> user_record = Pair(0,0);
      if (maybe_user_record != Nothing){
            user_record = fromJust(maybe_user_record);
      }
      Int failure_count = fst(user_record);
      Int success_count = snd(user_record);
      if (failed){
            // request failed ..
            failure_count = failure_count + 1;
      }else{
            // request completed ..
            success_count = success_count + 1;
            // calculate the income ..
            Rat price = priceValue(requestPrice(request));
            this.incrementTotalRequestIncomeBy(price); // <== REQUEST INCOME APPLIED
            // calculate CT penalty ..
            Rat dCT = timeValue(now()) - timeValue(requestReceiveTime(request));
            Rat maxCT = durationValue(requestMaxCT(request));
            if (dCT > maxCT){
                  Rat penalty_ct = global_xCT() * (dCT - maxCT) * requestPhi(request);
                  this.incrementTotalCTPenaltiesBy(penalty_ct); // <== CT PENALTY
            }
      }
      if (failure_count + success_count >= global_N()){
            Rat fr_penalty = this.calculateFailureRatePenalty(sla, failure_count);
            this.incrementTotalFRPenaltiesBy(fr_penalty); // <== FR PENALTY
            failure_count = 0;
            success_count = 0;
      }
      user_record = Pair(failure_count, success_count);
      sla_record = put(sla_record, user_id, user_record);
      put(histories, sla_id, sla_record);
}

Rat calculateFailureRatePenalty(SLA sla, Int failure_count){
      Rat result = 0;
      Rat max_fr = maxFR(sla);
      if (failure_count > max_fr){
            Int diff = failure_count - max_fr;
            Rat kSL = serviceLevelValue(slaServiceLevel(sla));
            result = diff * global_xFR() * kSL * kSL;
      }
      return result;
}
```

```
	//====================================
	// Private - Strings

	String requestStr(Request request){
		String r_id = this.requestIdStr(request);
		String r_size = toString(requestSize(request));
		String r_pri = toString(requestPriorityValue(requestPriority(request)));
		return "RQ(" + r_id + ",size=" + r_size + ",pri=" + r_pri + ")";
	}

	String vmStr(VM vm){
		VMData vm_data = await vm!vmData();
		String vm_id = this.vmDataIdStr(vm_data);
		String vm_power = toString(vmPower(vm_data));
		String vm_disk = toString(vmDisk(vm_data));
		return "VM(" + vm_id + ",power=" + vm_power + ",disk=" + vm_disk + ")";
	}

	String vmIdAndRequestIdAndDurationStr(VM vm, Request request, Duration d){
		String vm_and_r_ids = this.vmIdAndRequestIdStr(vm, request);
		String dur = this.durationStr(d);
		return vm_and_r_ids + ",D(" + dur + ")";
	}

	String vmIdAndRequestIdStr(VM vm, Request request){
		String vm_id = this.vmIdStr(vm);
		String r_id = this.requestIdStr(request);
		return "VM(" + vm_id + "),RQ(" + r_id + ")";
	}

	String requestIdStr(Request request){
		String s = toString(requestIdValue(requestId(request)));
		return "id=" + s;
	}

	String vmIdStr(VM vm){
		VMData vm_data = await vm!vmData();
		return this.vmDataIdStr(vm_data);
	}

	String vmDataIdStr(VMData vm_data){
		String s = toString(vmIdValue(vmId(vm_data)));
		return "id=" + s;
	}

	String durationStr(Duration d){
		String s = toString(durationValue(d));
		return "dur=" + s;
	}

}




//=====================================================================
// MAIN
//=====================================================================

{

	List<SLA> slas = list[
		SLA(0, 0, 10, ServiceLevel(bronzeSL())),
		SLA(1, 0, 10, ServiceLevel(silverSL())),
		SLA(2, 0, 10, ServiceLevel(goldSL()))
	];
	Simulator simulator = new local Simulator(slas);
	simulator.start();

}
```

B. Skeleton ABS Model for the ETICS Case Study

This appendix lists the *skeleton* code for the ABS model for Engineering's ETICS case study. This code is a greatly reduced version of the code given in Appendix A, aimed at capturing just the essential asynchronous control flow between the RPM and VMs - in particular:

- There is no distributed genetic algorithm (DGA). Instead, requests are either rejected or assigned to either existing or new VMs entirely at random. Accordingly, there is no need for the complex data types supporting the DGA, and indeed all the functional data types have been removed;
  - For legibility, we retain **`Request`** and **`Solution`** types, but these are defined just as **`String`** aliases (i.e. with no structured content);
- The **`Solver`** class is retained, but its only role is to occasionally request a solution from a randomly selected (remote) VM - thus capturing the 'distributed' aspect of the DGA;
- VMs are generated just as in the complete code, but each VM has identical *clock*, *cores* & *memory* resource properties, and the request **`Processor`** class has been simplified;
- The **`RPM`** remains largely intact, except that all solutions returned from solvers are ignored;
- All other interfaces/classes have been removed.

The skeleton code runs to 333 lines, and is structured into the following sections:

- Globals
- Dummy data types (defined as **`String`** aliases):
  `Request`
  `Solution`
- Interfaces & classes:
  `Solver`
  `Processor`
  `VM`
  `RPM`
- The **`main`** method

The skeleton ABS model is as follows:

```
//==========================================================================
// ETICS.abs
// SKELETON VERSION OF THE ENG "ETICS" MODEL
//==========================================================================

module ETICS;

import * from ABS.DC;


//==========================================================================
// GLOBALS
//==========================================================================

def Rat global_du() = 10; // VM time unit (in minutes)
def Rat global_dAT() = 1; // action time

def Rat randomf() = random(10000000)/10000000;


//==========================================================================
// DUMMY TYPES
//==========================================================================

type Request = String;
type Solution = String;
```

```
//===========================================================================
// SOLVER
//===========================================================================

interface Solver {
      Unit startSolving();
      Solution stopSolving();
      Solution bestSolution();
}

class Solver(RPM rpm) implements Solver{

      Bool cancelled = False;
      Solution best = "DUMMY SOLUTION";

      Solution stopSolving(){
            cancelled = True;
            return best;
      }

      Solution bestSolution(){ return best; }

      Unit startSolving(){
            cancelled = False;
            while (!cancelled){
                  await duration(2,2);
                  if (random(5) == 0){ // get 'best' solution from peer
                        List<VM> all_vms = await rpm!resourcePool();
                        Int n = random( length(all_vms) );
                        VM vm = nth(all_vms, n);
                        best = await vm!bestSolution();
                  }
            }
    }

}



//===========================================================================
// Processor
//===========================================================================

interface Processor {
      Unit process(Request request);
      Bool isFree();
}

class Processor(RPM rpm) implements Processor{

      Bool is_free = True;

      Unit process(Request request){
            println("Starting processing request " + request);
            is_free = False;
            await duration(1,2); // fake 'request processing'
            is_free = True;
            println("Finished processing request " + request);
      }

      Bool isFree(){ return is_free; }

}



//=======================================
// VM
//=======================================

interface VM{
      Bool canKill();
      Unit launch();
      Unit assignRequest(Request request);
      DeploymentComponent dc();

      // from Solver
      Unit startSolving();
      Solution stopSolving();
      Solution bestSolution();
}
```

```
class VM(RPM rpm) implements VM{

      Solver solver;
      DeploymentComponent dc;
      Processor processor;
      List<Request> queue = Nil;

      // --------------------
      // INIT
      // --------------------
      {
            solver = new Solver(rpm); // -> On New COG
            processor = new Processor(rpm); // -> On New COG
      }


      // ------------------------
      // Solver Interface Methods
      // ------------------------

      Unit startSolving(){
            solver!startSolving();
      }

      Solution stopSolving(){
            return await solver!stopSolving();
      }

      Solution bestSolution(){
            return await solver!bestSolution();
      }


      // --------------------
      // VM Interface Methods
      // --------------------

      Bool canKill(){
            Bool result = (length(queue) == 0);
            if (result){
                  result = await processor!isFree();
            }
            return result;
      }

      Unit launch(){
            dc = thisDC();
            await duration(1, 1); // fake 'system boot up time'
            this.processNextRequestIfAny();
      }

      Unit assignRequest(Request request){
            queue = Cons(request, queue);
            this!processNextRequestIfAny(); // async
      }

      DeploymentComponent dc(){
            return thisDC();
      }


      // ------------------
      // Private Methods
      // ------------------

      Unit processNextRequestIfAny(){
            if (processor != null){
                  Bool free = await processor!isFree();
                  if (free && length(queue) > 0){
                        Request request = head(queue);
                        queue = tail(queue);
                        await processor!process(request);
                        this!processNextRequestIfAny(); // process remaining requests
                  }
            }
      }

}
```

```
//===========================================================================
// RPM
//===========================================================================

interface RPM{
      Unit receive(Request request);
      List<VM> resourcePool(); // called by Solver (above)
}
class RPM() implements RPM{

      List<Request> pendingRequests = Nil;
      List<VM> resourcePool = Nil;
      Bool active = False;
      Solver solver;
      CloudProvider cloudProvider;
      List<Request> requestsFromLastCycle = Nil;

      // --------------------
      // INIT
      // --------------------
      {
            solver = new Solver(this); // -> Separate COG
            cloudProvider = new CloudProvider("name of a cloud provider"); // -> Separate COG
      }

      // ---------------------
      // RPM Interface Methods
      // ---------------------

      Unit receive(Request request){
            println("RPM received request " + request);
            pendingRequests = appendright(pendingRequests, request);
            if (!active){
                  this!activate();
            }
      }

      List<VM> resourcePool(){
            return resourcePool;
      }

      // ---------------------
      // Private Methods
      // ---------------------

      Unit activate(){
            List<Request> pending_requests = pendingRequests;
            pendingRequests = Nil;
        active = True;

            // **** stop all running solvers & get best solution ..

            Solution best = await solver!stopSolving(); // we can ignore the result
            List<VM> vms = resourcePool;
            while (vms != Nil){
                  VM vm = head(vms);
                  Solution s = await vm!stopSolving(); // we can ignore the result
                  vms = tail(vms);
            }

            // **** fake application of the solution (fake = random) ..

            while (requestsFromLastCycle != Nil){
                  Request request = head(requestsFromLastCycle);
                  Int action = this.getAction();
                  VM vm;
                  if (action == 0){
                        println("RPM rejected request " + request);
                  }else if (action == 1){
                        println("RPM assign request " + request + " to existing VM");
                        Int n = random( length(resourcePool) );
                        vm = nth(resourcePool, n);
                  }else{
                        println("RPM assign request " + request + " to new VM");
                        vm = this.createAndLaunchNewVM();
                  }
                  if (vm != null){
                        await vm!assignRequest(request);
                  }
                  requestsFromLastCycle = tail(requestsFromLastCycle);
            }
            // **** restart the solvers for the next round ..

            requestsFromLastCycle = pending_requests;
            vms = resourcePool;

            if (pending_requests != Nil){
                  solver!startSolving();
                  vms = resourcePool;
                  while (vms != Nil){
                        VM vm = head(vms);
                        vm!startSolving();
                        vms = tail(vms);
                  }
                  await duration(global_dAT(), global_dAT()); // main cycle duration
                  this!activate();
            }else{
                  active = False;
            }
      }

      // returns a random action (0:reject | 1:assign to existing VM | 2:assign to new VM)
      Int getAction(){
            Int action = 0; // reject the request
            Int r = random(11); // 0 to 10 incl.
            if (r < 7){
                  action = 1; // assign to existing VM
                  if (length(resourcePool) == 0){
                        action = 2; // assign to new VM
                  }
            }else if (r < 9){
                  action = 2; // assign to new VM
            }
            return action;
      }

      VM createAndLaunchNewVM(){
            Map<Resourcetype, Rat> resources = this.resourceMap();
            DeploymentComponent dc = cloudProvider.launchInstance( resources );
            [DC: dc] VM vm = new VM(this);
            await vm!launch();
            resourcePool = Cons(vm, resourcePool); // add the new vm to the pool
            this!killVMWhenPossible( vm );
            return vm;
      }

      Map<Resourcetype, Rat> resourceMap(){
            Map<Resourcetype, Rat> result = EmptyMap;
            put( result, Cores, 1 );
            put( result, Memory, 1 );
            return result;
      }

      Unit killVMWhenPossible(VM vm){
            Bool canKill = await vm!canKill();
            if (canKill){
                  resourcePool = without(resourcePool, vm); // remove the VM from the pool
                  DeploymentComponent dc = await vm!dc();
                  dc.release();
            cloudProvider.shutdownInstance( dc );
            }else{
                  this!killVMWhenPossible(vm); // keep calling ...
            }
      }

}

//=============================================================================
// MAIN
//=============================================================================
{
      RPM rpm = new RPM();
      Int i = 0;
      while (i < 100){
            i = i + 1;
            // wait for a random amount of time ...
            Rat delay = randomf()/100;
            duration(delay, delay); // blocks until 'delay' time units have passed
            // send the request to the RPM
            rpm!receive( "#" + toString(i) );
      }
}
```

C. SACO/CoFloCo Resource Analysis

This appendix details (in §C.1) the modifications to the ABS Model (Appendix A) required to obtain results from the SACO/CoFloCo resource analysis tool, together with the complete results (in §C.2), and a translation of the main result into executable code (in §C.3).

C.1. Required Modifications to the ABS Model

As described in §4 in the main text, the SACO resource analysis tool was applied to determine the number of computational steps entailed by the **`createNextGeneration`** method, with the invocation hierarchy shown in Fig. C.1-*a*:

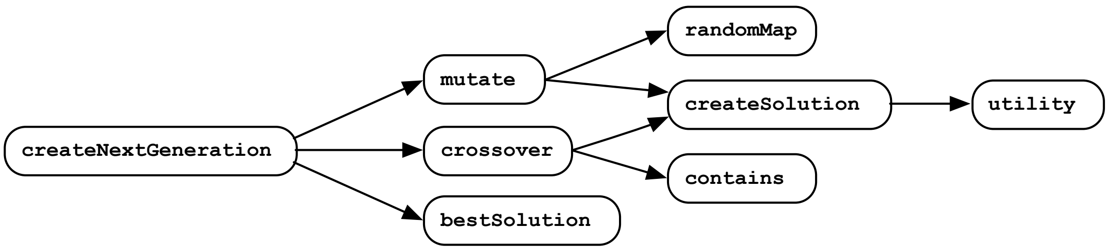

Figure C.1-a: Call hierarchy for the **`createNextGeneration(..)`** *method*

Within this hierarchy, the **`createSolution`** method proved to be a problem - and it was necessary to modify its implementation in order to get results from the SACO tool. The basic form of the *original* implementation (see Appendix A for the full version) is as follows:

```
Solution createSolution(Problem problem, List<Pair<RequestId,Int>> maps){
    //
    List<Request> rejected = Nil;
    Map<VMId, Pair<VMInfo, List<Request>>> assigned = EmptyMap;
    Rat utility = 0;
    Int n = length(maps);
    Int i = 0;
    while (i < n){
        Pair<RequestId,Int> map = nth(maps, i);
        /*
           BUILD THE 'rejected' AND 'assigned' collections
           includes the following call:
           "assigned = put(assigned, vm_id, Pair(vm_info, requests));"
        */
    }
    Set<VMId> vm_ids = keys(assigned);
    while (hasNext(vm_ids)){
        VMId vm_id = take(vm_ids);
        /*
           CALCULATE THE 'utility' OF EACH ASSIGNMENT
           includes the following call:
           "assigned = put(assigned, vm_id, Pair(vm_info, accepted));"
        */
        vm_ids = remove(vm_ids, vm_id);
    }
    return Solution(problem, rejected, assigned, Price(utility), maps);
}
```

The SACO tool stumbled over two features of the above code:

- The first is the use of the *built-in* function **`put(..)`** for building the **`assigned`** map (used within both **`while`** loops). To resolve this we needed to replace calls to **`put`** with calls to a new *user-defined* function, **`put_easy`**, with the following definition (added to the **`FUNCTIONS`** section of the original model in Appendix A):

```
def Map<A, B> put_easy<A,B>(Map<A, B> map, A key,B value)=
          insert( removeKey(map,key) ,Pair(key,value));
```

- The second issue is the use of the **`Set<VMId>`** construct for controlling the second **`while`** loop (it seems the SACO tool has problems interpreting the **`hasNext(vm_ids)`** termination condition). To resolve this we replaced the **`Set`** with a **`List`** (which is populated during the first loop).

The following code snippet shows the final form of the *revised* method (changes are indicated by the **`****`** comments):

```
Solution createSolution(Problem problem, List<Pair<RequestId,Int>> maps){
    //
    List<Request> rejected = Nil;
    Map<VMId, Pair<VMInfo, List<Request>>> assigned = EmptyMap;
    Rat utility = 0;
    Int n = length(maps);
    Int i = 0;
    List<VMId> vm_ids = Nil                            // added a new List // ****
    while (i < n){
        Pair<RequestId,Int> map = nth(maps, i);
        /*
           BUILD THE 'rejected' AND 'assigned' AND 'vm_ids' collections
           includes the following calls:
           "assigned = put_easy(assigned, vm_id, Pair(vm_info, requests));"  // ****
           "vm_ids = Cons(vm_id,vm_ids);"          // populate the new list // ****
        */
    }
                                                      // removed the old Set // ****
    while (vm_ids != Nil){                                                 // ****
        VMId vm_id = head(vm_ids);                                         // ****
        /*
           CALCULATE THE 'utility' OF EACH ASSIGNMENT
           includes the following call:
           "assigned = put_easy(assigned, vm_id, Pair(vm_info, accepted));"  // ****
        */
        vm_ids = tail(vm_ids);                                             // ****
    }
    return Solution(problem, rejected, assigned, Price(utility), maps);
}
```

With these modifications the SACO/CoFloCo tool gives the results shown in the next section.

C.2. Complete Results

This section presents the complete results from the SACO resource analysis for all the methods in the **`Solver`** class' **`createNextGeneration`** call hierarchy (refer to Fig. C.1-*a* above). Settings for the tool were left at default values except for the following:

- *Size abstraction for terms* = **`TypedNorms`**
- *backend* = **`CoFloCo`**

Results are presented in their original form (as output by the tool).

Method: **`bestSolution()`**

Results:

```
***Solver.bestSolution (6 ms)
 UB for 'Solver.bestSolution' = 2

 O(1)
```

Method: **`randomMap(Task task)`**

Results:

```
***Solver.randomMap (147 ms)
 - task_1: size of task wrt. List<VMInfo>

 UB for 'Solver.randomMap'(task_1) = 42*task_1+101

 O(n)
```

Method: **`utility(VMInfo vm_info, List<Request> requests)`**

Results:

```
***Solver.utility (3180 ms)
 - vm_info_1: size of vm_info wrt. Rat
 - vm_info_2: size of vm_info wrt. Bool
 - requests_1: size of requests wrt. Rat
 - requests_2: size of requests wrt. List<Request>

 UB for 'Solver.utility'(vm_info_1,vm_info_2,requests_1,requests_2) = 243*requests_2+268

 O(n)
```

Method: **`contains(List<RequestId> list, RequestId target)`**

Results:

```
***Solver.contains (113 ms)
 - list_1: size of list wrt. Rat
 - list_2: size of list wrt. List<RequestId>
 - target_1: size of target wrt. Rat

 UB for 'Solver.contains'(list_1,list_2,target_1) = 19*list_2+33

 O(n)
```

Method: **`createSolution(Problem problem, List<Pair<RequestId,Int>> maps)`**

Results:

```
***Solver.createSolution (15442 ms)
 - problem_1: size of problem wrt. Rat
 - problem_2: size of problem wrt. List<VMInfo>
 - problem_3: size of problem wrt. Map<RequestId, Task>
 - problem_4: size of problem wrt. Bool
 - maps_1: size of maps wrt. Rat
 - maps_2: size of maps wrt. List<Pair<RequestId, Rat>>
UB for 'Solver.createSolution'(problem_1,problem_2,problem_3,problem_4,maps_1,maps_2) =
max([max([33,33*maps_2+27+9*maps_2*maps_2]),nat(problem_2)*27+74+24*problem_3+max([7467,nat(
problem_2)*672*maps_2+nat(problem_2)*12+32*problem_3+512*problem_3*maps_2+nat(maps_1)*492*ma
ps_2+1840*maps_2+2528*maps_2*maps_2+nat(problem_2+maps_2)*16542+nat(problem_2+maps_2)*37752*
nat(problem_2+maps_2)+nat(2*problem_2+2*maps_2)*1216*nat(problem_2+maps_2)])])+22

 O(n^2)
```

Method: **`mutate(Solution solution)`**

Results:

```
***Solver.mutate (17202 ms)
 - solution_1: size of solution wrt. List<VMInfo>
 - solution_2: size of solution wrt. List<Request>
 - solution_3: size of solution wrt. Map<RequestId, Task>
 - solution_4: size of solution wrt. Rat
 - solution_5: size of solution wrt. Map<VMId, Pair<VMInfo, List<Request>>>
 - solution_6: size of solution wrt. Bool
 - solution_7: size of solution wrt. List<Pair<RequestId, Rat>>
 UB for
'Solver.mutate'(solution_1,solution_2,solution_3,solution_4,solution_5,solution_6,solution_7
) =
max([max([89,128*solution_7+65+28*solution_7*solution_7]),nat(solution_1)*138+369+112*soluti
on_3+max([14856,nat(solution_1)*1440*solution_7+nat(solution_1)*108+128*solution_3+1056*solu
tion_3*solution_7+3960*solution_7+5296*solution_7*solution_7+nat(nat(solution_1)+
-1)*1080*solution_7+nat(solution_1+solution_7)*33084+nat(solution_1+solution_7)*75504*nat(so
lution_1+solution_7)+nat(2*solution_1+2*solution_7)*2432*nat(solution_1+solution_7)])])+45

 O(n^2)
```

Method: **`crossover(Solution s1, Solution s2)`**

Results:

```
***Solver.crossover (21392 ms)
 - s1_1: size of s1 wrt. List<VMInfo>
 - s1_2: size of s1 wrt. List<Request>
 - s1_3: size of s1 wrt. Map<RequestId, Task>
 - s1_4: size of s1 wrt. Rat
 - s1_5: size of s1 wrt. Map<VMId, Pair<VMInfo, List<Request>>>
 - s1_6: size of s1 wrt. Bool
 - s1_7: size of s1 wrt. List<Pair<RequestId, Rat>>
 - s2_1: size of s2 wrt. Rat
 - s2_2: size of s2 wrt. List<Pair<RequestId, Rat>>
 UB for 'Solver.crossover'(s1_1,s1_2,s1_3,s1_4,s1_5,s1_6,s1_7,s2_1,s2_2) =
max([51*s1_7+54+5*s1_7*s1_7+max([578*s2_2+98+27*s2_2*s2_2+
(50*s1_7+50*s2_2)*s2_2,nat(s1_1)*66+53+80*s1_3+108*s1_7+6*s1_7*s1_7+max([nat(s1_1)*696*s1_7+
512*s1_3*s1_7+1780*s1_7+2528*s1_7*s1_7+nat(nat(s1_1)+
-1)*468*s1_7+nat(s1_1+s1_7)*23997+nat(s1_1+s1_7)*44044*nat(s1_1+s1_7)+nat(2*s1_1+2*s1_7)*141
2*nat(s1_1+s1_7),nat(s1_1)*105+306+nat(s1_1)*2088*s2_2+136*s1_3+1536*s1_3*s2_2+6986*s2_2+612
0*s2_2*s2_2+nat(nat(s1_1)+ -1)*1404*s2_2+ (60*s1_7+60*s2_2)+
(1624*s1_7+1624*s2_2)*s2_2+nat(s1_1+s1_7+s2_2)*64536+nat(s1_1+s1_7+s2_2)*125840*nat(s1_1+s1_
7+s2_2)+nat(2*s1_1+2*s1_7+2*s2_2)*4040*nat(s1_1+s1_7+s2_2)])+
(9*s1_7*s1_7+21*s1_7)]),nat(s1_1)*66+219+nat(s1_1)*696*s2_2+80*s1_3+512*s1_3*s2_2+2141*s2_2+
2559*s2_2*s2_2+nat(nat(s1_1)+
-1)*468*s2_2+nat(s1_1+s2_2)*23997+nat(s1_1+s2_2)*44044*nat(s1_1+s2_2)+nat(2*s1_1+2*s2_2)*141
2*nat(s1_1+s2_2)+ (203*s2_2+113+29*s2_2*s2_2)])+52

 O(n^2)
```

Method: **`createNextGeneration(List<Solution> previous,`**
`Int pop_count, Map<VMId, VM> pool)`

Results:

```
***Solver.createNextGeneration (893510 ms)
 - previous_1: size of previous wrt. List<VMInfo>
 - previous_2: size of previous wrt. List<Request>
 - previous_3: size of previous wrt. Map<RequestId, Task>
 - previous_4: size of previous wrt. Rat
 - previous_5: size of previous wrt. Map<VMId, Pair<VMInfo, List<Request>>>
 - previous_6: size of previous wrt. Bool
 - previous_7: size of previous wrt. List<Solution>
 - previous_8: size of previous wrt. List<Pair<RequestId, Rat>>
 - pop_count_1: size of pop_count wrt. Rat
 - pool_1: size of pool wrt. Interface
 - pool_2: size of pool wrt. Map<VMId, Interface>
 UB for
'Solver.createNextGeneration'(previous_1,previous_2,previous_3,previous_4,previous_5,previou
s_6,previous_7,previous_8,pop_count_1,pool_1,pool_2) =
max([16*pool_2,nat(previous_1)*28800*nat(previous_8)*nat(pop_count_1)+nat(previous_1)*58848*
nat(pop_count_1)+nat(previous_3)*21120*nat(previous_8)*nat(pop_count_1)+nat(previous_3)*4761
6*nat(pop_count_1)+nat(previous_7)*1026*nat(pop_count_1)+nat(previous_8)*100160*nat(previous
_8)*nat(pop_count_1)+nat(previous_8)*723360*nat(pop_count_1)+nat(pop_count_1)*1988474+nat(na
t(previous_1)+ -2)*1176*nat(pop_count_1)+nat(nat(previous_1)+
-1)*21600*nat(previous_8)*nat(pop_count_1)+nat(nat(previous_1)+
-1)*46128*nat(pop_count_1)+nat(previous_1+previous_8)*1792064*nat(pop_count_1)+nat(previous_
1+previous_8)*869504*nat(previous_1+previous_8)*nat(pop_count_1)+max([nat(previous_8)*128256
*nat(previous_1+previous_8)*nat(pop_count_1)+nat(nat(previous_1)+
-2)*1176*nat(pop_count_1)+nat(previous_1+previous_8)*1515712*nat(pop_count_1)+nat(previous_1
+previous_8)*1143936*nat(previous_1+previous_8)*nat(pop_count_1),nat(previous_1)*57600*nat(p
revious_8)*nat(pop_count_1)+nat(previous_1)*117456*nat(pop_count_1)+nat(previous_3)*42240*na
t(previous_8)*nat(pop_count_1)+nat(previous_3)*86016*nat(pop_count_1)+nat(previous_7)*3222*n
at(pop_count_1)+nat(previous_8)*214640*nat(previous_8)*nat(pop_count_1)+nat(previous_8)*1261
024*nat(pop_count_1)+nat(previous_8)*628992*nat(previous_1+2*previous_8)*nat(pop_count_1)+na
t(pop_count_1)*4692040+48*pool_2+1260*pool_2*nat(pop_count_1)+nat(nat(previous_1)+
-1)*43200*nat(previous_8)*nat(pop_count_1)+nat(nat(previous_1)+
-1)*86232*nat(pop_count_1)+nat(previous_1+2*previous_8)*9021920*nat(pop_count_1)+nat(previou
s_1+2*previous_8)*4932928*nat(previous_1+2*previous_8)*nat(pop_count_1)])])+25

 O(n^3)
```

C.3. Swift Code for the UB cost for `createNextGeneration`

Following is a translation of the cost expression for the **`createNextGeneration`** method (see §C.2) into executable swift code[18] - used to calculate the cost value for unit parameters reported in the main text (§4.3).

```
func nat(x: Int) -> Int{
    return x >= 0 ? x : 0
}

func UB(A: Int,B: Int,C: Int,D: Int,E: Int,F: Int) -> Int{
    return max(
    16*F
    ,
    nat(A)*28800*nat(D)*nat(E)+nat(A)*58848*nat(E)
        + nat(B)*21120*nat(D)*nat(E)
        + nat(B)*47616*nat(E)
        + nat(C)*1026*nat(E)
        + nat(D)*100160*nat(D)*nat(E)
        + nat(D)*723360*nat(E)
        + nat(E)*1988474
        + nat(nat(A)-2)*1176*nat(E)
        + nat(nat(A)-1)*21600*nat(D)*nat(E)
        + nat(nat(A)-1)*46128*nat(E)
        + nat(A+D)*1792064*nat(E)
        + nat(A+D)*869504*nat(A+D)*nat(E)
        + max(
            nat(D)*128256*nat(A+D)*nat(E)
                + nat(nat(A)-2)*1176*nat(E)
                + nat(A+D)*1515712*nat(E)
                + nat(A+D)*1143936*nat(A+D)*nat(E)
            ,
            nat(A)*57600*nat(D)*nat(E)
                + nat(A)*117456*nat(E)
                + nat(B)*42240*nat(D)*nat(E)
                + nat(B)*86016*nat(E)
                + nat(C)*3222*nat(E)
                + nat(D)*214640*nat(D)*nat(E)
                + nat(D)*1261024*nat(E)
                + nat(D)*628992*nat(A+2*D)*nat(E)
                + nat(E)*4692040+48*F+1260*F*nat(E)
                + nat(nat(A)-1)*43200*nat(D)*nat(E)
                + nat(nat(A)-1)*86232*nat(E)
                + nat(A+2*D)*9021920*nat(E)
                + nat(A+2*D)*4932928*nat(A+2*D)*nat(E)
            )
    )+25
}

let A = 1  // size of previous wrt. List<VMInfo>
let B = 1  // size of previous wrt. Map<RequestId, Task>
let C = 1  // size of previous wrt. List<Solution>
let D = 1  // size of previous wrt. List<Pair<RequestId, Rat>>
let E = 1  // size of pop_count wrt. Rat
let F = 1  // size of pool wrt. Map<VMId, Interface>

UB(A, B:B, C:C, D:D, E:E, F:F)  // result = 89856207
```

[18] The code is intended to be executed within a Swift 'playground'.

D. Java Code Generation: Small-Scale Test

This appendix lists the source ABS model (§D.1), and the Java code (§D.2) automatically generated for this model, for the small-scale code generation test referred to in §5 of the main document.

D.1. ABS Model

Following is the complete ABS model used for the small-scale code generation test (for the curious: this code implements ***Expr.1*** in §4.1 - the formula for the total number of possible 'assignments' given ***n*** requests and ***a*** available VMs).

```
module Perms;

interface Test{
     Int options(Int n, Int a);
}

class Test implements Test{

     Int options(Int n, Int a){
          return opts(n, a+n);
     }

     Int opts(Int n, Int m){
          Int sum = 0;
          if (n == 0 || m == 0){
               sum = 1;
          }else{
               Int x = 0;
               while (x < =n){
                    Int a = this.fact(n);
                    Int b = this.fact(n-x);
                    Int c = this.opts(n-x, m-1);
                    sum = sum + ((a/b) * c);
                    x = x + 1;
               }
          }
          return sum;
     }

     Int fact(Int x){
          Int res = 1;
          if (x > 0){
               Int a = this.fact(x-1);
               res = x * a;
          }
          return res;
     }

}

// MAIN
{
     Test t = new local Test();
     Int res = t.options(2,2);
     println("options(2,2) = " + toString(res));
}
```

The Java code generation tool was applied to the above model and produced the Java code listed in the next section.

D.2. Generated Java Code

Given the ABS model listed in the previous section (saved in file called **`Permutations.abs`**), the Java code generation tool produced four separate java files: **`Permutations.java`**, **`Functions.java`**, **`Test.java`** & **`Main.java`**. The first of these files was empty, the other three are listed below:

Functions.java

```
package perms;

import static abs.api.cwi.Functional.*;

import abs.api.cwi.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.*;
import java.util.concurrent.locks.*;
import java.util.function.*;

public class Functions {

}
```

Test.java

```
package perms;

import static abs.api.cwi.Functional.*;
import static perms.Functions.*;
import abs.api.cwi.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.*;
import java.util.concurrent.locks.*;
import java.util.function.*;

public class Test extends abs.api.cwi.LocalActor implements Test {
        private static final long serialVersionUID = 1L;

        public Integer options(Integer n, Integer a){
              return this.opts(n, (a + n));
              ;
        }

        public Integer opts(Integer n, Integer m) {
              java.lang.Integer sum = 0;
              if (((Objects.equals(n, 0)) || (Objects.equals(m, 0)))) {
                    sum = 1;
              }
              else {
                    java.lang.Integer x = 0;
                    while ((x <= n)) {
                          java.lang.Integer a = this.fact(n);
                          java.lang.Integer b = this.fact((n - x));
                          java.lang.Integer c = this.opts((n - x), (m - 1));
                          sum = (sum + ((a / b) * c));
                          x = (x + 1);
                    }
              }
              return sum;
        }
        public Integer fact(Integer x) {
              java.lang.Integer res = 1;
              if ((x > 0)) {
                    java.lang.Integer a = this.fact((x - 1));
                    res = (x * a);
              }
              return res;
        }

        public Test() {
        }

}
```

Main.java

```
package perms;

import static abs.api.cwi.Functional.*;
import static perms.Functions.*;
import abs.api.cwi.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.*;
import java.util.concurrent.locks.*;
import java.util.function.*;


public class Main {
        private static final long serialVersionUID = 1L;

        public Main(String[] args) throws Exception {

              // Init section: perms
              Test t = new Test();
              java.lang.Integer res = t.options(2, 2);
              System.out.println( ("options(2,2) = " + toString(res)));

              ;
        }

        public static void main(String[] args) throws Exception {
              new Main(args);
        }

}
```

Readers will note that the code contains a couple of small errors:

- The **`Test`** class is defined as implementing a **`Test`** interface, but *i)* no such interface is defined, and *ii)* unlike ABS, classes and interfaces may not share the same name;
- There is a spurious semi-colon (`;`) symbol in the **`options(..)`** method of class **`Test`** - which raises a compile error as 'unreachable code'. There is another spurious semi-colon, but this time innocuous, in the **`Main`** class constructor;

With these errors corrected, the code runs exactly as expected - giving the following, *correct* output:

```
options(2,2) = 29
```