Scalable Computing:
Practice and Experience

Scientific International Journal
for Parallel and Distributed Computing

ISSN: 1895-1767

Universitatea de Vest din Timişoara

Volume 14(1) March 2013

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 14, Number 1, March 2013

TABLE OF CONTENTS

DOI 10.12694/scpe.v14i1.822
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 1, pp. iii–iv. http://www.scpe.org

INTRODUCTION TO THE SPECIAL ISSUE ON SELECTED PAPERS FROM SYNASC 2012 WORKSHOPS

Dear SCPE readers,

One of the current trends in scientific cloud computing is that of migrating HPC to the clouds. This enables researchers to run their complex experiments on a pay-per-use basis without having to buy expensive datacenters. However it also raises some questions concerning the efficiency of cloud HPC and how cloud selection is made. Cloud computing is still in an early stage and solutions to these issues are currently being addressed. While some HPC applications such as MapReduce which does not require extensive intercommunication can be largely deployed on clouds, others like MPI jobs which require high speed networks are less suited for the average cloud infrastructure. To address this problem some cloud providers like Amazon have already begun offering virtualized clusters connected via high speed networks. The increase in the number of choices users must take makes selection a difficult subject especially due to the heterogeneous nature of the offers. To this aim an automatic platform for managing resources over multiple clouds for HPC is needed. These platforms must be able to broker user demands across several providers and meet specific objectives in terms of cost and time.

In this special issue we investigate these aspects and show some of the main contributions presented at dedicated workshops attached to the SYNASC 2013 conference. Namely we selected 4 papers from the *Workshop on HPC Services*, *Workshop on Management of Resources and Services in Cloud and Sky Computing* and *Workshop on Agents for Complex Systems*. A fifth paper presented during the main conference was also published in this number. The first paper argues the necessity of HPC for computational intensive mathematical problems such as that of generating triangulations of PL 3- and 4-manifolds represented by edge-coloured graphs. The second paper addresses the problem of cloud interoperability and proposes a multi-PaaS application management solution. This provides a big step towards offering multi-cloud HPC services for problems such as the one previously mentioned. The third paper deals with benchmarking the efficiency of multi-cloud platforms. This step is crucial especially when using automated cloud platforms as it allows them to optimize their selection strategy based on real information on the underlying systems. The fourth paper presents an automated agent based negotiation model. While the agents are selfish they are also motivated by the necessity to cooperate with others for achieving their objectives. The model is well applicable to a cloud computing scenario where user agents need to negotiate for resources handled by provider agents.

The final paper, which is not part of the special issue but has been presented at the main SYNASC conference track, deals with evolutionary structural testing. Emphasis is put on the study of the "switch" statement in order to automatically generate test data for triggering a particular branch of the program. A new evolutionary structural approach based on a Compact and Minimized Control Flow Graph relying on two different formulas for evaluating the test data performance is proposed.

Marc Eduard Frîncu
West University of Timisoara, Romania

DOI 10.12694/scpe.v14i1.823
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 1, pp. 5–15. http://www.scpe.org

GENERATION OF CATALOGUES OF PL N-MANIFOLDS: COMPUTATIONAL ASPECTS ON HPC SYSTEMS

ALESSANDRO MARANI*, MARZIA RIVI*, AND PAOLA CRISTOFORI†

Abstract. Within mathematical research, Geometric Topology deals with the study of piecewise-linear n-manifolds, i.e. triangulable spaces which appear locally as the n-dimensional Euclidean space. This paper reports on the computational aspects of an algorithm for generating triangulations of PL 3- and 4-manifolds represented by edge-coloured graphs. As the number of graph vertices is increased the algorithm becomes computationally expensive very quickly, making it a natural candidate for the usage of HPC resources. We present an optimized, parallel version of the algorithm that is suitable for deployment of multi-core systems. Scalability results are discussed on two different platforms, namely an IBM iDataPlex Linux cluster and the IBM supercomputer BlueGene/Q.

Key words: High Performance Computing, n-manifolds, coloured triangulations, edge-coloured graphs.

AMS subject classifications. 57Q15 - 57M15 - 68W10.

1. Introduction.

Geometric Topology deals with piecewise-linear (PL) n-manifolds [2], i.e. compact topological manifolds for which there is a triangulation such that each point has a neighbourhood which is piecewise-linearly isomorphic (*PL-homeomorphic*) to an affine n-simplex. Catalogues of triangulations of PL n-manifolds are valuable sources of data: they yield examples to test conjectures, calculate invariants and make comparisons; they can offer insight into the structural properties of the represented manifolds and may suggest ideas for further theoretical investigation.

In particular, since each compact (topological) 3-manifold admits a PL structure and any two PL structures on the same topological 3-manifold are equivalent (i.e. PL-homeomorphic, see [2]), the study of triangulations of PL 3-manifolds is naturally related to the problem of classification, which is still one of the main topics of 3-dimensional topology. The possibility of representing manifolds by combinatorial structures, together with recent advances in computing power, enabled topologists to construct exhaustive tables of small (i.e. obtained by a small number of simplices) 3-manifolds based on different representation methods. In the closed case (i.e. compact and without boundary), catalogues have already been produced and analysed by many authors [8, 19, 20], with a particular focus on combinatorial properties of minimal triangulations.

On the other hand, the problem of classification in dimension four must take into account that a topological 4-manifold not always admits PL structures or may admit non-equivalent ones. For example, although there exists a classification of simply-connected topological 4-manifolds, long established by Freedman [7], the study of (PL) equivalence classes of such structures, especially with regard to their minimal representatives, is an interesting and still open subject of research. Several examples of different PL 4-manifolds triangulating the same topological 4-manifold have recently been presented and the subject is being continuously updated (see for example [9]), but no exhaustive catalogue is available, yet.

Crystallization theory is a representation method for PL n-manifolds by means of a particular class of edge-coloured graphs (*gems*, i.e. Graphs Encoding Manifolds), which are dual 1-skeletons of vertex-coloured (pseudo)triangulations. Topological properties of a manifold are thus encoded in the combinatorial structure of its crystallizations, which are particularly suitable for computer manipulation. Furthermore, crystallization theory allows to develop an algorithmic approach to the generation of censuses of triangulations of compact PL n-manifolds represented by edge-coloured graphs. Unfortunately, the scope of tables of triangulations of n-manifolds is limited by the significant amounts of time required to generate them: in general, a census of triangulations (coloured triangulations are no exception) formed by t n-simplices requires computing time at least exponential in t.

The problem of reducing the generation time can be faced from a topological point of view by excluding some typical graph configurations, which can be removed without affecting the list of represented manifolds (as it happens with dipoles and ρ-pairs, see Sect. 2) or which forbid the graph to represent a manifold, such as those

*Department of SuperComputing Applications and Innovation, CINECA, via Magnanelli 6/3, 40033 Casalecchio di Reno, Bologna, Italy
†Department of Computer, Mathematical and Physical Sciences, University of Modena and Reggio Emilia, via Campi 213/B, 41125 Modena, Italy

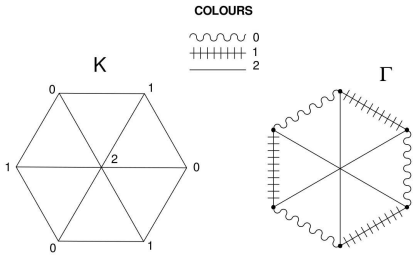


FIGURE 2.1. *Representation of a 2-dimensional torus by a triangulation and its corresponding edge-coloured graph.*

not satisfying Eqs. 3.2 and 3.3 in Sect. 3. As a consequence, positive effects on the generating algorithm come from a sort of "branch and bound" technique. Another direction of improvement relies on the parallelization of the algorithm in order to exploit high performance computing (HPC) infrastructures, which is the focus of the work presented in this paper.

The usage of supercomputing resources has allowed the generation of catalogues of 3-manifold crystallizations with up to 32 vertices, which have already been completely classified up to 30 vertices (both for orientable [13] and non-orientable [14] cases), and the generation of catalogues of 4-manifold crystallizations up to 20 vertices, for which some classification results have already been obtained from their analysis, as reported in [16].

This paper discusses the strategy adopted to optimize and parallelize the generation algorithm and shows performance results on different architectures for the 3- and 4-dimensional cases. In Sect. 2 we introduce the concepts and summarise results from crystallization theory which are required in the development of algorithmic procedures for the generation of PL n-manifolds censuses. A sketch of the generation algorithm is provided in Sect. 3, where specific conditions are detailed for dimensions 3 and 4. Section 4 contains a description of the parallelization strategy implemented, while Sect. 5 shows scalability results on two specific HPC platforms.

2. Representation of PL-manifolds by edge-coloured graphs. For basic PL-topology, topology of 3- and 4-manifolds and elementary notions about graphs, we refer to [1, 3, 4, 6]. For surveys about crystallization theory see [10, 11, 12]. All manifolds are assumed to be closed and connected, unless explicitly mentioned.

A *coloured triangulation* of a PL n-manifold M is a triangulation of M by means of a pseudo-complex whose vertices are labelled by the integers $\{0, \dots, n\}$, so that vertices of the same simplex have different labels. The dual 1-skeleton of a coloured triangulation K of M is a (multi)graph $\Gamma = (V(\Gamma), E(\Gamma))$ whose edges inherit a coloration from K: an edge e of Γ is coloured c if and only if c is the missing colour in the vertices of the $(n-1)$-simplex of K dual to e. In this case, we say that Γ *represents* M or is a *gem* (Graph Encoding Manifold) of M. It is easy to see that M is orientable if and only if Γ is bipartite.

Note that, as a result of the above construction, the coloration of the elements of $E(\Gamma)$ is injective on each pair of adjacent edges. Any regular graph of degree $n+1$ equipped with such an edge-coloration is called an $(n+1)$-coloured graph (without boundary). The elements of the set $\Delta_n = \{0, 1, \dots, n\}$ are called *colours*; moreover, for each $i \in \Delta_n$, we denote by Γ_i the n-coloured graph obtained from Γ by deleting all edges coloured by i. Given an $(n+1)$-coloured graph we can always construct a coloured pseudo-complex $K(\Gamma)$ by taking an n-simplex for each vertex of Γ, by colouring its vertices by Δ_n and, for each $i \in \Delta_n$, by identifying the $(n-1)$-faces of two n-simplices σ and σ' opposite to their i-coloured vertices if and only if the corresponding vertices of Γ are i-adjacent (see Fig. 2.1 for an example).

The above constructions can be generalised in order to take into account coloured triangulations of manifolds with non-empty boundary. In this case the dual 1-skeletons with the inherited edge-coloration miss some n-coloured edges and are called $(n+1)$-coloured graphs *with boundary*. If Γ is such a graph, its *boundary graph* $\partial\Gamma$ is an n-coloured graph (without boundary) whose associated pseudo-complex is the boundary of $K(\Gamma)$.

Most of the following definitions and results can be easily generalised to the boundary case but, for sake of simplicity, we will restrict them to the closed one and, except when explicitly pointed out, we will consider only graphs without boundary. We say that a $(n+1)$-coloured graph Γ is *contracted* when its associated

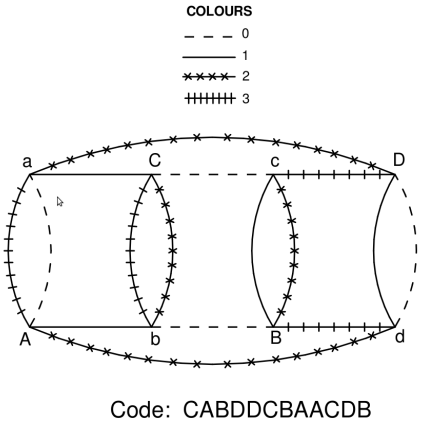


FIGURE 2.2. *A crystallization of the orientable $\mathbb{S}^2$-bundle over $\mathbb{S}^1$ together with its code and the vertex-labelling and coloration associated to it.*

triangulation $K(\Gamma)$ has the minimal number of vertices, i.e. $n+1$. A contracted gem of a PL n-manifold M is called a *crystallization* of M. Contractedness can be checked directly on the coloured graph. In fact duality establishes a bijective correspondence between the connected components of $\Gamma_{\hat{i}}$ and the i-coloured vertices of the associated triangulation for each $i \in \Delta_n$. As a consequence, a gem Γ of a n-manifold M is a crystallization of M if and only if the graph $\Gamma_{\hat{i}}$ is connected for each $i \in \Delta_n$.

Characterising gems of PL n-manifolds among $(n+1)$-coloured graphs is a problem which, by the following result, is strictly related to the recognition of gems of $(n-1)$-spheres.

PROPOSITION 2.1. [10] *An $(n+1)$-coloured graph Γ represents a PL n-manifold iff, for each $i \in \Delta_n$, all connected components of $\Gamma_{\hat{i}}$ represent $(n-1)$-spheres.*

Classical results presented in [10] guarantee that each n-manifold admits a crystallization; obviously, it generally admits many of them and it is a basic problem how to recognise crystallizations (or, more generally, gems) of the same manifold. The easiest case is when two gems are *colour-isomorphic*, i.e. there exists an isomorphism between the graphs, which preserves colours up to a permutation of Δ_n. It is quite trivial to check that two colour-isomorphic gems produce the same polyhedron. The following result assures that colour-isomorphic graphs can be effectively detected by means of a suitably defined numerical *code* [5], which can be directly computed for each of them (see [5, 17] for the related *rooted numbering algorithm* and the example shown in Fig. 2.2).

PROPOSITION 2.2. [17] *Two gems are colour-isomorphic iff their codes coincide.*

Furthermore the code can be used to represent numerically the coloured triangulations in order to manipulate them by computers. Actually it is the most efficient way to represent a coloured graph not only from the point of view of avoiding duplicates of the same triangulation, but also because it contains only the essential information which allow to reconstruct the graph (other kind of representations such as the incidence matrix, in fact, are mainly redundant). Figure 2.2 shows an example of code for the orientable case: the string of the code displays the i-adjacencies of the vertices labelled by small letters for $i \in \{1, \ldots, n\}$. In the non-orientable case the string would be longer since it ought to contain also the n-adjacencies of the capital letters.

The problem of recognising non-colour-isomorphic gems representing the same manifold has been solved too, but not algorithmically. A finite set of moves - the so called dipole moves - has been defined such that two gems represent the same manifold if and only if they can be related by a finite sequence of such moves [18].

DEFINITION 2.3. *An h-dipole $\theta = (x, y)$ in a $(n+1)$-coloured graph Γ is a subgraph consisting of two vertices x and y connected by h edges coloured by $c_1, \ldots, c_h$, such that x and y belong to different connected components of the graph $\Gamma_{\hat{c}_1 \ldots \hat{c}_h}$ obtained by deleting all edges of Γ coloured by $c_1, \ldots, c_h$.*

By deleting the vertices of a h-dipole θ from a $(n+1)$-coloured graph Γ and pasting together the hanging edges according to their colours (see Fig. 2.3), we obtain a new $(n+1)$-coloured graph Γ'. The transformation from

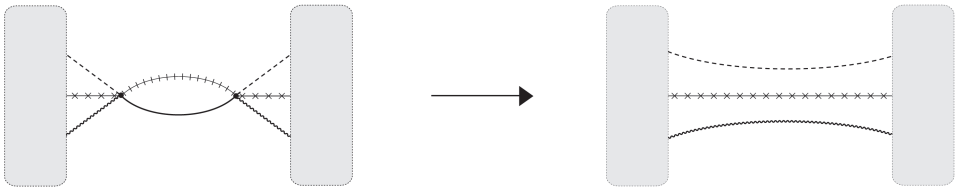


FIGURE 2.3. *Dipole move*

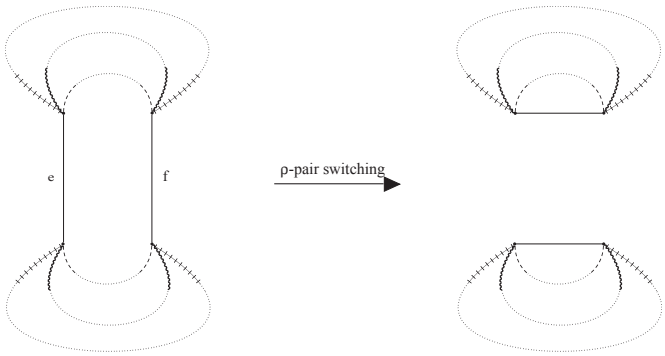


FIGURE 2.4. *Switching of a ρ-pair.*

Γ to Γ' is called the *cancellation of* θ, its inverse the *addition of* θ. Both are called *dipole moves*. Neither cancellations nor additions of a dipole change the represented manifold [18], so dipole moves are an easy tool for manipulating gems without changing the PL-homeomorphism type of the underlying manifold.

Another useful kind of moves relies on the concept of ρ_h-pairs [5, 15]:

DEFINITION 2.4. *A pair* (e, f) *of distinct* i*-coloured edges in a* $(n+1)$*-coloured graph* Γ *is said to form a* ρ_h*-pair iff* e *and* f *belong both to exactly* h *common bi-coloured cycles of* Γ.

ρ-pairs can be eliminated by *switching* (see Fig. 2.4). The following proposition shows the effect of switching on the represented manifold:

PROPOSITION 2.5. [15] *Let* Γ *be a crystallization of a (connected)* n*-manifold* M*,* $n \geqslant 3$ *and let* Γ' *be obtained by switching a* ρ_h*-pair in* Γ.

(a) If $h = n-1$ *then* Γ' *is a crystallization of* M.

(b) If $h = n$ *then* Γ' *is a crystallization of an* n*-manifold* M' *such that* $M \cong M' \# (\mathbb{S}^{n-1} \otimes \mathbb{S}^1)$[1].

A $(n+1)$-coloured graph without ρ_{n-1}- and ρ_n-pairs is called *rigid*. The restriction to the class of rigid crystallizations does not affect the set of represented PL-manifolds, as the following result proves.

PROPOSITION 2.6. [15] *Each closed connected PL* n*-manifold* M *admits a rigid crystallization. Moreover, if* M *is handle-free (i.e. there is no* M' *such that* $M \cong M' \# (\mathbb{S}^{n-1} \otimes \mathbb{S}^1)$*), it admits a rigid crystallization of minimal order.*

3. Generation algorithms. By Proposition 2.1, generation of catalogues of all PL n-manifolds represented by edge-coloured graphs with a fixed number of vertices requires:

- to proceed inductively on dimension n;
- to perform sphere recognition at each step.

In fact, the input data of the generation algorithm in dimension n are the codes of all gems (not necessarily crystallizations) representing $(n-1)$-spheres, which have to be generated previously. To each of these graphs, n-coloured edges are added in all possible ways so as to obtain a crystallization of a closed n-manifold (possible attachments are therefore limited by topological constraints). This algorithm becomes computationally very intensive as the number of vertices of the graphs grows, so it has been necessary to improve its efficiency. The

[1] $\mathbb{S}^{n-1} \otimes \mathbb{S}^1$ denotes the orientable or the non-orientable (according to the orientability of M and M') $\mathbb{S}^{n-1}$-bundle over $\mathbb{S}^1$.

theoretical results presented in Sect. 2 allow to exclude "a priori" a large number of possible configurations. In particular, by Proposition 2.6 and the cited results about dipole moves, we can restrict the catalogues to rigid crystallizations with no dipoles[2].

Let p be a positive integer, we will denote by $\mathcal{C}_n^{(2p)}$ (resp. $\tilde{\mathcal{C}}_n^{(2p)}$) the catalogue of all non-isomorphic rigid bipartite (resp. non-bipartite) crystallizations of PL n-manifolds with $2p$ vertices and lacking in dipoles. Because of contractedness and rigidity, the starting set of the generation algorithm will take into account only connected n-coloured graphs representing the $(n-1)$-sphere with no ρ_{n-1}-pairs. We will denote this set by $\mathcal{S}_n^{(2p)}$. Let $\bar{\Gamma}$ be an $(n+1)$-coloured graph with boundary obtained from an element of $\mathcal{S}_n^{(2p)}$ by addition of n-coloured edges, then $\bar{\Gamma}$ will be kept for further additions if and only if:

(i) it contains no $n-1$ edges with the same endpoints (otherwise there will be ρ-pairs in the resulting completed graph);

(ii) for each $i \in \{0, \dots, n-1\}$, $\bar{\Gamma}_{\hat{i}}$ represents an $(n-1)$-sphere with holes.

The above described restrictions allow to prune the generation tree and succeed in reducing considerably both the computation time and the size of the resulting catalogues, by keeping only essential triangulations. The outline of the algorithm is presented below.

Input: $\mathcal{S}_n^{(2p)}$
new $\mathcal{C}_n^{(2p)}$, $\tilde{\mathcal{C}}_n^{(2p)} = \emptyset$;
for each $\Sigma \in \mathcal{S}_n^{(2p)}$ **do**
  new pair $(\Gamma, \partial\Gamma) = (\Sigma, \Sigma)$;
  set $queue = \emptyset$;
  $(\Gamma, \partial\Gamma) \to queue$;
  while $queue \neq \emptyset$ **do**
    get $queue.firstElement(\bar{\Gamma}, \partial\bar{\Gamma})$;
    if $\partial\bar{\Gamma} = \emptyset$ **then**
      if $\bar{\Gamma}$ is rigid and contracted **then**
        if $\bar{\Gamma}$ is bipartite **then** $\bar{\Gamma} \to \mathcal{C}_n^{(2p)}$;
        else $\bar{\Gamma} \to \tilde{\mathcal{C}}_n^{(2p)}$;
        end if
      end if
    else $v =$ random vertex of $\bar{\Gamma}$;
      new $ver = (w_0, w_1, \dots, w_k)$ array of vertices which can be joined to v;
      for each $w_i \in ver$ **do**
        if v, w_i have not $n-1$ common edges **then**
          new edge $e = n$-coloured edge joining v and w_i;
          new pair $(\Gamma', \partial\Gamma') = (\bar{\Gamma} \cup \{e\}, \partial\bar{\Gamma} \setminus \{v, w_i\})$;
          if Γ' satisfies condition (ii) **then**
            $(\Gamma', \partial\Gamma') \to queue$;
          end if
        end if
      end for
    end if
  end while
end for
Output: $\mathcal{C}_n^{(2p)}$, $\tilde{\mathcal{C}}_n^{(2p)}$

The algorithms for generation of catalogues of triangulations in dimension 3 and 4 have been sufficiently developed from the theoretical point of view and have been implemented in C++ programs. Although they share the approach described above, different combinatorial conditions have to be implemented in order to realise the conditions on Γ' and $\bar{\Gamma}$.

[2]In dimension three, contractedness and rigidity assure that dipoles do not appear. In higher dimensions the absence of dipoles must be checked.

3.1. Dimension three. Dimension three has great advantages from the computational point of view, because:

- the generation of the set $\mathcal{S}_3^{(2p)}$ can be performed by a very efficient recursive algorithm based on a result by Lins [5]: all elements of $\mathcal{S}_3^{(2p)}$ are obtained from those of $\mathcal{S}_3^{(2p-2)}$ by means of the *antifusion of an edge*[3] except for the 1-skeleton of a prism, which appears only when p is even;
- 2-spheres with and without holes can be easily and efficiently recognised by computing their Euler characteristic directly on the representing graphs;
- the zero Euler characteristic identifies closed 3-manifolds.

In particular, the conditions to be checked in the generation algorithm become as follows: a 4-coloured graph Γ *without* boundary is a rigid crystallization of a closed 3-manifold if and only if it has no ρ-pairs and

$$\sum_{i,j\in\Delta_3} g_{ij} - 2p - 4 = 0, \tag{3.1}$$

where g_{ij} is the number of $\{i,j\}$-coloured cycles of Γ; while a 4-coloured graph Γ' *with* boundary satisfies condition (ii) if and only if for each $r \in \{0,1,2\}$, we have

$$2g_{\hat{r}} - {}^{\partial}g_{\hat{r}} = \sum_{i,j\in\Delta_3-\{r\}} \dot{g}_{ij} - m, \tag{3.2}$$

where $2g_{\hat{r}}$ (resp. ${}^{\partial}g_{\hat{r}}$) is the number of connected components (resp. not regular connected components) of $\Gamma'_{\hat{r}}$, m is the number of 3-coloured edges of Γ' and $\dot{g}_{ij}$ is the number of closed $\{i,j\}$-coloured paths of Γ'.

3.2. Dimension four. The recognition of the 3-sphere, which is involved both in the generation of the set $\mathcal{S}_4^{(2p)}$ and in Proposition 2.1, cannot be performed through easy computations. Nevertheless it can be solved for graphs with a low number of vertices by dipole eliminations since, by the 3-dimensional classification results [13], it is known that no rigid crystallization of $\mathbb{S}^3$ exists (different from the "trivial" one of order two) with less than 24 vertices. Furthermore condition (ii) would be very heavy to check, since it implies recognition of 3-spheres with holes. Instead, we use a weaker condition, which is equivalent to require $\Gamma_{\hat{i}}$ to be a manifold (with boundary), i.e. each connected component of $\Gamma_{\hat{i}\hat{j}}$ must represent $\mathbb{S}^2$ possibly with holes for each pair of colours $i,j \in \Delta_3$. This is equivalent to require the following equality to hold:

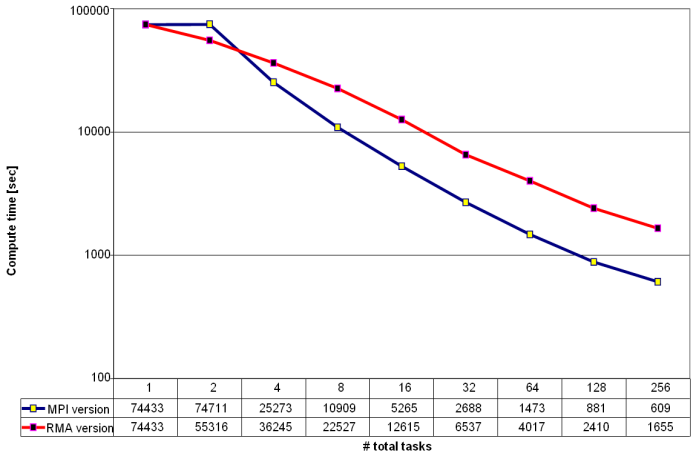
$$\sum_{k,t\in\Delta_4\setminus\{i,j\}} g_{kt} - \frac{\bar{p}}{2} = 2g_{\hat{i}\hat{j}} - \bar{g}_{\hat{i}\hat{j}} \tag{3.3}$$

where g_{kt} is the number of $\{k,t\}$-coloured cycles of Γ, $\bar{p}$ is the number of vertices of Γ lacking in 4-coloured edges, $g_{\hat{i}\hat{j}}$ (resp. $\bar{g}_{\hat{i}\hat{j}}$) is the number of connected components of $\Gamma_{\hat{i}\hat{j}}$ (resp. $\partial\Gamma_{\hat{i}\hat{j}}$).

4. Parallelization strategy. Although theoretical optimizations have been introduced in order to reduce the computational cost for generating a catalogue of crystallizations with a fixed number of vertices, the computation becomes more and more intensive as the number of vertices increases, because of the combinatorial procedure on which the algorithm relies and the large number of input spheres that are processed. Therefore, a parallel version of both the 3- and the 4-manifold generation algorithms described in Sect. 3 have been implemented by exploiting the Message Passing Interface (MPI) paradigm [21].

Parallelization is quite straightforward, as the data can be distributed among several tasks that can work independently from each other. Since the number and the computation time of each crystallization generated by a single sphere can vary significantly depending on the sphere processed, the workload may not be well balanced if each task processes the same number of input spheres. For this reason the work has been distributed among tasks according to a master-slave structure, where the master reads all the sphere codes from the input file and distributes a sphere at a time to slaves. Those slave processes will generate all the possible crystallizations associated to the sphere they received; when a slave has ended the processing of a sphere, it saves the results on its own memory and asks the master for a new one. Master process either provides a new sphere, if some of them have not been processed yet, or collects all the crystallizations produced by the slave task. Note that starting

[3] This operation is the inverse of the cancellation of an edge, which occurs in the same way as for a 1-dipole.



	1	2	4	8	16	32	64	128	256
MPI version	74433	74711	25273	10909	5265	2688	1473	881	609
RMA version	74433	55316	36245	22527	12615	6537	4017	2410	1655

FIGURE 4.1. *Computing times on a Linux cluster to generate 3-dimensional PL manifolds represented by crystallizations with 32 vertices: balanced distribution of spheres by the master versus a RMA approach. The RMA version is less performant than MPI because of the poor implementation of remote memory access on this platform.*

from different spheres we can obtain colour-isomorphic crystallizations, i.e. with the same code, therefore each slave stores crystallization codes in two *set* containers (for orientable and non-orientable manifolds respectively) of the STL library [23] so that results are not replicated.

This strategy implies that the master process does not contribute to the computation, because it must be ready to provide new data as soon as it is required by a slave. We also investigated an MPI-2 Remote Memory Access (RMA) approach which allows the master to process some spheres after having read and stored the input data in a memory window. In fact, each process can get from that window its input data by itself (*one-sided communication*, where memory windows access is regulated by a synchronisation command). In this case another window is required to store a counter keeping track of the already distributed spheres. Such a solution, tested for dimension 3, turned out to be not very efficient, because its performance depends on the system used. In particular its efficiency is guaranteed only on systems with hardware support for remote-memory access, such as SGI Altix or Sun Fire [25]. Whereas it shows poor performance on IBM SMP systems like the one we used for our tests, because the MPI-2 implementation is not optimized for RMA operations.

See Fig. 4.1 for a comparison of the computing times between the original, referred to as MPI, and the RMA approach for generating 3-manifold crystallizations with 32 vertices. Moreover, the contribution of the master to the computation has a negligible impact on the performance as the number of processors increases. Therefore we decided to keep the original strategy and implement it also for the 4-dimensional case.

At the end of the overall computation, the master will have collected all the results produced by each slave process. It will then write the results in two output files: one containing a list of triangulations of orientable manifolds (i.e. bipartite crystallizations), the other for non-orientable ones (as of now, this is true only for the case of the 3-dimensional manifold generation program: in dimension four, because of the very small number of non-orientable manifolds discovered, all the manifolds are still written in a single catalogue). Output produced by each slave is collected only at the end of the overall computation, in order to reduce communications with the master. However, when the number of crystallizations produced by each slave is very large (e.g. in dimension 4), the collection of results by the master can become a serious bottleneck. This can be avoided by letting each slave process write its own output and eventually post-processing all the files in a second step in order to merge them into only two. Moreover, as the number of input spheres increases, each slave may have to deal with a total number of output crystallizations that is too large for its local memory. Notice also that some processes can store isomorphic, and therefore redundant, codes on their local memory, but they cannot find it out since they do not communicate. This translates into a troublesome waste of memory space. Therefore a further small change, required for generating crystallizations of PL 4-manifolds with more than 18 vertices, has been implemented: allow the generation program to process a subset of the input data, so that the list of input spheres can be split in several chunks and each of them can be processed one at a time. The last two modifications discussed (removing the final communication bottleneck and splitting the input processing) have

$2p$	2	8	12	14	16	18	20	22	24	26	28	30	32
$\mathcal{S}_3^{(2p)}$	1	1	1	1	2	2	8	8	32	57	185	466	1543
$\mathcal{C}_3^{(2p)}$	1	1	1	1	3	4	23	44	262	1252	7760	56912	444102
$\tilde{\mathcal{C}}_3^{(2p)}$	0	0	0	1	1	1	9	12	88	480	2790	21804	170367

TABLE 5.1

Number of input gems and corresponding crystallizations generated in dimension 3 up to 32 vertices. Some values, i.e. $2p = 4, 6, 10$*, are missing, since in these cases there are no input 2-spheres and consequently no crystallizations.*

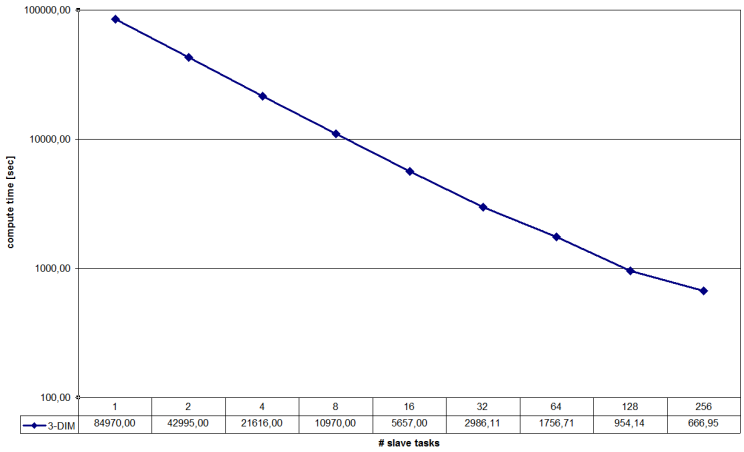


FIGURE 5.1. *Scaling of the computing time with the number of slaves on PLX for the generation of 3-manifold crystallizations with 30 vertices.*

been implemented only on the 4-dimensional generation algorithm: as for dimension three, the number of input and output codes is still small enough to be handled without such changes.

5. Performance results. Some benchmark tests have been made in order to evaluate, for both programs, the scalability with the number of processors on two different systems. The first one is a 3288 cores Linux Infiniband cluster IBM iDataPlex DX360M3 (referred to as PLX), made of 274 IBM X360M2 12-way computing nodes with 2 Intel Xeon Westmere 6-core E5645 2.4GHz processors and a memory of 48GB per node. The second one is an IBM BlueGene/Q system (referred to as FERMI) made of 10240 computing nodes with a chip of 16 IBM PowerA2 1.6 GHz cores and 16GB of memory each.

Performance of the code implementing the generation of triangulations of 3-dimensional PL manifolds has been tested using as an input the catalogue of gems with 30 vertices representing the 2-sphere. In this case we have 466 input 3-coloured graphs producing 78716 crystallizations (see Table 5.1 for a list of the number of input gems representing the 2-sphere and crystallizations produced for increasing numbers $2p$ of vertices). The computation for our test is intensive enough to allow us to complete a scalability test on the PLX system, exploring the range between 1 and 256 slave processes. Computing times plotted in Fig. 5.1 show a linear scalability until the number of slaves becomes greater than 128, because of the increasing communication overhead with the master.

In dimension 4, as the number of input spheres and crystallizations generated is very large even for a low number of vertices (see Table 5.2), the code is suitable to run on massively parallel architectures such as the BlueGene/Q. We considered the case of the 16 vertices catalogue in order to compare performances observed on this platform with the ones on PLX. Figure 5.2 shows scalability with the number of slaves ranging from 64 to 512, both on the PLX and the FERMI systems. As we expected, FERMI computing times are worst than PLX ones because the BlueGene cores are less powerful. For the same reason scalability on FERMI is better than on PLX because slaves' requests for new input data should be more distributed over the time as the processing time of each input sphere is longer, thus avoiding the communication bottleneck.

In order to explore scalability on FERMI over a very large range of processors (from 1024 to 16384), we considered the generation of the 18 vertices catalogue and measured the computing time with respect to the

$2p$	2	8	10	12	14	16	18	20
$\mathcal{S}_4^{(2p)}$	1	9	39	400	5255	95870	1994952	45654630
$\mathcal{C}_4^{(2p)}$	1	1	0	0	1109	4512	44803	47623129

TABLE 5.2

Number of input gems and corresponding crystallizations generated in dimension 4 up to 20 vertices. Some values, i.e. $2p = 4, 6$*, are missing, since in these cases there are no input 3-spheres and consequently no crystallizations.*

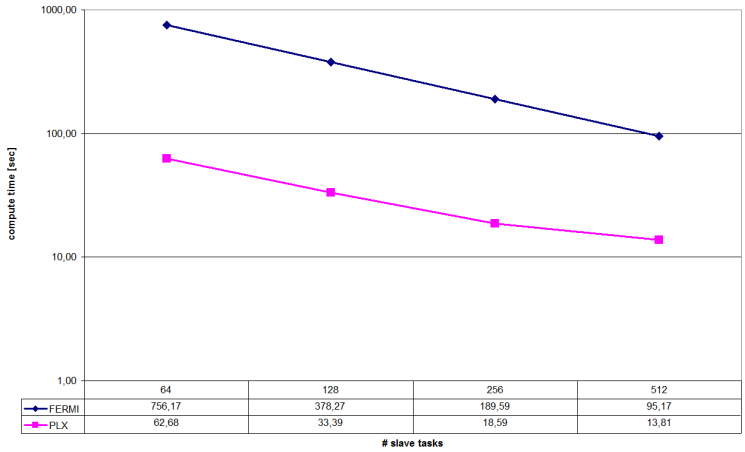
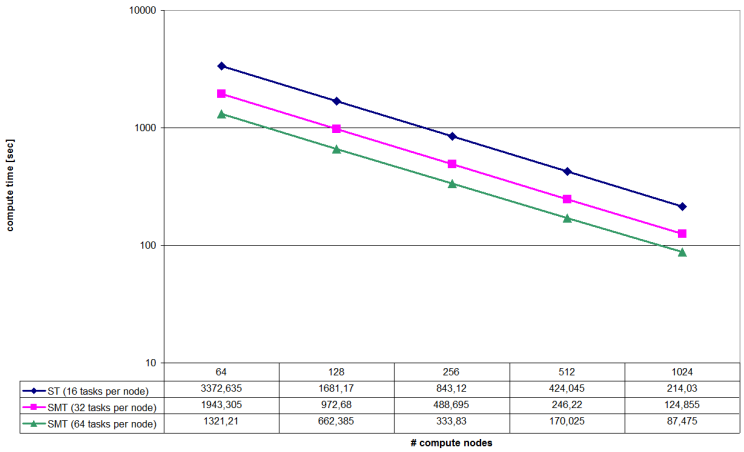


FIGURE 5.2. *Scaling of the computing time with the number of slaves for the generation of 4-manifold crystallizations with 16 vertices. Comparison between FERMI and PLX timings.*

number of total MPI tasks, instead of slave processes. This has been decided due to the peculiar configuration of the system [24], where access to the compute nodes is mediated by the access to specific nodes with the task of dealing with I/O operations ("I/O nodes"). More specifically, in order to let your simulation run on FERMI's compute nodes, an I/O node needs also to be allocated, and since there is one of such nodes every 64 or 128 compute nodes, each resource allocation needs to require (a multiple of) such number of compute nodes. FERMI system is made of 10 racks (each containing 16K cores): 2 of them have 16 I/O nodes each, implying a minimum job allocation of 64 nodes (1024 cores); the other racks have 8 I/O nodes each, implying a minimum job allocation of 128 nodes (2048 cores). PowerA2 cores can schedule for execution two or four threads in the same clock cycle, therefore it is possible to use a single core as two (resp. four) virtual CPUs. This technique is called Simultaneous Multi-Threading (SMT), to distinguish it from the standard Single Thread (ST). Figure 5.3 shows a linear scalability both for the ST and the SMT modes. Deploying SMT improves the performance by reducing the computing time about 42% and 60% by using two and four virtual CPUs respectively. Communication does not affect scalability even for a large number of cores, not only because processors are less powerful and the processing of each input sphere is more intensive, but also because in dimension 4 the output results are not collected by the master. On the other hand, I/O performance may degrade (see Fig. 5.4 as an example) since the I/O nodes are the only ones able to interact with the file systems and each slave writes its own file.

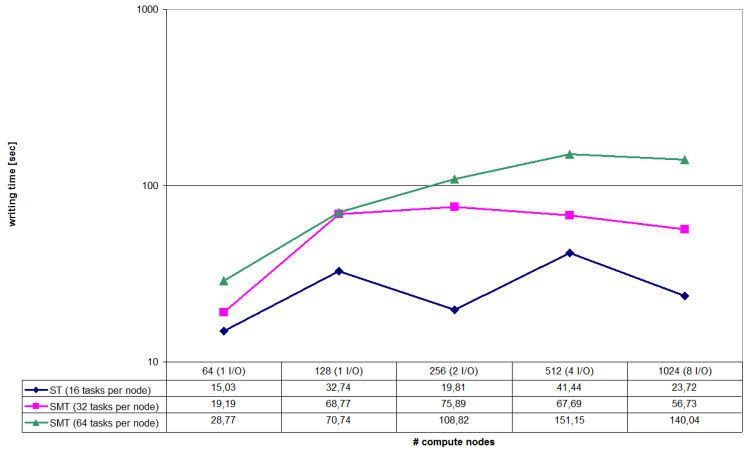
As the number N of input data will increase (with the number of vertices) or scalability will start to degrade (for an increasing number of slaves), a further improvement for the program will be letting the master distribute chunks of a fixed number of input spheres, instead of only one at a time: this should reduce communications with the master and avoid a possible bottleneck. This solution will preserve the workload balance by taking the chunk size very low with respect to N.

6. Conclusions and future developments. In this paper we have described an algorithm for generating 3- and 4-dimensional PL manifolds represented by edge-coloured graphs with a fixed number of vertices. The combinatorial nature of the algorithm has allowed to parallelize it according to a simple master-slave procedure by using the MPI paradigm. However, since the computing time of each slave is dependent on the input data, a suitable data distribution strategy has been adopted in order to have a balanced workload and let the code be able to scale over a very large number of processors. We have discussed scalability results on different high



	64	128	256	512	1024
ST (16 tasks per node)	3372,635	1681,17	843,12	424,045	214,03
SMT (32 tasks per node)	1943,305	972,68	488,695	246,22	124,855
SMT (64 tasks per node)	1321,21	662,385	333,83	170,025	87,475

FIGURE 5.3. *Scaling of the computing time with the number of cores on FERMI for the generation of 4-manifold crystallizations with 18 vertices.*



	64 (1 I/O)	128 (1 I/O)	256 (2 I/O)	512 (4 I/O)	1024 (8 I/O)
ST (16 tasks per node)	15,03	32,74	19,81	41,44	23,72
SMT (32 tasks per node)	19,19	68,77	75,89	67,69	56,73
SMT (64 tasks per node)	28,77	70,74	108,82	151,15	140,04

FIGURE 5.4. *Output times on FERMI for writing all the 4-manifold crystallizations generated with 18 vertices. All MPI tasks write concurrently their own results on separate files. The data size may increase with the number of tasks because some crystallizations may be replicated (in this case we reached a maximum size of 1.1GBytes). There is one I/O node available every 128 compute nodes. The first test with 64 compute nodes has been executed in a special rack where an I/O node is available every 64 compute nodes.*

performance computing architectures: iDataPlex Linux Cluster and BlueGene/Q.

Our future work will investigate further improvements of the code performance, both from a theoretical point of view by searching for other topological results that simplify the computation and from a HPC point of view by introducing a second level of parallelization based on a shared memory paradigm. In the latter case we can accelerate time-consuming parts, such as the computation of the crystallization code, by exploiting the OpenMP [22] Application Program Interface where each slave (MPI task) can run either on a single core in a SMT mode or in a single node if a large number of OpenMP threads is required. Finally we will further explore I/O behaviour when large sized output data is produced concurrently by a large number of processors in order to understand its impact on the total wallclock time, in particular on BlueGene like systems.

We envisage that improvements of the code performances will enable us to extend the existing crystallization catalogues of 3- and 4-manifolds to a higher number of vertices of the representing graphs. In dimension three, we could thus make comparisons with other censuses generated by means of different combinatorial methods. With regard to PL 4-manifolds, we expect that our catalogues would be useful to investigate minimal combinatorial structures and test conjectures about known invariants, as well as giving examples of non-equivalent triangulations of some “interesting” topological manifolds.

Acknowledgments. This work is performed under the auspices of G.N.S.A.G.A. of I.N.d.A.M. (Italy) and financially supported by M.I.U.R. of Italy, University of Modena and Reggio Emilia, funds for selected research topics. We acknowledge the CINECA award under the Italian Supercomputing Resource Allocation (ISCRA) initiative, for the availability of high performance computing resources.

REFERENCES

[1] P. J. Hilton and S. Wylie, *An introduction to algebraic topology - Homology theory*, Cambridge Univ. Press, 1960.
[2] C.Rourke and B. Sanderson, *Introduction to piecewise-linear topology*, Springer Verlag, New York - Heidelberg, 1972.
[3] A. T. White, *Graphs, groups and surfaces*, North Holland, 1973.
[4] J. Hempel, *3-manifolds*, Annals of Math. Studies, 86, Princeton Univ. Press, 1976.
[5] S. Lins, *Gems, computers and attractors for 3-manifolds*, Knots and Everything 5, World Scientific, 1995.
[6] R. Mandelbaum, *Four-dimensional topology: an introduction*, Bull. Amer. Math. Soc. 2(1) (1980), pp. 1–159.
[7] M.H. Freedman, *The topology of four-dimensional manifolds*, J. Differential Geom. 17 (1982), pp. 357–453
[8] B. A. Burton, *Enumeration of non-orientable 3-manifolds using face-paring graphs and union-find*, Discrete Comput. Geom. 38 (2007), pp. 527–571.
[9] A. Ahmedov and B. Doug-Park, *Exotic smooth structures on small* 4*-manifolds with odd signatures*, Invent. Math. 181(3)(2010), pp. 577–603.
[10] M. Ferri, C. Gagliardi and L. Grasselli, *A graph-theoretical representation of PL-manifolds. A survey on crystallizations*, Aequationes Math. 31 (1986), pp. 121–141.
[11] P. Bandieri, M. R. Casali, and C. Gagliardi, *Representing manifolds by crystallization theory: foundations, improvements and related results*, Atti Sem. Mat. Fis. Univ. Modena Suppl. 49 (2001), pp. 283–337.
[12] P. Bandieri, M. R. Casali, P. Cristofori, L. Grasselli, and M. Mulazzani, *Computational aspects of crystallization theory: complexity, catalogues and classification of 3-manifolds*, Atti Sem. Mat. Fis. Univ. Modena, 58 (2011), pp. 11–45.
[13] M. R. Casali and P. Cristofori, *A catalogue of orientable 3-manifolds triangulated by* 30 *coloured tetrahedra*, J. Knot Th. Ram., 17 (2008), pp. 579–599.
[14] P. Bandieri, P. Cristofori, and C. Gagliardi, *Nonorientable 3-manifolds admitting coloured triangulations with at most 30 tetrahedra*, J. Knot Th. Ram., 18 (2009), pp. 381–395.
[15] P. Bandieri and C. Gagliardi, *Rigid gems in dimension n*, Bol. Soc. Mat. Mexicana (3) 18 (2012), pp. 55–67.
[16] M. R. Casali, *Catalogues of PL-manifolds and complexity estimations via crystallization theory*, Oberwolfach Report, 24 (2012), pp. 58–61(DOI: 10.4171/OWR/2012/24).
[17] M. R. Casali and C. Gagliardi, *A code for m-bipartite edge-coloured graphs*, Rend. Ist. Mat. Univ. Trieste 32 suppl.1, (2001), pp. 55–76.
[18] M. Ferri and C. Gagliardi, *Crystallization moves*, Pacific J. Math. 100 (1982), pp. 85–103.
[19] B. Martelli and C. Petronio, *Census 7, Census 8, Census 9, Census 10, Tables of closed orientable irreducible 3-manifolds having complexity c* , $7 \leq c \leq 10$,
http://www.dm.unipi.it/pages/petronio/public_html/files/3D/c9/c9_census.html
[20] S. Matveev, *Recognition and tabulation of three-dimensional manifolds*, Doklady RAS 400(1) (2005) pp. 26–28 (Russian; English trans. in Doklady Mathematics 71 (2005) pp. 20–22).
[21] http://www.mpi-forum.org
[22] http://openmp.org
[23] http://www.sgi.com/tech/stl
[24] http://www.hpc.cineca.it/content/ibm-fermi-user-guide
[25] W. Gropp and R. Thakur, *Revealing the performance of MPI RMA implementations*, Proc. of the 14th European PVM/MPI Users' Group, September 2007, pp. 272–280.

Edited by: Marc Eduard Frîncu
Received: Feb 28, 2013
Accepted: Mar 29, 2013

DOI 10.12694/scpe.v14i1.824
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 1, pp. 17–32. http://www.scpe.org

A USER-CENTRIC MULTI-PAAS APPLICATION MANAGEMENT SOLUTION FOR HYBRID MULTI-CLOUD SCENARIOS

DIMITRIS ZEGINIS[§,¶] FRANCESCO D'ANDRIA[†] STEFANO BOCCONI[‡] JESUS GORRONOGOITIA CRUZ[†] ORIOL COLLELL MARTIN[†] PANAGIOTIS GOUVAS[‖] GIANNIS LEDAKIS[‖] AND KONSTANTINOS A. TARABANIS[§,¶]

Abstract. Cloud Platform as a Service (PaaS) is a rapidly growing IT paradigm which enables software developers to deploy applications without the burden of software platform maintenance. Currently, the PaaS market is dominated by a few providers that promote incompatible standards. This introduces adoption barriers that prevent the interoperability between heterogeneous PaaS offerings, so software developers are not able to manage distributed applications spanning multiple public/private clouds. In this paper we present a multi-PaaS application management solution as a result of the Cloud4SOA European project that addresses these challenges. To clarify this approach a distributed deployment and cloud bursting scenarios are used.

Key words: Cloud computing, Platform as a Service, Multi-Cloud Management, interoperability, portability, user-centric, hybrid Cloud

AMS subject classifications. 68M11, 68P10, 68Q55, 68U35

1. Introduction. Cloud computing is a rapidly growing IT paradigm which transforms the Internet into a global market of on-demand resources. Since 2007, when the term "Cloud computing" started becoming popular, it has managed to evolve into one of the most promising and influential IT trends, occupying the first places in Gartner's list of top strategic technologies for 2013 [12].

Cloud computing first emerged as Infrastructure-as-a-Service (IaaS) boosted by the significance acquired by Amazon Web Services. In parallel, Salesforce was offering a Software-as-a-Service (SaaS) platform (force.com) that included a customization layer and was based on the concepts of Application Service Provision (ASP) and Utility computing dating back to the 70's. Soon, driven by the existence of force.com and the eruption of this market with the entrance of Google's App Engine, the existence of a middle-ware layer in between IaaS and SaaS became clear: Platform-as-a-Service (PaaS).

PaaS [20] enables simplified consumption of Cloud infrastructure and supports Cloud applications. Main consumers of PaaS services are of two user groups: *i)* enterprises with their own internal software development activities and *ii)* Independent Software Vendors (ISV) interested in selling SaaS services on top of a hosted PaaS. The PaaS layer has undertaken a major shift, from mainly accounting on the technologies provided by only two large players as Microsoft (Azure) and Google (App Engine), to the wide variety of capabilities and programming languages found in PaaS offerings today. These new entrants offer a vast range of products, addressing specialized niches as well as broader markets. Often PaaS providers do not even offer their own execution environment, but they rely on IaaS providers for this, becoming effectively a proxy for the consumption of IaaS services. Another trend in the PaaS market is that established IaaS vendors are pushing up the stack to enable programming frameworks on top of their infrastructure, in order to add innovation-driven business models on top of their core IaaS segment. On the other hand, SaaS vendors offer platform tools to tailor their on-demand portfolio with the intention of creating customer loyalty and establishing a wider marketplace around their offering. As a result, there is a very diverse and heterogeneous collection of capabilities offered by PaaS providers.

The heterogeneity among Cloud PaaS providers refers to diversities in supported programming tools (languages, frameworks, runtime environments and databases), in the various types of underlying infrastructure, and even in capabilities available for each PaaS. In such a dynamic and evolving market, application and data portability and semantic interoperability between heterogeneous Cloud PaaS providers become a major issue of concern [23]. The current state of the Cloud PaaS market is therefore far from a global arena open to every player. Giant IT vendors monopolize the market and impose their own standards and formats [10, 11]. Hence, Cloud users, i.e. business and individual developers, are often locked within a specific platform environment

[†]ATOS Spain SA, Av Diagonal, 200, 08018 Barcelona - Spain (francesco.dandria@atos.net, jesus.gorronogoitia@atos.net, oriol.collell.external@atosresearch.eu).

[‡]Cyntelix Corporation BV Baron van Nagellstraat 136, 3771 LL Barneveld - The Netherlands (sbocconi@cyntelix.com).

[§]Centre for Research and Technology Hellas 6th Klm. Charilaou - Thermi Road, 570 01 Thessaloniki - Greece (zeginis@iti.gr, kat@iti.gr).

[¶]Information Systems Lab, University of Macedonia, Thessaloniki, Greece (zeginis@uom.gr, kat@uom.gr).

[‖]Singular Logic, Athens, Greece (pgouvas@gmail.com, g.ledakis@gmail.com).

because it is practically infeasible for them, due to high complexity and switching cost, to move their applications from one platform to another [16]. Reducing the semantic interoperability barriers between different Cloud vendors constitutes the most important step for realizing the global Cloud market vision [18].

Another set of important issues related to the acceptance and spread of Cloud computing is security and privacy. Many organizations with highly sensitive data (e.g. customers' data or health records), because of security or regulatory concerns, cannot move directly to public Clouds. Currently there exist three different deployment models for Cloud computing [20]:

(i) *Public Cloud*. The Cloud infrastructure is provisioned to the general public for open use. It may be owned, managed, and operated by a business, academic, or government organization. It exists on the premises of the Cloud provider.

(ii) *Private Cloud*. The Cloud infrastructure is provisioned locally for exclusive use by a single organization. It is owned, managed, and operated by that same organization and exist within the premises of the organization.

(iii) *Hybrid Cloud*. The Cloud infrastructure is a composition of the two Cloud infrastructures (private, public) that remain unique entities, but are bound together by standardized or proprietary technology enabling data and application portability.

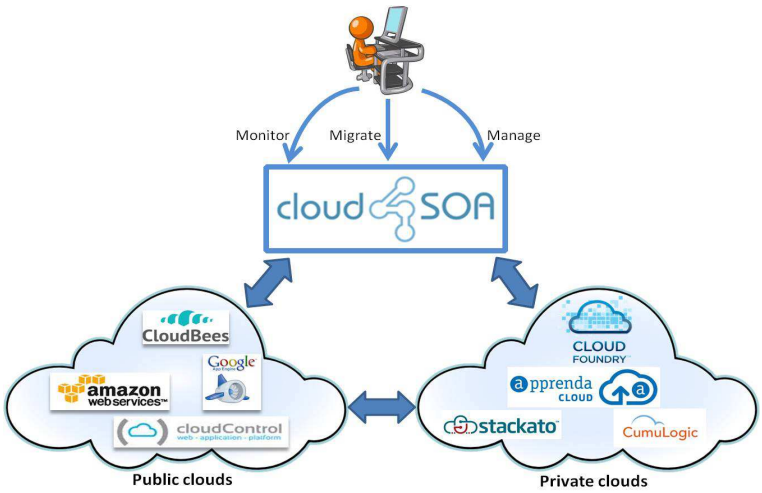


FIGURE 1.1. *The Cloud4SOA: multi-PaaS application management*

The main benefits of a private Cloud with respect to a public one is the possibility to have a more secure environment, since the Cloud is dedicated entirely to the use of the organization avoiding applications from other sources to interfere with them; moreover it offers a localized and completely customizable environment that can be adapted to the organization's needs. On the other hand, a private Cloud has some key drawbacks with respect to a public Cloud, including the ongoing investment of local infrastructure (perhaps a blocking point for SMEs) and scaling limitations, in addition to the lack of the pay-per-use model that helped foster the rise of the Cloud. Organizations that need the benefits of both private and public Clouds can exploit the hybrid model. The hybrid Cloud seeks to alleviate the inherent limitations of purely public and private approaches by combining them into a multi-Cloud that leverages their strengths [25].

In this paper we use a proof-of-concept hybrid multi-Cloud scenario to demonstrate the multi-PaaS application management solution developed by the Cloud4SOA project[1] (Fig. 1.1). Cloud4SOA introduces a broker-based architecture [17, 19] whose main goal is to address and tackle semantic interoperability challenges at the PaaS layer. The architecture is equipped with management and monitoring services providing the appropriate flexibility to handle public, private and hybrid deployment models.

This is an extended version of the work originally presented in [8]. The remainder of this paper is organized as follows. Section 2 presents the related work. Section 3 presents the Cloud4SOA architecture in a nutshell introducing the main layers and the core functionality. Section 4 further describes the layers of the architecture

[1] `http://www.Cloud4soa.eu/`

that enable the multi-PaaS application management and monitoring. Section 5 presents the hybrid usage scenario to clarify the Cloud4SOA functionality. Finally, Sect. 6 concludes this paper and presents future directions.

2. Related work. This section reviews the literature related to multi-PaaS management of applications. Specifically, we reviewed the existing Cloud computing architectures that have been proposed by EU-funded and national R&D projects. We mainly focus on architectures that deal with Cloud computing (semantic) interoperability, even if this is not their primary objective. Moreover, we reviewed brokers and multi-cloud managers which provide an abstract interface that masks the differences among heterogeneous Cloud providers enabling them to interoperate and collaborate.

2.1. R&D projects. The 4CaaSt project [21] envisions an advanced PaaS platform which supports the optimized and elastic hosting of internet-scale applications. This platform facilitates programming of rich applications and enable the creation of a business ecosystem where applications coming from different providers can be tailored to different users, mashed up and traded together. 4CaaSt introduces a broker-based architecture which enables developers to control their applications' lifecycle.

CumuloNimbo [14] aims to provide a scalable PaaS Service which enables secure and un-partitioned data transactions resulting in consistent applications and at the same time ensuring the independent and optimized use of resources at a minimum cost. The CumuloNimbo subsystems is self-healing, automatically repairing themselves in the case of technical failures in order to avoid problems during the service provisioning.

The CONTRAIL project [13] enables enterprises to be both a Cloud provider when its infrastructure is not used at its maximum and a Cloud customer in periods of peak activity. This way, cooperation and resource sharing over Cloud federations is supported by the means of standardized interfaces. To this end, CONTRAIL provides a system in which resources that belong to different operators are integrated into a single homogeneous federated Cloud that users can access seamlessly.

The mOSAIC project [9] aims at the development of a platform that serves as an environment for competition between Cloud providers. mOSAIC focuses both on IaaS and PaaS layers by allowing applications to specify their service requirements (requested by their users) in terms of a Cloud ontology, while the platform, using a brokering mechanism, will perform a search of services in order to find the best matches with application's requirements. The main outcome of the mOSAIC project is a common API for communicating multi-Cloud resources. Unlike Cloud4SOA, the mOSAIC approach is based on the personalization of the IaaS to meet user requirements at the PaaS layer.

The CloudTM project [6] defines a programming paradigm to facilitate the development and administration of Cloud applications. It develops a self-optimized distributed transactional memory middleware that supports interoperability with heterogeneous persistent stores utilizing abstraction techniques. It is composed by two main parts: the Data Platform and the Automatic Manager.

The RESERVOIR project [22] aims at the development of an innovative service-oriented infrastructure that allows the dynamic interoperability of Cloud providers for the reliable delivery of services. To this end, it separates the roles of service provider and infrastructure provider. Service providers interact with the end-users, understand and address their needs. They do not own the computational resources; instead, they lease them from infrastructure providers which interoperate with each other creating a seamlessly infinitive pool of IT resources.

The SLA@SOI [24] developed a service-oriented infrastructure that allows the exchange of IT services with flexibility and well defined conditions and costs. This infrastructure leads towards the evolution of a service-oriented economy with formally specified SLAs and automated transactions between services.

The Cloud@Home is a national project that aims to resolve the problem of incompatibilities among resources composing an interoperable Cloud environment [7]. The Cloud@Home server-side is divided into two main subsystems: the management and the resource. On the one hand, the management subsystem is responsible for enrolling and managing the distributed resources and services providing a unique point of access. On the other hand, the resource subsystem implements the lower level functionalities (e.g. execution, storage).

Table 2.1 summarizes the R&D projects that deal with Cloud computing interoperability. For each project we investigated the targeted Cloud layer (i.e. IaaS, PaaS, SaaS), the usage of semantics as a main feature of the proposed architectures, the resolution of interoperability between diverse Cloud offerings, the portability of applications between Cloud offerings and finally the user-centricity (i.e. the focus on the needs of the end-users).

Based on the results of table 2.1, we see that only mOSAIC and Cloud4SOA follow a user-centric approach

TABLE 2.1
Overview of the R&D projects

Project	Cloud Layer	Semantics	Interoperability	Portability	User-centric
4CaaSt	PaaS	✗	✗	✓	✓
CumuloNimbo	PaaS	✗	✗	✓	✓
CONTRAIL	IaaS/PaaS	✗	✓	✓	✗
mOSAIC	IaaS/PaaS	✓	✓	✓	✓
Cloud-TM	IaaS	✗	✗	✓	✓
RESERVOIR	IaaS	✗	✓	✓	✓
SLA@SOI	IaaS	✓	✗	✓	✓
Cloud@Home	IaaS	✗	✓	✓	✓
Cloud4SOA	PaaS	✓	✓	✓	✓

to achieve semantic interoperability and portability of applications across diverse heterogeneous PaaS offerings. Cloud4SOA is focused only on the PaaS layer, while mOSAIC focuses both at IaaS and PaaS.

2.2. Brokers and multi-Cloud managers. LibCloud[2] supported by the Apache Software Foundation, is a library that can abstract and handle IaaS resources from diverse Cloud providers. Specifically, it provides a unified interface to the computing resources and can manage Cloud servers and storage from different Cloud providers. It also offers security and pricing functionalities.

DeltaCloud[3] abstracts the differences between diverse Clouds offering a single entry point for all the Cloud offering. Specifically, DeltaCloud enables the management of IaaS resources (i.e. compute and storage). The compute resources include the management of the instances, images, firewalls, IP addresses etc. The storage resources include the storage volumes that can be attached to a running instance and blob storage.

RightScale[4] has released the Cloud Management Platform, a Web-based platform which facilitates the deployment and the management of applications spanning multiple Cloud infrastructures - private, public, or hybrid. It delivers complete automation, support for complex deployments, while ensures flexibility, control, and portability.

Enomaly[5] has introduced the Enomaly Elastic Computing Platform (ECP) which empowers service providers (hosting providers, and managed service providers) with a complete Cloud-in-a-box platform that enables them to offer revenue-generating Cloud hosting (infrastructure-on-demand or IaaS) services to their customers.

OpenStack[6] is a free, open-source platform that service providers can use to offer infrastructure services similar to Amazon Web Services' EC2 and S3. It has two main parts: (i) Nova, originally developed by NASA for its computer processing services, and (ii) Swift, the storage service component developed by Rackspace. The OpenStack API can handle compute and storage resources.

OpenNebula[7] is an open-source Cloud computing toolkit for managing heterogeneous distributed data center infrastructures. OpenNebula orchestrates storage, network, virtualization, monitoring, and security technologies to deploy multitier services as virtual machines on distributed infrastructures, combining both datacenter resources and remote Cloud resources, according to allocation policies.

Table 2.2 summarizes the Cloud Brokers which provide an abstract interface that masks the differences among heterogeneous Cloud offerings. For each broker we investigated whether they deal with IaaS resources management (i.e. compute, network, storage), whether they deal with PaaS management (i.e. management of applications) and whether they deal with Cloud monitoring (i.e. monitor resources, applications etc.).

In order to implement a multi-PaaS application management solution the two main features required are the PaaS management (to manage applications at different PaaS) and the Cloud monitoring (to monitor the applications across PaaS). Based on the results of table 2.2 only Cloud4SOA satisfies these two requirements. Moreover Cloud4SOA offers a matchmaking service that allows searching among the existing PaaS offerings those that best match the developer's needs.

[2]http://libCloud.apache.org/
[3]http://deltaCloud.apache.org/
[4]https://www.rightscale.com
[5]http://www.enomaly.com/
[6]http://www.openstack.org/
[7]http://opennebula.org/

TABLE 2.2
Overview of the brokers and multi-cloud managers

Cloud Broker	IaaS management	PaaS management	Cloud monitoring
LibCloud	✓	✗	✗
DeltaCloud	✓	✗	✗
RightScale	✓	✗	✓
Enomaly	✓	✗	✓
OpenStack	✓	✗	✗
OpenNebula	✓	✗	✓
Cloud4SOA	✗	✓	✓

3. Cloud4SOA architecture in a nutshell. The Cloud4SOA reference architecture facilitates Cloud-based application developers in searching for, deploying and governing their business applications on the PaaS offerings that best match their needs. Additionally, switching between PaaS providers is supported in an easy and guided way, without putting the application or the underlying data at risk. This functionality is enabled by semantically interconnecting heterogeneous PaaS offerings. As described in the introduction, heterogeneity refers to the effect of numerous PaaS providers entering the Cloud market with dissimilar architectures, services, models and programming paradigms. Therefore, in the scope of Cloud4SOA, interoperability is defined as the ability of applications and their data to seamlessly be deployed on and/or migrated between Cloud PaaS offerings that are using the same technological background but different data (information) models and Application Programming Interfaces (APIs).

The Cloud4SOA reference architecture has been designed to enable interoperability and cross-platform application management. The Cloud4SOA reference architecture consists of five layers (see Fig. 3.1):

(i) The **Front-end layer** supports the user-centric focus of Cloud4SOA and the easy access of both Cloud-based application developers and Cloud PaaS providers to the provided functionalities.

(ii) The **Semantic layer** is the backbone of the architecture and spans the entire architecture, resolving interoperability conflicts by providing a common basis for publishing and searching different PaaS offerings.

(iii) The **SOA layer** comprises of a toolbox that is accessible through the Service Front-end layer. It implements the core functionalities offered by the Cloud4SOA system (see below).

(iv) The **Governance layer** implements the business-centric focus of Cloud4SOA where PaaS providers and application developers can establish business relationships through SLA agreements and can manage and monitor Cloud-based applications.

(v) The **Repository layer and harmonized API** acts as an intermediary between the platform and the various PaaS offerings by providing a harmonized API which enables the management of applications across different PaaS offerings.

Cloud4SOA's reference architecture provides four main functionalities briefly described in the following:

1. **Semantic Matchmaking** allows to searching among the existing PaaS offerings those that best match the developer's needs. Matchmaking heavily capitalizes on semantic technologies to: *i)* align the user requirements and the PaaS offerings even if they are expressed in different terms and *ii)* resolve the semantic conflicts between diverse PaaS offerings and allow matching of concepts between different PaaS providers that may use different naming or even different measurement units.

2. **Multi-platform application Management** supports the efficient deployment and governance of applications in a PaaS-independent way. The developers can manage the life-cycle of their applications is a homogenized way independently of the specific PaaS offering the application is deployed.

3. **Application Migration** offers the functionality to migrate already deployed applications from one PaaS offering to another. Moving an application between PaaS offerings consists of two main steps: i) moving the application data and ii) moving and redeploying the application itself to the new PaaS offering.

4. **Monitoring** offers a unified platform-independent mechanism, to monitor the health and performance of business-critical applications hosted on multiple Clouds environments in order to ensure that their performance consistently meets expectations defined by the SLA. In order to consider the heterogeneity of different PaaS offering Cloud4SOA provides a monitoring functionality based on unified platform independent metrics.

4. Enabling user-centric multi-PaaS application management and monitoring. This section justifies and presents the Cloud4SOA layers that enable the multi-PaaS application management and monitoring.

A common pitfall in building interfaces is to focus more on the technology rather than on how the user will

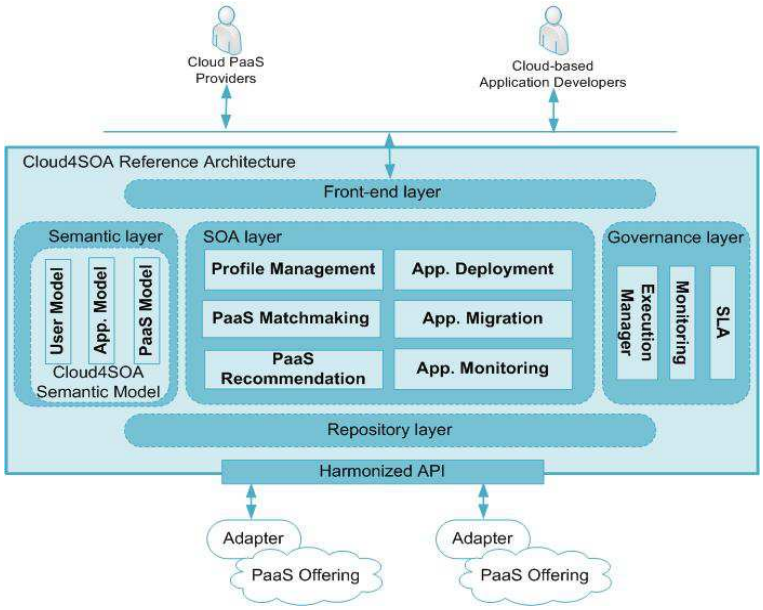


FIGURE 3.1. *The Cloud4SOA Reference Architecture*

make use of this technology. In such cases interfaces mirror what the technology can offer, without providing the user with a usable abstraction above this technology. Complexity and too many details are a common result of this mindset. The design of the Cloud4SOA interface (see Sect. 4.1) considers the user's point of view.

As discussed in the introduction, even when developers use technologies that are standardized (e.g. J2EE), they face a severe lock-in problem while trying to deploy/migrate their applications to a different PaaS provider since the APIs used by each provider differ. Thus, in order to enable multi-PaaS application management and monitoring there is a need for a harmonized API (see Sect. 4.2) which covers functionalities offered by the majority of PaaS providers.

Finally, in a multi-Cloud scenario it is vital to constantly monitor business-critical applications hosted on various Cloud environments, to ensure that their performance consistently meets expectations and that Cloud resources are being effectively used, independent of the technology used to implement such applications. Section 4.3 presents the approach proposed by Cloud4SOA to enable the multi-PaaS application management and monitoring.

4.1. Front-end layer - a user-centric approach. The Front-end layer is not limited to providing a pleasant look for the interface, but also to support the users to understand the data and services presented by the interface and efficiently interact with them. Supporting the user in these two key points is what makes an interface "intelligent". Cloud4SOA tackles these challenges by: *i)* providing a metaphor for the interface the user can understand and is familiar with, *ii)* providing the user with contextual information about the data visualized by the interface and *iii)* providing a dynamic interface easily adapted to user needs.

4.1.1. Dashboard Metaphor. In the scope of user interface design the most problematic issue is modeling the interaction [15]. Designing the user-system interaction is a complex process because of the two facets of human-computer interaction: coping with technical constraints as well as human factors [3]. In general, a model of an interaction can be seen as a way to understand and improve the usability of the interface [2, 5].

In our work we adopt the following definition for interaction model [4]: *"An interaction model is a set of principles, rules and properties that guide the design of an interface. It describes how to combine interaction techniques in a meaningful and consistent way and defines the "look and feel" of the interaction from the user's perspective. Properties of the interaction model can be used to evaluate specific interaction designs."*

Since Cloud4SOA interface needs to support user interaction and asynchronous display of events such as monitoring, the dashboard metaphor was chosen. The dashboard metaphor is something users are familiar with since it is vastly adopted (e.g. car dashboards) and offers the following advantages in the Cloud4SOA case:

(i) Loosely coupled widgets represent single instruments on the dashboard, this enables the addition/removal of functionality by adding/removing widgets.

(ii) Widgets can update their display asynchronously and independently of one another, as instruments on a dashboard.

(iii) Widget can be minimized or moved, allowing the user to specify what information should be always on the dashboard.

(iv) Depending on the device used to access the dashboard, some widgets can be hidden e.g. in case of a mobile phone screen, without the need to scroll.

Another capability of the dashboard is the presentation of **contextual information** based on the semantics used to model developers' applications and Cloud offerings. An immediate use for the User Interface is to display the semantic description of the item. This already provides some context to the user when selecting an item. Semantic modelling can also be used in disambiguating terms. The Semantic layer provides mapping between different terms that can be used by the User Interface to suggest to the user that the term he is using has a particular relation to the term the PaaS provider uses, e.g. that a *dyno* is equivalent to n CPUs.

The testing phase of the dashboard metaphor showed that the interface could result confusing to users if there were too many widgets on the same page. Therefore we provided a simplified and more guided dashboard by grouping functionality. This consisted in defining several "places", or sections of the interface based on particular actions the user wants to perform, such as search for a suitable PaaS offering. Inside each place the dashboard metaphor was still valid, but involving less widgets.

FIGURE 4.1. *The Application Profile editor, one of the places of the Cloud4SOA User Interface.*

A screenshot of the design can be seen in Fig. 4.1, where the Application Profile editor is shown. Here the user can enter the profile describing the application he or she wants to deploy. The layout is organized as follows: *i)* A header contains a navigation tool bar, *ii)* a central body shows the widgets associated to the place and some instructions of the actions that the user can perform in the place and *iii)* a footer displays static information and status communications.

4.1.2. Dynamic Interface Creation. It can be very time-consuming to redesign an interface layout when fields are added, modified or removed from the data schemas to be displayed. Generating dynamically the interface allows accommodating (semi) automatically the layout when data structures change. A dynamical interface can be designed considering two aspects of the problem: *i)* What items (such as Computing resources) and what properties of each items (such as number of CPUs) should be displayed and *ii)* How to display them, i.e. whether to use a drop-down list, sliders or radio buttons.

Both problems can be tackled by exploiting the semantic modeling. At runtime the interface queries for the objects that are modeled and for their properties. Subsequently rules are defined so that different types of visualization are chosen according to the data that needs to be displayed. An example is to use a slider when values are numerical varying over a range.

This has been implemented in Cloud4SOA by defining a domain specific language[8]. This language allows to define how data is represented in the visual form, including field validation and decoration (labels, tooltips and so on). An example for the field *Programming Language* in the Application Profile editor is the following:

```
field(FieldMetadata.create(PROGRAMMING_LANGUAGE)
 .label(''Programming language'')
 .editType(COMBOBOX)
 .tooltip(''Main programming language for the application'')
 .relatedEntityType(COMBO_PROGRAMMING_LANGUAGE)
);
```

The definition of the form's attributes can be retrieved from the semantic model, which leads to a fully model driven semantic repository editor. An example of this can be seen in the *Add component* item in menu of the Application Profile editor (top-center of Fig. 4.1). The *Network*, *Compute*, *HTTP Request Handler* and *Storage* menu items are subclasses of *Hardware Component* in the Semantic model.

Another typical dynamic interface for semantically modelled data is **faceted browsing**. Faceted browsing allows users to filter dynamically the items in a repository which satisfy the conditions posed on the chosen facets. Faceted browsing shows the information in a continuously updated way while the user selects options/criteria. The user is provided with an immediate response while he refines his query results. The advantages are twofold:

(i) Users can start with specifying few parameters, and decide to add more or revise some of the already chosen ones based on the result list.

(ii) Users do not risk wasting time in specifying all parameters of queries yielding an empty result set, since they receive immediate feedback on what parameter caused the query to return no results.

Cloud4SOA dashboard provides a faceted search widget to browse, filter and see details on existing PaaS offerings, based on faceted criteria selected by the user (see Fig. 4.2). Facet types (columns) and their values correspond to PaaS Offering properties (i.e. ratings, software and hardware components, etc.), which are obtained by inspecting, dynamically, the repository of offerings.

The user can select concrete values (criteria) for the facet types displayed by the widget. The matching PaaS offering result set is continuously updated and incompatible facet values filtered off the facet columns in order to avoid further empty searches. Multiple values within the same facet type are logically connected with the OR operator, while facet types are connected with AND operator.

4.2. Repository layer and harmonized API. In order to alleviate the lock-in problem and achieve harmonized interaction with many PaaS offering, two architectural decisions have been taken:

(i) The formulation of one harmonized API which aims to abstract the specificities between various PaaS offerings;

(ii) The introduction of PaaS-specific Adapters to bridge the business logic between the harmonized API and the PaaS offerings.

There are major differences that exist in almost all aspects of PaaS offerings, thus the necessity of an API that is generic enough to readily include a wider variety of situations is evident. The API contains a number of operations that support the management of the Cloud-based applications independent of the specific API of the underlying PaaS offering. An overview of the methods that constitute the API and the supported PaaS offering is depicted at Fig. 4.3. The "Migrate" line indicates the PaaS offerings that are compatible for migration. The

[8] http://www.martinfowler.com/bliki/DomainSpecificLanguage.html

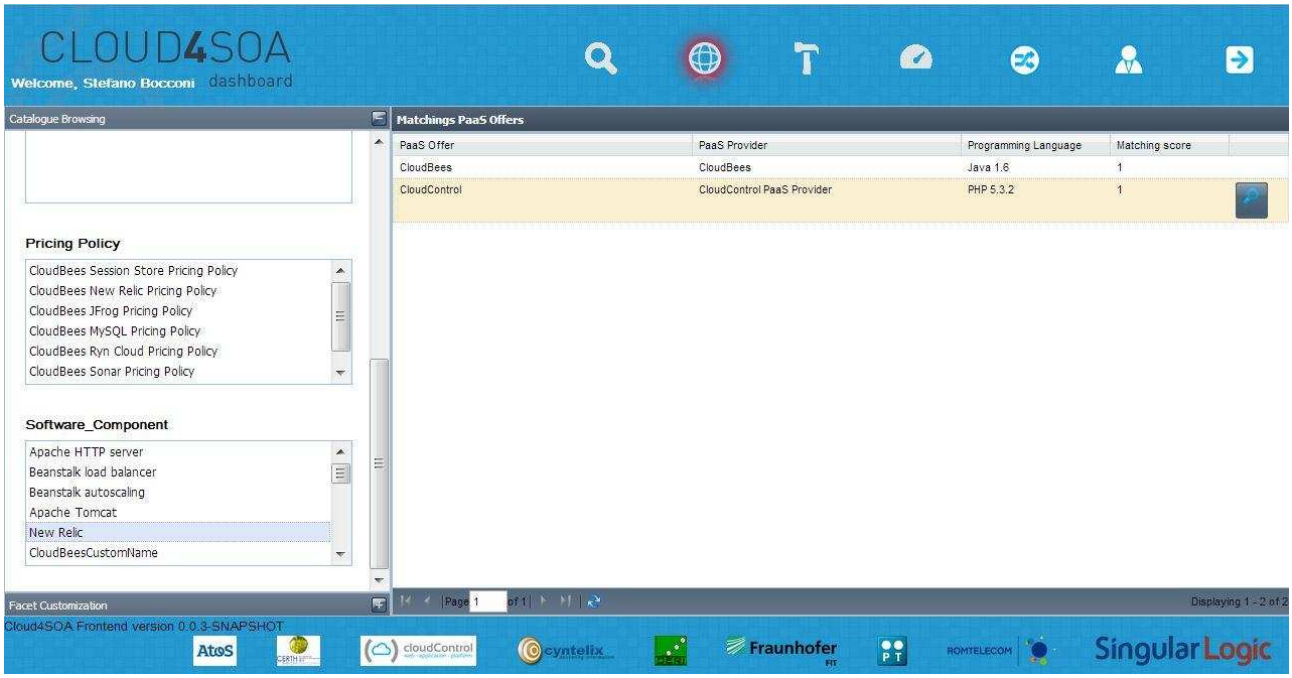


FIGURE 4.2. *Faceted Browsing.*

migration is only possible between the platforms that are mentioned with the same number in this line (e.g. AWS Beanstalk, CloudBees and CloudFoundry are compatible; CloudControl and Heroku are compatible too).

Cloud4SOA Capabilities	AWS Beanstalk	CloudBees RUN@cloud	CloudControl	Heroku	CloudFoundry	OpenShift
Matchmaking:						
The criteria and information of the PaaS within Cloud4SOA's matchmaking search.	✓	✓	✓	✓	✓	✓
Governance						
Deploy	✓	✓	✗	✗	✓	✗
Deploy through GIT	✗	✗	✓	✓	✗	✓
Start	✓	✓	✓	✓	✓	✓
Stop	✓	✓	✓	✓	✓	✓
Delete	✓	✓	✓	✓	✓	✓
Create Database	✓	✓	✓	✓	✓	✓
Delete Database	✓	✓	✓	✓	✓	✓
Import Data to Database*	✓	✓	✓	✓	✓	✓
Export Data from Database*	✓	✓	✓	✓	✓	✓
Monitoring						
get Status	✓	✓	✓	✓	✓	✓
get HttpResponse Time	✓	✓	✓	✓	✓	✓
get ICMP Ping Response Time	✓	✓	✓	✓	✓	✓
Migration						
Migrate**	⇆(1)	⇆(1)	⇆(2)	⇆(2)	⇆(1)	✗
*	Importing and Exporting Data to a database is done by using the database configuration parameters, and is supported to a limited set of database systems					
**	Migration of application and its data between different platforms is supported, but it is only possible between specific platforms that are mentioned with the same number in the table					

FIGURE 4.3. *API methods and PaaS offering compliance*

PaaS-specific Adapters are responsible for bridging the business logic between the introduced harmonized API and the APIs of the PaaS offerings by translating the functions of one to another and vice versa. The bridging of various PaaS platforms results on the creation of an overlay network which is responsible for the information fusion between the applications that are hosted at various distributes PaaS providers and the Cloud4SOA central system. Beyond the information fusion, the aforementioned Adapters provide the functionality of deployment, migration, governance and monitoring of an application.

In order to provide high-level functionalities (e.g. matchmaking) there is a need for persistently storing the semantic profiles of developers, PaaS providers and applications. For this reason, Cloud4SOA uses a repository in order to store both semantic and syntactic data. Moreover, there are additional requirements imposed by security and SLA issues.

Regarding the security issues, each developer has to interact seamlessly with every PaaS offering. In most of the cases the communication between the PaaS and the developer is accomplished through the usage of asymmetric cryptography (public-private key infrastructure). Therefore, a specific amount of information has to be persistently stored in the repository. Indicative information include the developer's public key and specific details per each project (version-control-system info).

Beyond security issues, the SLA management imposes many requirements since every deployed application has to be monitored. Therefore, specific data has to be stored at the repository about the monitoring-jobs, monitoring-statistics, SLA violations etc.

4.3. Governance layer - monitoring and SLA management. Cloud providers typically present very diverse architectures providing diverse resource-level metrics used to provide fine-grained Quality of Service (QoS) guarantees. As a consequence, Cloud consumers are not able to compare offerings they are looking for. In order to consider the heterogeneity of diverse Clouds architectures, Cloud4SOA provides a Service Level Agreement (SLA) management layer based on PaaS monitoring functionality that follows a unified and Cloud technology-agnostic approach. This approach defines specific metrics allowing Cloud consumers to monitor the health and performance of their business-critical applications hosted on multiple Clouds environments (see Fig. 4.4 where the Monitoring widget is shown).

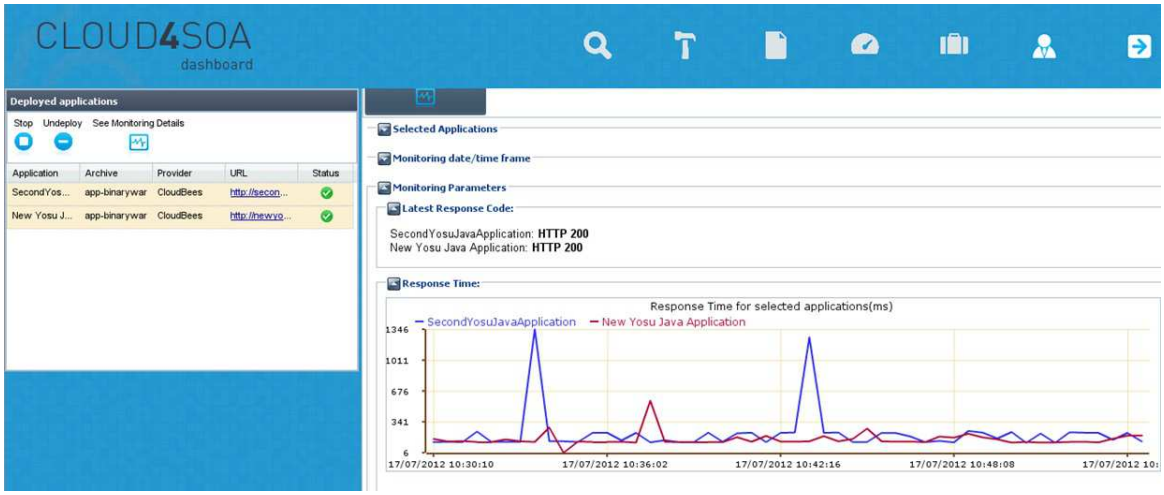


FIGURE 4.4. *Application monitoring*

The monitoring functionality leverages on a range of standardized and unified metrics of different nature (resource/infrastructure level, container level, application level, etc.) that, based on the disparate underlying Cloud providers, allow the runtime monitoring of distributed applications so as to enforce the end-to-end QoS, regardless of where they are deployed across different PaaS.

In the scope of Cloud4SOA several metrics have been defined (Table 4.1) from the Cloud resource as well as the business application perspective, but not all of them have been enforced at runtime since some of them only provide useful information about the status of the application.

In order to provide quality of service guarantees, SLA plays a crucial role. However, in the scope of project, SLA management does not aim at representing a contractual relationship between the cloud consumers and the PaaS providers, nor establishing one. Instead, Cloud4SOA's SLA management describes the functionality of the application that is delivered, the functional and non-functional properties of the negotiated resources, and the duties of each party involved. As a result, its specifies the service level objective (SLO) that must be met in order to fulfill a guarantee as well as policies to pick the right violation level (i.e. warning, medium violation, migration etc.) and consequently suggest the appropriate recovery actions to take.

TABLE 4.1
Cloud PaaS metrics

Metric	Description	App. status	Cloud performance
CPU load	The amount of computational work that the application performs	✓	
Memory load	The amount of memory consumed by the application	✓	
HTTP response code	Includes custom status messages to understand the health of the application, but also the performance of the Cloud	✓	✓
Application and DB Response Time	Time that measures the efficiency and speed with which servers deliver requested web content to end users.	✓	✓
Application container Response Time	The elapsed time between the end of an inquiry or demand on a Cloud system and the beginning of a response.		✓
Cloud Response Time	The time the Cloud needs to process and forward to the application the incoming call.		✓

To deal with a SLA in an automatic way (i.e. to have mechanisms that automate the SLA setting up, monitoring and enforcement) the SLA itself has to be expressed through Cloud4SOA in a formalized way using an SLA specification language. Several languages for SLA specification have been reviewed (e.g. WSOL, WSLA, WSML etc.) but the WS-Agreement specification [1] was selected as the most appropriate:

(i) WS-Agreement offers a protocol to be followed for the negotiation process and a common understanding (i.e. language) of the objects the negotiation is about. Thus, it enables the automatic creation of SLAs.

(ii) The outcome of a successful negotiation with WS-Agreement is a SLA with binding character for both parties to deliver a reliable service to the end-user.

(iii) WS-Agreement is a novel but well-accepted standard for creating and enforcing SLAs in distributed environments as well as monitoring their resources properties.

Cloud4SOA provides a RESTful implementation of the WS-Agreement standard. On top of the implementation the governance layer offers three main functionalities (Fig. 4.5) that enable users negotiate and enforce SLA, as well as recover from a SLA violations:

(i) **SLA Negotiation**. Allows the automatic negotiations on behalf of PaaS providers, based on the semantic description of offerings and the QoS requirements specified by the Application Developer.

(ii) **SLA Enforcement**. Supervise that all the agreements reached in a SLA agreement are respected (i.e. measurements are within the thresholds established in SLA agreement for QoS metrics).

(iii) **SLA violation recovery**. Whenever the execution of the business application does not satisfy the SLA (i.e. breaches of the agreement occurs), the most appropriate recovery action (e.g. warning messages, stop or migration of the application) is suggested based on the policies defined by the software developer.

The SLA violations leveraged by the Cloud4SOA system in order to provide an enhanced rating mechanism which is responsible for rating negatively a PaaS offering whenever a violation occurs.

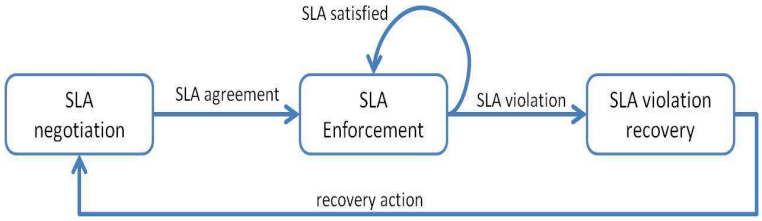


FIGURE 4.5. *SLA negotiation - enforcement - recovery process*

5. A hybrid multi-Cloud scenario. This section presents a hybrid multi-Cloud scenario that shows the way to manage multiple Cloud environments of varying natures (i.e. public, private, hybrid). The hybrid cloud

model is used to manage both public and private clouds using a single platform (i.e. the Cloud4SOA platform). Specifically, we describe the way to integrate a private PaaS into Cloud4SOA and describe a set of scenarios that have been simulated to evaluate the usefulness and usability of the adopted approach. The scenarios contain a distributed deployment into more that one platforms and cloud bursting.

5.1. Integrating a private PaaS to Cloud4SOA. In order to enable hybrid deployment we have installed an instance of CloudFoundry[9], an open-source private PaaS solution, and integrated it with Cloud4SOA. With this integration, any Cloud user can easily manage a hybrid deployment approach using the Cloud4SOA platform. The process for integrating a PaaS provider (Fig. 5.1) is described in detail at the following paragraphs.

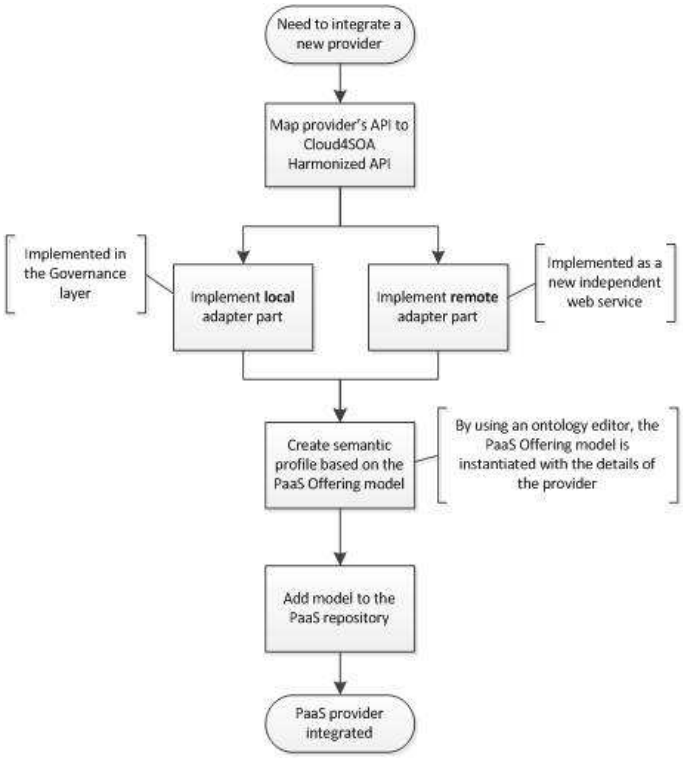


FIGURE 5.1. *Flowchart showing the process followed to integrate a new PaaS provider to Cloud4SOA*

The first step consists on analyzing the API of the provider and designing the necessary wiring to implement each of the methods in the harmonized API (see Fig. 4.3). In the CloudFoundry integration this step was one of the toughest ones most difficult since it is still young and it lacks a extensive set of documentation. This required to a directly examine direct examination of the code of the platform and to get involved in the developer's community in order to resolve all doubts.

Once the API is analyzed, the adapter needs to be implemented. The adapter is divided into two parts: a local one that runs along Cloud4SOA and a remote one that runs in the provider along applications deployed on it. The implementation of the local part is included in the Governance layer of the system while the remote part is an independent component that is automatically deployed to the provider each time an application is deployed. CloudFoundry offers a set of open-source libraries to interact with the API. We used these libraries to implement the adapter since they provide a reliable way to interact with CloudFoundry which contributes to the overall robustness of the adapter.

Once the adapter has been implemented and tested, the semantic profile of the provider has to be created and stored at the semantic repository. The semantic profile enables the description of functionalities offered (e.g. programming languages, software components, hardware components etc) as well as SLAs and Service quality terms.

[9] `http://www.Cloudfoundry.com/`

5.2. The hybrid scenarios. In order to evaluate the adequacy and usefulness of the proposed approach into a hybrid deployment we have identified two scenarios. These scenarios have been simulated using a complex modular framework provided by Portugal Telecom Inovacao (PTIN). The framework is formed by a set of pervasive services which can intelligently take proactive actions to select, rate and deliver suitable content (e.g. photos, videos, etc.) based on user's context information. The framework offers three basic services:

(i) The Context Enabler is a SOAP web service that subscribes and receives context information about a specific user.

(ii) The Group Manager Enabler is a rule engine responsible for identifying, creating and managing groups of consumers based on their context information.

(iii) The Content Selection Enabler is a rule engine that aims at rating and selecting content (e.g. photos, videos, etc.) for users based on their group(s).

Since the framework is based on several autonomous services, it is possible to exploit the distribution of them across several public/private platforms and test their behavior in each environment and, thus, it is adequate to simulate the defined hybrid deployment scenarios, which are described below.

Distributed deployment. In this scenario an application composed by two autonomous parts is distributed between a private Cloud and a public Cloud. On one hand, the part deployed privately may contain sensitive or confidential data that requires to be stored in the same country as the company and secured by using a set of custom policies. On the other hand, the part deployed in the public Cloud may require a higher level of reliability or performance. By using Cloud4SOA, a developer can manage the deployment and governance of both parts from the same place and compare their performance in a graphical way by using the monitoring tools provided by the system.

We have used two of the pervasive services from the PTIN framework to simulate the scenario. First, the Group Manager Enabler is deployed in a public Cloud, CloudBees in this case, since the rule engine can be highly demanding in terms of performance. Then, the Context Enabler is deployed in the private CloudFoundry instance as it can contain sensitive users' information. We assume that the third service (the Content Selection Enabler) is deployed in another PaaS and that the Context Broker runs locally on the PTIN infrastructure.

Bursting on demand. In this scenario all parts of an application are deployed in the same private Cloud and are governed by a SLA between the developer and the provider. After some time, Cloud4SOA detects a violation on an SLA term and triggers a notification about the issue. The developer, then, driven by this warning, decides to burst the critical part of the application to a public Cloud able to fulfill the SLA. By using Cloud4SOA, a developer can be notified about such SLA violations and decide the actions to take. Moreover, if he or she decides to burst a part of an application, he or she can perform the migration from the same Cloud4SOA system. Figure 5.2 shows the scenario in a graphical way.

In this scenario, the two services (the Group Manager Enabler and the Context Enabler) are deployed in the private CloudFoundry[10] instance and after a warning from Cloud4SOA stating that some performance-related SLA terms are being violated for the Group Manager Enabler, it bursts to a public Cloud.

5.3. Discussion and evaluation. An evaluation of the performance and usefulness of Cloud4SOA has been conducted based on the two hybrid scenarios deployed. The evaluation shows that Cloud4SOA can provide benefits to user willing to adopt a hybrid approach, since it allows deploying, governing and monitoring both parts of a hybrid Cloud from the same single place. Moreover, it provides useful information and notifications about the performance of the applications and the fulfillment of SLAs and allows bursting an application in case of SLA violation.

Moreover, we analyzed the overhead imposed by the Cloud4SOA system compared to the usage of the provider's APIs or tools directly when executing operations such as deploy, start, stop or undeploy. Table 5.1 shows the comparison of the time that Cloud4SOA takes to execute some basic deployment and governance operations. The values shown are the average of the results obtained from repeating each operation 10 times. From these results we see that there is an overhead in time imposed by the system, but, in general, it is acceptable taking into account the added-value provided.

Integrating CloudFoundry with Cloud4SOA has also been useful to learn more about the complexity of adding a new provider to the system. The ultimate goal is to enable an easy to follow process that can be performed by providers willing to join Cloud4SOA themselves. The process is quite easy however work is still required to enhance this experience and to reduce the amount of intervention required.

[10]http://www.Cloudfoundry.com/

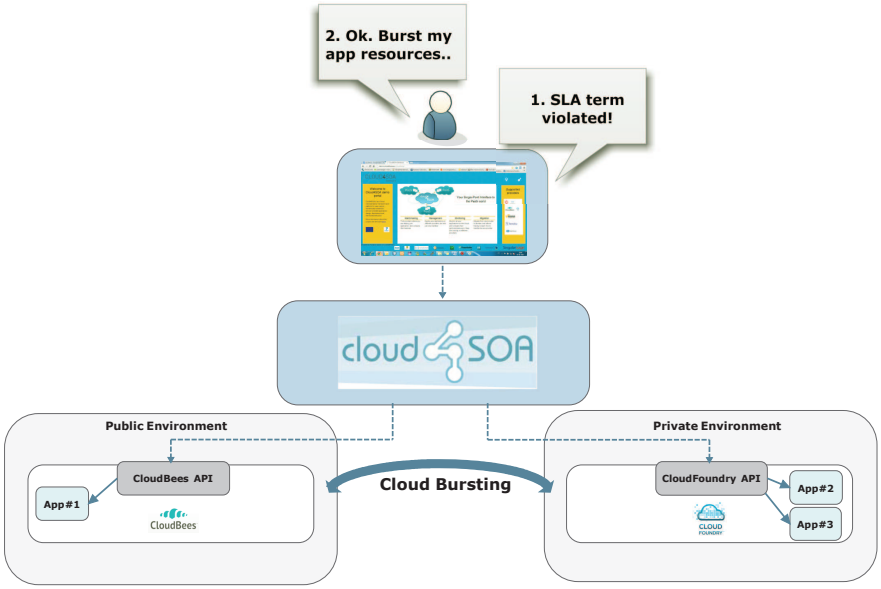


FIGURE 5.2. *Bursting on demand scenario representation*

TABLE 5.1
Comparison of Performance between Cloud4SOA and the PaaS API

Operation	Cloud4SOA	Provider API
Deploy (CloudFoundry)	42 sec.	28 sec.
Deploy (CloudBees)	155 sec.	130 sec.
Stop (CloudFoundry)	11 sec.	5 sec.
Undeploy(CloudFoundry)	15 sec.	6 sec.

Regarding the adapter, the experience has been more positive with respect to the remote adapter than with the local one. As discussed in Sect. 5.1, the remote adapter has been implemented as a brand new independent component and did not require any additional wiring code in the Governance layer. Moreover, Cloud4SOA provides a Generic Remote Adapter template that greatly helps in implementing the adapter.

With respect to the local adapter, the implementation requires a better understanding of the Cloud4SOA architecture and code, especially on the Governance layer. Moreover it supposes a threat for the integrity of the platform since a change in the code of this layer may affect the functioning of the whole platform. In order to overcome this, it should be possible to encapsulate the local part of adapters as independent projects in a similar way to the remote part.

6. Conclusions. Currently, the PaaS market is still quite young, chaotic and highly fragmented, dominated by a few providers which use and promote incompatible standards and formats. This introduces adoption barriers due to the lock-in issues that prevent the application and data portability and the semantic interoperability between heterogeneous Cloud PaaS offerings. Additionally, there is a need by software developers not only to migrate applications from one Cloud platform to another but also to manage and monitor distributed applications spanning multiple public and/or private Clouds.

In this work we present the Cloud4SOA solution for multi-Cloud application management, monitoring and migration; all of which are completely independent of a particular vendor. In this way developers are not just aided in the form of a tool, but are provided a great opportunity for Cloud software development in general, alleviating the lock-in that has long been attached to platform adoption.

The evaluation of Cloud4SOA, based on the two hybrid scenarios, showed that there is an overhead imposed by the platform but it is considered acceptable taking into account the added-value provided since it allows deploying, governing and monitoring of applications across diverse heterogeneous public and private Clouds.

Concluding the main contributions of Cloud4SOA can be summarized in the following:

(i) It adopts a user-centric approach for the UI design in order to offer friendly and intuitive interfaces

tailored to the user's needs.

(ii) It solves the interoperability issues between heterogeneous Cloud PaaS environments by offering a harmonized API.

(iii) It offers a multi-PaaS application management and monitoring solution that enables the deployment of applications at public, private or hybrid multi-Cloud environments.

Acknowledgments. This work is partially funded by the European Commission in the context of the Cloud4SOA project, under the contract No. 257953 within the Seventh Framework Programme of the European Community. The authors would like to thank Esen Kunt and Stefano Travelli for their contribution to the design/implementation of the User Interface and David Goncalves Cunha from PTIN for providing the framework used for the hybrid scenarios.

REFERENCES

[1] A. Andrieux, , K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, *Web services agreement specification (ws-agreement)*, Specification from the Open Grid Forum (OGF), (2007). available at: http://www.ogf.org/documents/GFD.107.pdf.

[2] Y. Anokwa, G. Borriello, T. Pering, and R. Want, *A User Interaction Model for NFC Enabled Applications*, in Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops, Washington, DC, USA, 2007, IEEE Computer Society, pp. 357—-361.

[3] M. Beaudouin-Lafon, *An overview of human-computer interaction*, Biochimie, 75 (1993), pp. 321–329.

[4] M. Beaudouin-Lafon, *Instrumental interaction: an interaction model for designing post-wimp user interfaces*, in Proceedings of the SIGCHI conference on Human Factors in Computing Systems, CHI '00, New York, NY, USA, 2000, ACM, pp. 446–453.

[5] D. Benyon and D. Murray, *Applying user modeling to human-computer interaction design*, Artificial Intelligence Review, 7 (1993), pp. 199–225.

[6] E. Bernard, J. Cachopo, B. Ciciani, D. Didona, F. Giannone, M. Little, S. Peluso, F. Quaglia, L. Rodrigues, P. Romano, and V. A. Ziparo, *Cloudtm d2.1: Architecture draft*, (2011). available at: http://www.gsd.inesc-id.pt/ romanop/files/deliverables/D2_1.pdf.

[7] V. D. Cunsolo, S. Distefano, A. Puliafito, and M. Scarpa, *Cloud@home: Bridging the gap between volunteer and cloud computing*, Proceedings of the 5th International Conference on Intelligent Computing (ICIC 2009), (2009).

[8] F. D'Andria, S. Bocconi, J. G. Cruz, J. Ahtes, and D. Zeginis, *Cloud4soa: Multi-cloud application management across paas offerings*, Management of resources and services in Cloud and Sky computing (MICAS) 1st Workshop, organized in conjunction with SYNASC 2012, Timisoara, Romania, 2012.

[9] B. di Martino, D. Petcu, R. Cossu, P. Goncalves, T. Mahr, and M. Loichate, *Building a mosaic of clouds*, in Proceedings of the 16th European Conference on Parallel and Distributed Computing (Euro-Par 2010), Ischia-Naples, Italy, 2010.

[10] Economist, *Battle of the clouds*, (2009). available at http://www.economist.com/opinion/displaystory.cfm? story_id=14644393

[11] ———, *Clash of the clouds*, (2009). available at: http://www.economist.com/displaystory.cfm?story_id=14637206

[12] Gartnet, *Gartner identifies the top 10 strategic technology trends for 2013*, (2013). available at: http://www.gartner.com/newsroom/id/2209615.

[13] P. Harsh, Y. Jegou, R. Cascella, and C. Morin, *Contrail virtual execution platform challenges in being part of a cloud federation*, In Proceedings of the 4th European conference on Towards a service-based internet (ServiceWave'11), Berlin, Heidelberg, 2011, pp. 50 – 61.

[14] R. Jimenez-Peris, M. Patino-Martinez, K. Magoutis, A. Bilas, and I. Brondino, *Cumulonimbo: A highly-scalable transaction processing platform as a service*, ERCIM News 89, Special Issue on Big Data, (2012). available at: http://ercim-news.ercim.eu/en89/special/cumulonimbo-a-highly-scalable-transaction-processing-platform-as-a-service.

[15] G. Lee, C. M. Eastman, T. Taunk, and C.-H. Ho, *Usability principles and best practices for the user interface design of complex 3D architectural design and engineering tools*, International Journal Human-Computer Studies, 68 (2010), pp. 90–104.

[16] N. Loutas, ed., *D1.1 Requirements Analysis Report*, 2011. available at: http://www.cloud4soa.eu/sites/default/files/Cloud4SOA %20D1.1%20Requirements%20Analysis.pdf.

[17] N. Loutas, ed., *D1.3 Cloud4SOA Reference Architecture*, 2011. available at: http://www.cloud4soa.eu/sites/default/files/D1.3 %20Cloud4SOA%20Reference%20Architecture.pdf.

[18] N. Loutas, E. Kamateri, and K. Tarabanis, *A semantic interoperability framework for cloud platform as a service*, In Proceedings of 3rd IEEE International Conference on Cloud Computing Technology and Science, Athens, Greece, 2011, pp. 280 – 287.

[19] N. Loutas, V. Peristeras, T. Bouras, E. Kamateri, D. Zeginis, and K. A. Tarabanis:, *Towards a reference architecture for semantically interoperable clouds.*, CloudCom 2010, (2010).

[20] NIST, *A nist definition of cloud computing*, (2011). available at: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[21] E. Oliveros, S. Garcia, S. Strauch, T. Binz, A. Spriestersbach, K. Stanoevska, S. Carrie, and J. Niemoeller, *4caast value proposition*, white paper, (2011).

[22] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth,

J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, *The reservoir model and architecture for open federated cloud computing*, IBM Systems Journal, (2008).

[23] A. Sheth and A. RanabahuS, *Semantic modeling for cloud computing, part i and ii*, IEEE Internet Computing Magazine, 14 (2010), pp. 81 – 83.

[24] W. Theilman, R. Yahyapour, and J. Butler, *Multi-level sla management for service-oriented infrastructures*, Proceedings of the 1st European Conference on Towards a Service-Based Internet (ServiceWave '08), Madrid, Spain, 2008.

[25] H. Zhang, G. Jiang, K. Yoshihira, H. Chen, and A. Saxena, *Intelligent workload factoring for a hybrid cloud computing model*, in Proceedings of the 2009 Congress on Services - I, SERVICES '09, Washington, DC, USA, 2009, IEEE Computer Society, pp. 701–708.

Edited by: Marc Eduard Frîncu
Received: Mar 15, 2013
Accepted: Mar 29, 2013

DOI 10.12694/scpe.v14i1.825
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 1, pp. 33–45. http://www.scpe.org

THE MOSAIC BENCHMARKING FRAMEWORK: DEVELOPMENT AND EXECUTION OF CUSTOM CLOUD BENCHMARKS

GIUSEPPE AVERSANO*, MASSIMILIANO RAK† AND UMBERTO VILLANO‡

Abstract. A natural consequence of the pay-per-use business model of Cloud Computing is that cloud users need to evaluate and to compare different cloud providers in order to choose the best offerings in terms of trade-off between performance and cost. But at the state of the art, in cloud environments no real grants are offered by providers about the quality of the resources offered and no clear ways exists to compare two different offerings. Moreover, the high elasticity of cloud resources (virtual machines can be added or removed in few minutes) makes the evaluation of such systems a hard task. In this paper we propose to build ad-hoc benchmark applications, whose behavior is strictly related to user needs and which can be used to compare different providers. The proposal is based on the use of the mOSAIC framework, which offers a deployable platform and an API for building provider-independent applications. Due to such independence, we are able to compare directly multiple cloud offers. The paper details the proposed approach and the framework architecture implemented in order to apply it. Simple case studies illustrate how the framework works in practice. Moreover the paper presents a detailed analysis of the state of the art and of the problem of benchmarking in cloud environment.

Key words: Cloud computing, Benchmark, Cloud Performance, mOSAIC, Platform as a Service

1. Introduction.

The emerging cloud computing paradigm owes most of its success to the pay-per-use business model. One of the key idea of the cloud is that all resources, even the ones that usually are physically managed, are offered *as a service* exploiting virtualization techniques. The charges for their use depends only on their actual real usage of the resources acquired. This approach reduces maintenance costs and helps to optimize resource usage, as they are dynamically acquired only when really needed.

As a natural consequence, cloud users need to evaluate and compare different cloud providers in order to choose the best offerings in terms of the trade-off between performance and price. But, at the state of art, techniques and tools to perform this comparison are still lacking in cloud environments. As a matter of fact, providers offer no clear grants about the quality of the resources offered, and no clear ways exist to compare two different offerings. The only support offered by providers is the adoption of Service Level Agreements (SLAs). These are contracts between service consumer and provider that assure the level of quality granted, in the form of natural language contracts with very few quantitative references.

In this context, the most common approach to evaluate the quality of cloud services is the use of monitoring tools. They constantly evaluate the state of resources acquired, as the bandwidth available to users, or the actual throughput of services. Sometimes such services are offered by providers, as Amazon CloudWatch, while in other cases third-party services have to be used. However, it should be noted that monitoring can be exploited only *after* that the resources have been actually acquired. Benchmarking techniques, instead, aim at predicting and describing in a synthetic way the behavior of resources to be (possibly) acquired, performing off-line evaluations.

In order to point out clearly the difference among monitoring and benchmarking, in the following we will refer to benchmarking activities when we aim at performing static evaluations of performance indexes, while we will refer to monitoring when measurements are performed on the software *in production*, i.e., when it is subject to a workload due to real user requests. As such, a benchmark is a program that generates a well-known workload on the system under test (SUT) and enables the expert to measure a set of predefined performance indexes. In our work, we exploit benchmarking techniques to compare alternative cloud solutions.

In this paper, an extended version of [24], we tackle the benchmarking problem in the context of provider-independent cloud frameworks, as mOSAIC[20, 22], Contrail[19, 23] or Optimis[15], which enable their users to develop applications independently of the target providers. In such frameworks, developers create their solution using local resources (or even cloud platforms) and choose the provider where the developed services are to be offered in a second step, comparing the offerings on the basis of the software developed. In such a case, the availability of a benchmarking solution that can help the end user to perform a clear comparison among different offerings is of great interest.

*Department of Engineering, University of Sannio, Piazza Roma 21, 82100 Benevento, Italy (giuseppe.aversano@unisannio.it).
†Department of Information Engineering, Second University of Naples (SUN), Via Roma 29, 81031 Aversa, Italy (massimiliano.rak@unina2.it).
‡Department of Engineering, University of Sannio, Piazza Roma 21, 82100 Benevento, Italy (villano@unisannio.it).

We propose a novel approach to benchmarking, based on the idea of building custom benchmark applications, able to measure indexes strictly related to cloud user needs. We adopted the mOSAIC framework, which is a cloudware which helps a cloud developer to build provider-independent applications. Thanks to the proposed benchmarking framework, a mOSAIC user can easily generate new applications from its mOSAIC-based ones. These applications can successively used can be run, thanks to mOSAIC, on many different providers. The result is that the cloud user, with minimal cost and effort, is made able to run benchmarks dynamically, collecting comparable results from multiple providers.

The remainder of this paper is organized as follows. The next section briefly outlines the state of art in cloud benchmarking, while section 3, briefly summarizes the mOSAIC framework concepts. Section 4 illustrates the approach adopted and the possible different benchmarking scenarios for a mOSAIC application. Section 5 summarizes the framework organization and the main mOSAIC components offered. Section 6 illustrates how to build a mOSAIC application which manages the benchmark life cycles while Section 7 describes the benchmark applications adopted for some key components and for a simple full application. Finally, section 8 describes the results obtained and outlines our future work.

2. Related Work. This section aims at offering a clear and global view of the state of art about benchmarking in cloud environment. Its goal is to outline the new challenges in cloud environment, the currently available solution and the new proposals. The roles of benchmarking and of monitoring of services are sometimes misunderstood. In the following we will refer to benchmarking activities when we aim at performing static performance evaluations of indexes. A benchmark is a program that generates a well-known workload on the system under test (shortly, *sut*) and enables the expert to measure a set of predefined performance indexes. At the state of the art, few solutions exist that tackle the problem of benchmarking in cloud environments, even if a lot of different approaches exists in different contexts. This section will firstly present benchmark results and products in many different contexts (benchmark in HPC systems, comparison between IaaS providers, existing frameworks, etc.), Then, a brief critical analysis of the solution presented is performed, outlining the open research problems.

2.1. HPC Benchmarking. Even if recently cloud providers have started to offer HPC-oriented services (like the Amazon HPC service eAmazon, HPC Services, available at http://aws.amazon.com/hpc-applications/), HPC users started to be interested to cloud paradigm from its birth (as shown by the incredible amount of discussions comparing GRID and Cloud). In HPC systems, performance benchmarking is a well known practice. Many stable benchmarking suites of codes exist, among which NAS Parallel Benchmarks (NPB) [4], SkaMPI[5], LinPack [3] (just to mention the most famous ones).

Such benchmarks were adopted to compare HPC systems, like the ones in the Top500 list [9]. An interesting survey of HPC-related problems can be found in [11].

The first results in the context of cloud benchmarking can be found in the tentative of adapting such benchmarks to cloud based clusters, in order to understand the real applicability of solutions based on virtual machines in this context. Probably the most relevant paper in this field is the one by Walker [27] that tries to outline the real performance of cloud based clusters. The paper results are now outdated, but the approach followed is interesting.

Papers as [14, 28] propose the systematic benchmarking of clusters of virtual machines, obtained from IaaS cloud providers, and offer some interesting consideration about the performance perceived by users.

2.2. Benchmark Standard Solutions. It is a matter of fact that, at the state of the art, there exists no standard for cloud benchmarking. Here, we focus on two of the most important benchmark consortiums that are addressing cloud-related benchmarking: SPEC[8] and TPC[10]. SPEC has developed through the years some solutions that can be adopted in the context of cloud environments. From March 2011 a sub-committee dedicated to cloud has been formed, and works on cloud-related benchmarking problems. Currently the offerings of SPEC that can be considered of interest in the cloud-oriented contexts are SPECvirt[7] and SPA SOA [6]. Moreover, the SPECweb benchmark, which focuses on web servers/application server benchmarking, is also of high interest. The TPC [10] is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry. In the context of clouds, which are born to offer transactional services, TPC benchmarks are of great interest. TPC can be considered as application-level benchmarks in cloud environments and they are a basis for the evaluation of the actual performance offered by standard transactional software on the top of IaaS-delivered machines.

2.3. Benchmarking Services. The adoption of standard benchmarks, as described above, is interesting to compare the offerings from different cloud providers, mainly at IaaS level. The main drawback of the standard benchmarks is that they are not designed to cope with the elasticity of cloud offerings. They just provide an evaluation of static systems, and are not able to manage systems that dynamically may change over time, with highly variable performance. A possible solution is to benchmark the systems after any elastic mutation, for example by adopting the ubiquitous as-a-service approach. This is the solution adopted in [18, 25], where the use of a benchmarking service is proposed, that starts up the benchmark execution on any newly-delivered machine. CloudHarmony [1] is probably the most interesting solution of this kind. It offers a very large collection of customizable benchmarks that is continuously executed on remote resources. The solution is very interesting and looks similar to the ones adopted for monitoring. In fact, in this case, it is hard to define clearly the difference between monitoring and benchmarking. While CloudHarmony is built on the top of standard benchmarks, CloudSleuth [2] can build up a cloud application benchmark, which generates a standard workload on target services in order to perform comparisons.

2.4. Cloud-Oriented Benchmarks. Currently only few cloud-specific proposals for benchmarking exist in the literature. The contribution can be summarized by mentioning the following Cloudstone[26] and CloudCmp[17] projects. These are academic projects, which aim at offering flexible frameworks able to build up custom workloads to be run in cloud environments. Cloudstone is an academic open source project from the UC at Berkeley. It provides a framework for testing realistic performance. The project does not publish as of yet comparative results across clouds, but provides users with the framework that allows them to get their measurements. The reference application is a social Web 2.0-style application. The Cloudstone stack can be divided into three subcategories: web application, database, and load generator. When running benchmarks, the load generator generates load against the web application, which in turn makes use of the database. The Web 2.0 application used in Cloudstone is Olio, an open source social-event application with both PHP and Ruby implementations. Cloudstone uses the Ruby implementation. Nginx and HAProxy are used to load balance between the many Rail servers (i.e., thin servers). Cloudstone supports the use of either MySQL or PostgreSQL. Replication when using MySQL is supported using the built-in master-slave replication features. Replication when using PostgreSQL is supported using PGPool-II, a type of load balancer for PostgreSQL database servers. Cloudcmp instead focuses on typical Cloud-based IaaS services (storage and virtual machines) for which a set of custom kernel benchmark were proposed that aims at comparing key features of the systems. The academic project, also supported by some private companies, was applied to compare few common cloud providers. The main limit of this kind of benchmarks is that they do not take into account the variability of results of the same provider in different conditions. Moreover, the results, even if offer a general idea of provider cloud behavior, can hardly be used by a service provider to make a real choice, taking into account how its application behaves.

2.5. A Critical Analysis. The above presented solutions clearly state the main problem of benchmarking in cloud environments. These are: how to represent with a single (or a few) indexes the dynamicity of a cloud environment? how useful is to perform few static measurement? As a matter of fact, at the state of the art, the main direction for performance evaluation of cloud offerings is oriented to the use of monitoring. Benchmarking solutions, like the ones proposed commercially by CloudHarmony or CloudSleuth, try to adapt benchmarking solutions to typical monitoring approaches. Moreover, currently it is impossible to identify a single workload representing the incredibly large variety of cloud services offered. The direction adopted is to reuse benchmarks targeted at specific services, and to try to deliver them in a more flexible way, as in [16], which uses TPC-W to evaluate the cloud services delivered. It should be noted that benchmark results are not reliable in cloud environments. The same request of resource may lead to completely different behavior, as shown in [12, 13]. This means that benchmarking should be applied on the single specific resource, more than on generic resources offered by a given cloud provider. This approach, on the other side, has a high impact on the costs. It is questionable whether it is acceptable to pay for a resource usage which is just targeted to execute a benchmark?

In conclusion, benchmarking tools should be more flexible than the those commonly adopted in other environments. The adoption of a single performance index is not acceptable and workload definition should be customizable by the benchmarker user, according to its specific needs.

3. The mOSAIC Framework. mOSAIC aims at providing a simple way to develop cloud applications [20, 21, 22]. Hence, the target user for the mOSAIC environment is the application developer (*mOSAIC user*). In mOSAIC, a cloud application is structured as a set of components running on cloud resources (i.e., on resources leased from a cloud provider) and able to communicate with each other. Cloud applications are often

provided in the form of Software-as-a-Service, and can also be accessed/used by users other than the mOSAIC developer (i.e., by *final users*). In this case, the mOSAIC user acts as service provider for final users.

The mOSAIC framework is composed of a few stand-alone components. Among them, the most important roles are played by the *Platform* and the *Cloud Agency*. The first one (mOSAIC Platform) enables the execution of applications developed using the mOSAIC API. The second one (Cloud Agency) acts as a provisioning system, brokering resources from a cloud provider, or even from a federation of cloud providers.

mOSAIC can be useful in three different scenarios:

- when a developer wishes to develop an application not tied to a particular cloud provider;
- when an infrastructure provider aims at offering "enhanced" cloud services in the form of SaaS;
- when a final user (e.g., a scientist) wishes to execute his own application in the cloud because he needs processing power.

A mOSAIC application is built up as a collection of interconnected *mOSAIC components*. Components may be (*i*) core components, i.e., predefined helper tools offered by the mOSAIC platform for performing common tasks, (*ii*) COTS (commercial off-the-shelf) solutions embedded in a mOSAIC component, or (*iii*) *cloudlets* developed using the mOSAIC API and running in a *Cloudlet Container*. mOSAIC cloudlets are stateless, and developed following an event-driven asynchronous approach [20, 21].

The mOSAIC platform offers ready-to-use components such as queuing systems (*Rabbitmq* and *zeroMQ*), which are used for component communications, or an HTTP gateway, which accepts HTTP requests and forwards them to application queues, NO-SQL storage systems (as KV store and columnar databases). mOSAIC components run on a dedicated virtual machine, named mOS (mOSAIC Operating System), which is based on a minimal Linux distribution. The mOS is enriched with a special mOSAIC component, the *Platform Manager*, which makes it possible to manage set of virtual machines hosting the mOS as a virtual cluster, on which the mOSAIC components are independently managed. It is possible to increase or to decrease the number of virtual machines dedicated to the mOSAIC Application, which will scale in and out automatically.

A cloud application is described as a whole in a file named *Application Descriptor*, which lists all the components and the cloud resources needed to enable their communication. A mOSAIC developer has both the role of developing new components and of writing application descriptors that connect them together.

4. Benchmarking Problem Analysis. The problem we aim at solving is to offer facilities for off-line evaluation of mOSAIC applications and their behavior when running on resources from many different cloud providers. This analysis will be of help to the developer to design better his solutions, and to compare different design choices.

The benchmarking problem can be faced as a typical performance evaluation problem. Benchmarking tools will be a clear definition of the execution conditions for the target application to be evaluated. In any performance evaluation problem the key to success is the identification of the goals of the performance analysis. This section aims at stating the different goals that the benchmarking procedure may assume. In order to identify such goals it is important to make some key assumptions: the target of our benchmarks are always mOSAIC entities (modules, components, resources), and the benchmarking users are mOSAIC actors.

We have identified a few main benchmarking scenarios that represent possible conditions under which a benchmarking procedure should take place. We describe each of these scenarios through three elements: Target User, Main Goal, Target Resource.

The Target User is the user interested in performing the benchmarking. He could be the mOSAIC User (Developer) or a Final User (i.e., the one that uses applications developed with mOSAIC). The Main Goal is the expected result of the benchmarking procedure (e.g., comparing different providers).

The Target Resource is the subject of the Benchmarking procedure, i.e., what the Benchmark aims at stressing (the system under test).

The main scenarios we focus on are summarized in Table 4.1. Each scenario is enriched with a brief description of the goal to be achieved by the performance analysis.

4.1. The benchmarking framework approach. The analysis presented outlined the main requirements of the benchmarking framework for the mOSAIC platform. The first aspect to be taken into account is that the component-based approach adopted in mOSAIC has two side effects on benchmarking. The positive one is that it is possible to evaluate independently the single pieces of a solution. This helps in defining a benchmarking framework composed of independent benchmarks that can be composed in order to fulfill the different requirements of each scenario. On the bad side, the easiness in changing the configurations of the mOSAIC

TABLE 4.1
Summary of Benchmarking Scenarios

Target User	**Goal**	**Target Resource**	**Goal Description**
Developer	Compare Providers	Cloud resources and/or services	The goal is to enable a developer to choose among services offered by different providers
Developer	Evaluate mOSAIC	mOSAIC Components	The goal is to enable a developer to choose among different components and evaluate how each component behaves
Developer	Compare Applications configurations	Mosaic applications	The goal is enable the mOSAIC developer to compare two different mOSAIC applications in order to configure them at best
Developer	Offer Benchmark as a service	Cloud providers	The goal is to help developers which aim at developing benchmark as a service applications
Developer	Predictions	mOSAIC applications, components and resources	The goal is to help developer to build up benchmarks whose goal is to predict the behavior of a target application
Final User	Compare Applications	mOSAIC applications	The goal is to enable the final user to know the performance offered by the target applications for comparison with other offerings (SLA-related problem)

platform at run-time may imply a huge number of different tests to be performed and difficulties in interpreting evaluation results. Moreover, the analysis outlines the need to customize workloads and to organize them, in preference to the definition of complex workloads. Workloads can be identified for some of the key components of the framework, while mOSAIC users should be able to easily enrich the set of workloads according to their needs. The component approach enables the reuse of the workloads defined.

In conclusion, the main requirements of the benchmarking framework are the capability of offering tools that help mOSAIC user to build up custom solutions and to execute targeted benchmarks for predefined goals. In addition to this, the benchmarking framework should help the developer to identify the systems (components, resources, etc) to be benchmarked, to isolate them and to start up a benchmarking procedure on them.

4.2. Benchmarking Methodology. The Benchmarking framework founds on the idea of using mOSAIC modularity and its component-based approach to implement all the above presented concepts. The proposed scenarios put in evidence (except the scenario 4) that in any case when applying a benchmark to a system, we go through mOSAIC components and interfaces. So it is possible to build up the full framework as a collection of mOSAIC components connected with each other.

In order to implement a benchmark in mOSAIC the approach adopted is the following. Once the System under Test (SUT), i.e., the target of the benchmarking, has been identified, a Benchmarking Model is defined

around it. This represents the actual benchmark and it will be implemented as a mOSAIC application.

Once the Benchmark Model application (BM application from now on) has been designed and all its components are available, a mapping of it on the top of the resources will be defined. This represents the definition of the testing condition.

In order to control (start, stop, monitor) the Benchmarking procedure, the BM application is enriched with dedicated components that, among other features, offer a simple interface to final users. These components can be seen as an additional and complementary application, named *The Benchmarker*. This application resides on resources different from the ones adopted for the BM application, and it has the role of controlling the benchmark evolution. To summarize, the Benchmarker will be build by the mOSAIC User (the Developer) through a set of dedicated components, that solve intermediate problems. Examples of such components are:

- Monitor of the Benchmarking procedure
- Manager of Benchmarking results
- Manager of Benchmark tests
- User interfaces

4.3. Pros and Cons of the approach. The above proposed approach enables a clear distinction among Workloads, Benchmark Target and Benchmark conditions. Moreover, it helps in reducing the overhead due to data collection (by defining the role of the Benchmarker and its mapping on resources). When evaluating mOSAIC components, mOSAIC applications and cloud resources, this solution clearly identifies the effect of each module and leads to a clear understanding of the offered performance.

The main drawback of the approach adopted is that it may imply additional costs in the benchmarking procedure. Resources must be acquired in order to be evaluated. Moreover, the Benchmarker consumes additional resources.

5. The Benchmarking Framework. The approach described in the above section leads to a global vision of the benchmarking framework shown in Figure 5.1. The picture puts in evidence that Benchmarking Components are organized in few modules:

- **Benchmark Models**: this module collects the components needed for building up a specific benchmark model. The benchmarking framework offers mainly the components needed to build up custom benchmark workloads. It is up to the developer to use them, along with all the other mOSAIC API to build up a dedicated Benchmark model.
- **Benchmarker**: this module collects all the components dedicated to control the benchmarking evolution. Mainly it offers a set of controllers, which start, stop and manage the execution of Benchmarking Models, Monitoring and Analyzers components, which collect data results and perform the analysis on results collected to produce the required performance indexes.
- **Benchmark GUI**: this module collects all the components needed to offer a simple interface to the Benchmarker user (developer or possibly final user). This module reuses existing components from mOSAIC (as the HTTP gateway) and offers some examples of HTTP backend, i.e., the components adopted to send messages to the controller.

The Benchmarking Models represent the description of the workload and of the system under test. It is up to the framework user (typically a developer) to build up such models on the basis of the goals of his analysis.

Benchmark Models are composed of two key elements: The System under test and the workload generators. The System Under test can be any of the target resources identified in the previous section (cloud resources, mOSAIC components, mOSAIC application, mOSAIC modules). The Workloads represents the real benchmark to be applied. This module offers facilities which help in the development of such generators. These components can be customized by the mOSAIC User in order to build up custom solutions.

While the Benchmarker GUI is a simple customization of common mOSAIC components dedicated to web-based interfaces, the *Benchmarker* is the key of the benchmarking procedure. If the Benchmark Model represents *what* to benchmark, the Benchmarker is the implementation of *how* benchmarking takes place. The core of the Benchmarker is the *Controller*, which controls the benchmark executions on the basis of messages received on a dedicated queue.

6. Benchmark Control Components. As outlined before, the benchmark applications are composed of two main block: *Benchmarker* and *Benchmark Models*. In this section we describe the main components devoted to control the benchmark execution (*Benchmarker*).

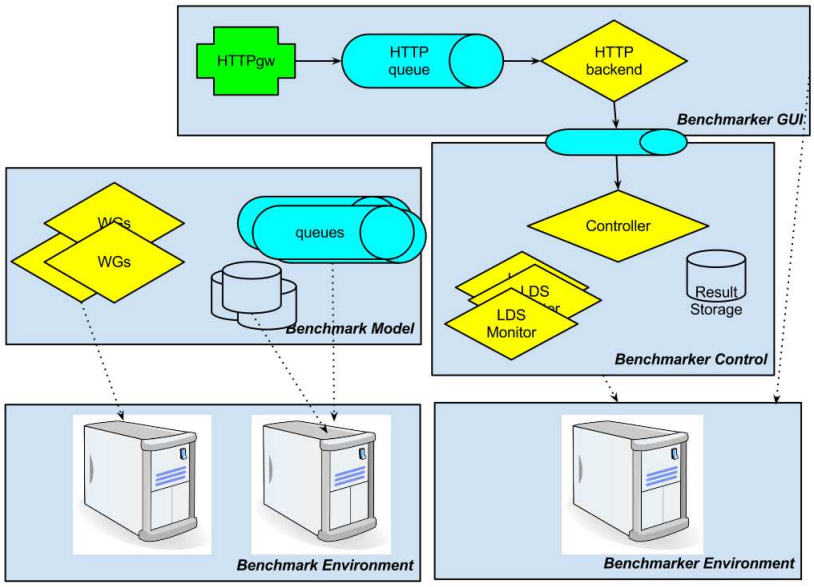


FIGURE 5.1. *A view of the benchmarking framework*

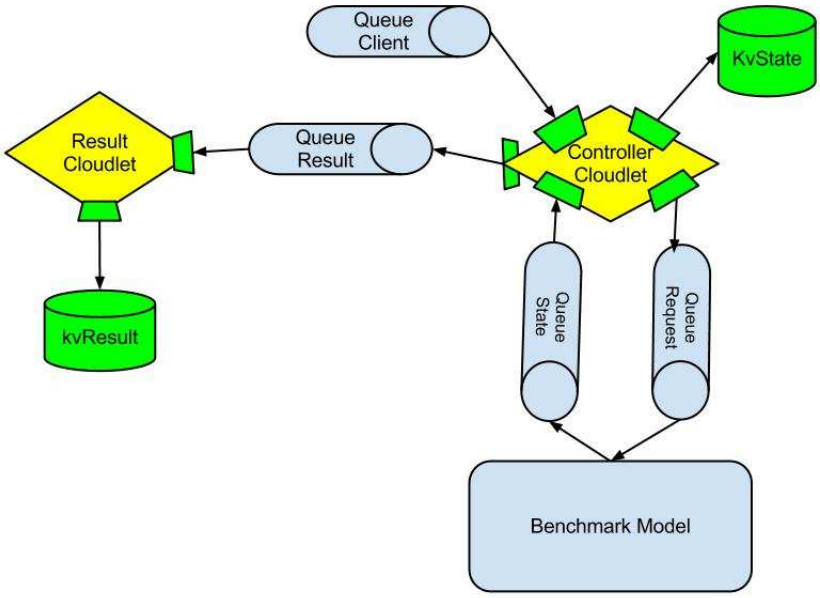


FIGURE 6.1. *The Benchmark Controller Components*

The mOSAIC benchmarking framework offers mainly two key components: the Controller Cloudlet and the Result Cloudlet. A typical mOSAIC Benchmark Control application is structured as shown in figure 6.1. The Result Cloudlet has simply the role of collecting the benchmark results and tostore them in a key-value (KV) Store; the Controller is devoted to maintain the full benchmark life cycle. The Controller Cloudlet maintains state information in a KV store resource and evolves during the benchmark execution according to the user requests received from the *Queue Client*. The Controller is embedded with a Workload generator that sends out messages that activate the Benchmark model and stresses it through the request and State queues. Moreover, it forwards the results to the Result cloudlet when they are available.

The mOSAIC Benchmarking framework offers a set of simple Workload Generators that can be extended by

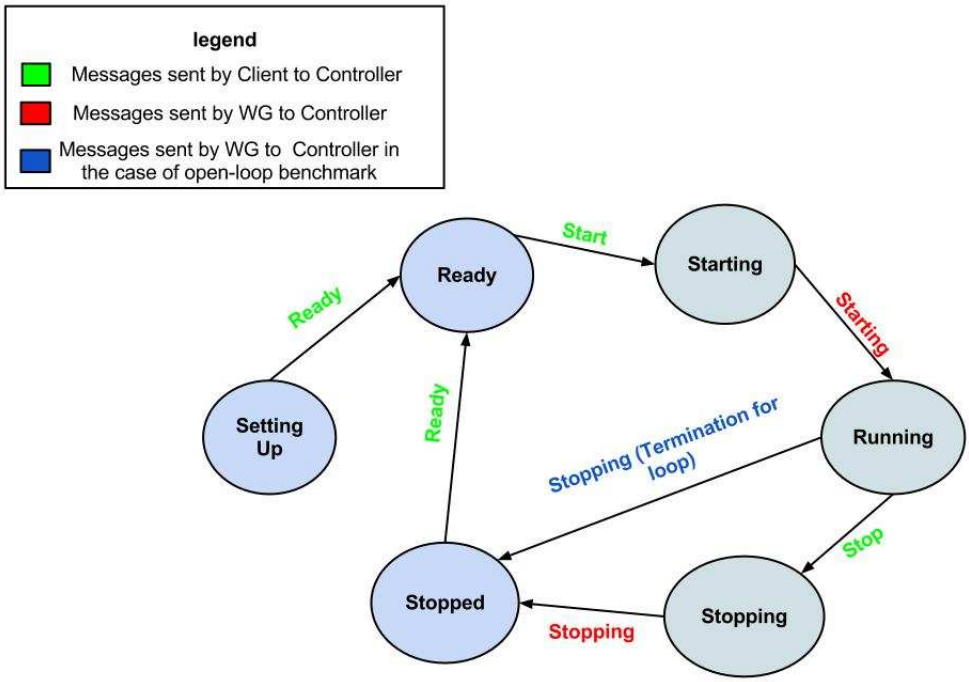


FIGURE 6.2. *The Benchmark Control States*

mOSAIC Developers in order to build up custom workloads. At the state of the art, mOSAIC offers a generator that sends out one or more bursts of concurrent messages, a sequence of different messages, or generates a fixed set of messages with known delays. In the future, we aim at offering generators that stress the application with other type of workloads.

It should be noted that all the controller components can be deployed on dedicated machines in order to avoid the sharing of resources among components dedicated to benchmark control and the benchmark model (i.e., what is to be benchmarked). All the benchmark applications can be used using the same pattern. They behave according to the state diagram presented in figure 6.2. The events that enable the state transitions are linked to messages that End users can send to the benchmark application, or to messages generated internally by the application.

The Benchmarking states have the following meaning:

- **SettingUp**: The benchmark application is starting (not all the components are available, the End user should avoid to interact with application).
- **Ready**: The benchmark application is ready to receive requests from the End user. All components correctly started.
- **Starting**: A Start request was sent by the End user to Controller (through the web interface).
- **Running**: The benchmarking application is running some benchmark tests. End users should not interact with the application (in any case, any End user message is intercepted by the controller that does not deliver it to the application).
- **Stopping**: The benchmarking application has received a Stop message and it is waiting that the tests are completed, without sending more messages. The Stop message is not needed in the case of workloads with fixed amount of messages.
- **Stopped**: The benchmarking application has terminated the tests. Note that in this case it is not ready to start a new set of tests (i.e., to accept a Start message). New tests can be started only in the Ready state (reachable through a Ready message).

The following messages are enabled:

- **Ready**: This message indicates that the client aims at starting a new set of benchmarks. The message moves the benchmarking app to Ready from SettingUp or Stopped states to Ready (it is discarded in other cases).

- **Start**: This message starts the benchmark procedure. It is accepted only when the application is in Ready state (otherwise it is discarded).
- **Stop**: This message ends the benchmark procedure, i.e., it advise the controller to end sending out messages. It is accepted only in Running state. It is not needed when the workload assumes a fixed (or limited) number of messages.
- **Plot**: This message makes available the full set of results produced by the benchmark procedure. The message is accepted only in Stopped state (otherwise it is discarded). Note that it does not materially plot the results, unless the Web UI is opportunely configured.
- **Check**: This message makes available the state of the system.

The Controller discards all the messages received from the user that are not explicitly managed in the above described state diagram. This implies that if an user sends many Start messages together, only one is considered and all the other are discarded. Moreover, once a benchmark has been completed, the user needs to ask explicitly (through the *Ready* message) to make it available for a new set of tests.

Such Controlling applications are usually interfaced by means of the mOSAIC HTTP gateway, which offers a Web-based interface and accepts the above described messages through a simple REST interface, using the benchmark messages as content of the HTTP message:
`(URL: http://<ip> :1337, METHOD: POST).`

It is possible to read the state of the benchmarking process through a dedicated REST call, which depends on the benchmark Model to use:
`URL: http://<ip>:1337/raw/<bucket>/<key>, METHOD: GET`
Moreover, another REST call lets the user retrieve the benchmark results:
`URL: http://<ip>:1337/raw/<bucket>/<key>, METHOD: GET`
A Set of simple HTTP clients and scripts are offered in order to automate the benchmarking process.

7. Benchmark Examples in mOSAIC. In this section we focus on how to use the mOSAIC benchmarking framework to make an evaluation of a mOSAIC cloud application. We consider the first three scenarios described in section 4, i.e., the comparison of cloud providers (the stress of Cloud Resources described in the next two subsections), the comparison of mOSAIC components (the stress of cloudlets described in the third subsection) and the evaluation of a full mOSAIC Applications (last subsection). We executed the tests described hereafter on a private cluster based on OpenNebula, enabled in order to run the mOSAIC platform. In the mOSAIC repositories (`https://bitbucket.org/mosaic`) it is possible to retrieve the code of the benchmarking framework and bundle versions of mOSAIC that enable its execution on every kind of machine and cloud provider (it was tested on OpeNebula, Openstack, Cloudsigms, Deltacloud, Flexiscale, Amazon and others).

7.1. Benchmarking Cloud Resources: Queue. The main goal of applications that stress a single queue is to evaluate the response time needed for each single message. The main goal of the benchmarking process is to offer a reference to developers in order to predict the time needed in their application to deliver a message from a component to another. The performance of a Queue depends heavily on a lot of parameters that are hardly controllable in a cloud environment (for example, using mOSAIC RabbitMQ COTS component it depends on the VMs on which they run, and on the number of other components sharing CPU and memory). The goal of this benchmark is to create a simple repeatable evaluation that can be used to tune the application. Tests can be repeated (consuming resources) online, in order to perform a new evaluation in different conditions. We offer three different Benchmark applications: QueueBenchmark, QueueBenchmarkBlock, QueueBenchmarkStep, which generate three different loads on the queue. The first one produces a set of concurrent requests. The second one repeats the first one a known number of times, the third generates a sequence of concatenated messages. Repeating these tests with a variable number of messages and repetitions can help the developer to understand how the queue performs in different load conditions. The results, reported in table 7.1, show that Response time depends on the Number of concurrent messages (the measurement with a low number of concurrent messages, instead, is unreliable).

7.2. Benchmarking Cloud Resources: KV Store. The main goal of applications that stress a Key-value Store is to evaluate how many sequences of SET/GET operations can be completed in a fixed interval of time. The main goal of the benchmarking process is to offer a reference to developers in order to predict the time needed in their application to access the Key-value Store. The performance of a KV store heavily depends on a lot of parameters that are hardly controllable in a cloud environment (as an example, using the mOSAIC Riak COTS component, it depends on the VMs on which they run, and on the number of other components

TABLE 7.1
Mean Burst Response time (ms) varying the number of concurrent messages (nM) and message dimension (Dim)

Message Dim vs number of Messages	$nM = 1$	$nM = 10$	$nM = 100$
Dim=4Kb	25ms	164ms	2194ms
Dim=8Kb	32ms	331ms	5018ms
Dim=12Kb	25ms	542ms	14044ms

TABLE 7.2
Mean Burst Response time (ms) varying the number of concurrent sequences (nM) and file dimension (Dim)

Message Dim vs number of Messages	$nM = 1$	$nM = 10$	$nM = 100$
Dim=4Kb	92ms	529ms	6785ms
Dim=8Kb	100ms	1285ms	12345ms
Dim=12Kb	96ms	2006ms	19199ms

sharing CPU and memory). The goal of this benchmark is to create a simple repeatable evaluation that can be used to tune the application. Tests can be repeated (consuming resources) on-line, in order to perform a new evaluation in different conditions. We offer three different Benchmark applications: BenchmarkKvStore, BenchmarkKvStoreVariant, BenchmarkErrorKvStore that generate different loads on the KV store: the first one issues a (parameterized) sequence of SET/GET operations which are submitted concurrently to the KV store, while the second one performs GET operations only when all SET operations are completed. Both applications measure the overall response time and the mean number of SET/GET operations executed per second. The last one generates the same load of the first application, but checks the coherence of results (i.e., evaluates the effect of *eventual consistency* typical of KV stores).

Repeating these tests with a variable number of messages and repetitions helps the developer to understand how the KV performs in different conditions. Table 7.2 summarizes an example of results for the first benchmark on our testbed.

7.3. Benchmarking a Simple Cloudlet. Cloudlet benchmarking heavily depends on the target cloudlet to be evaluated. In any case it is possible to generalize the approach, considering that the majority of the cloudlets have similar behavior (for example: receive a message, make evaluations and send results, or receive a message, access a KV store. make evaluations and send results). We can classify this behaviors and produce target benchmarks of general use for each different pattern. At the state of the art, we focused on "Filter" cloudlets, which operate as filters, receiving messages, evaluating the content and generating a new message on the basis of evaluations. An alternative pattern is the one the accessing KV stores. In this case, the application proposed to benchmark KV store can be used as a template to generate the new custom benchmark application. Benchmarking Filter cloudlets needs the same control application adopted for Queues and it is possible to reuse the same client that automates the benchmark. Table 7.3 contains the results of such a benchmark on a typical cloudlet we use as test. This cloudlet analyzes XML documents and reports the result in an output message.

7.4. Full Application benchmarking. In this section we aims at showing how to benchmark a complete mOSAIC application, evaluating its behavior under well known stressing workloads. Differently from benchmarking mOSAIC resources (i.e. KV stores, Queues) or single cloudlets, a full application is composed of a lot of different components that interact in different ways and react in different ways to stressing conditions. Full Application benchmarking aims at identifying the application bottlenecks and critical points, in order to fine tune them on cloud environments.

Full Application benchmarking implies two different aspects:

- Evaluate the application as a whole under stressing loads which are known to the developer
- Evaluate the application components in order to understand how each of them behaves and reacts to different working conditions.

The Full Application can be evaluated using a benchmark application whose controlling part is the same of the Filter cloudlet, substituting the user interface (usually HTTPgw or mHTTPgw) with the controlling

TABLE 7.3
Mean Burst Response time (ms) varying the number of concurrent requests (nM) and file dimension (Dim)

Message Dim vs number of requests	$nM = 1$	$nM = 10$	$nM = 100$
Dim=4Kb	49ms	239ms	4659ms
Dim=8Kb	67ms	679ms	83595ms
Dim=12Kb	74ms	910ms	16435ms

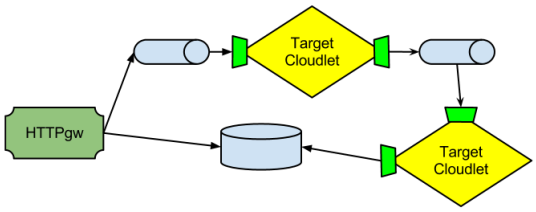


FIGURE 7.1. *A simple application to be benchmarked*

benchmark application. In this case the benchmark submits the stressing load to the application which generate its own results.

In order to evaluate the single components, thanks to the mOSAIC modularity, it is possible to extract each cloudlet of the application and test it independently (see benchmarking cloudlets), evaluating it under specific stressing condition.

In order to illustrate how the approach works, instead of detailing all the pieces of the framework, we will use a simple example, showing how to build a benchmark application starting from a mOSAIC application.

Figure 7.1 sketches the structure of the target mOSAIC application. For simplicity's sake, it is a toy application which receives messages from the mOSAIC web interface (through a queue). It performs an elaboration on messaages content, and forwards the result to a second application component which stores them in a Key-Value store.

Even if this application is very simple, its pattern can be adopted on many different real applications. In this paper our purpose is just to illustrate how to derive a benchmark application.

The first step of benchmarking procedure is to identify *what* to evaluate, the global application or single components of it. In this example, the core of the application is the *Target Cloudlet*, which performs the most complex work and may be a bottleneck of the application. The goal of the benchmark application will be to evaluate the conditions under which the component should be replicated, in order to grant application responsiveness.

As a result we can derive a new mOSAIC application, whose main architecture is described in figure 7.2. The new application is enriched with the GUI, the controller and a set of other components which produce a fixed, well known workload.

The derived application can be run, independently of the starting application, on any cloud provider through the mOSAIC Framework, collecting the results of execution in different working conditions.

As an example table 7.4 summarizes the results of the execution of the target cloudlet under the same requests, varying the number of machines acquired and the number of cloudlet instances started. This analysis will help a developer to evaluate the effects of application scalability.

One of the most interesting aspects is that the benchmarking application was developed, *without any additional code*. In fact, we just reused the set of components offered in mOSAIC, composing them in order to stress the target cloudlet.

8. Conclusions and Future Work. Cloud benchmarking is an hot topic, in which few effective solutions are available at the state of the art. In this paper we proposed a benchmarking framework, focused on the mOSAIC platform, which helps cloud application developers to build up custom benchmarking application, which can be used to compare different cloud provider offerings. We have illustrated the main concepts adopted in the framework and illustrated their adoption in different scenarios. We have shown that it is possible to

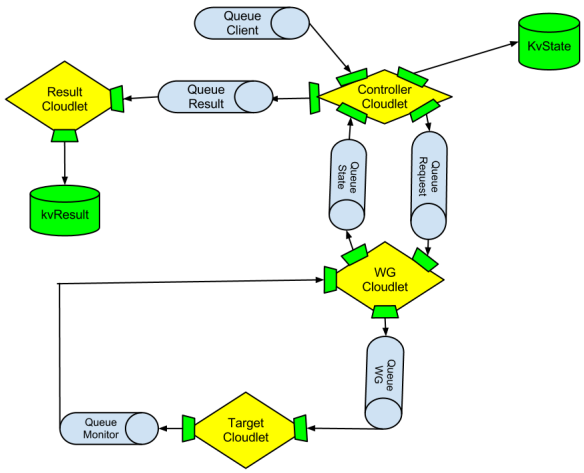


FIGURE 7.2. *The derived benchmarking application*

TABLE 7.4
Mean Cloudlet Response time (ms) varying the number of Virtual Machines (VM) and of Cloudlet instances (nC)

Cloudlet vs Virtual Machines	$nM = 1$	$nM = 2$	$nM = 3$
nC=1	122432ms	115803	119315
nC=2	72604	70890	76916
nC=3	84336	55423	54044

make comparisons among different cloud providers with a set of ad-hoc benchmarks that stresses the mOSAIC components of an application and helps in making a clear evaluation of the quality of the services offered through mOSAIC-based applications. The solution we proposed enabled both a fine grained analysis of single components of a mOSAIC application and an application-oriented comparison of Cloud provider offerings. Even if the focus of the paper and of the framework is on mOSAIC based application the approach (and the framework) can be easily adapted even for non-mOSAIC Applications. The Approach proposed (i.e. subdivision among benchmark control and benchmark model) can be applied as a general model for building custom benchmarks. Moreover the *Benchmarker*, i.e. the application which controls the benchmark life cycle and generate loads against the benchmark model, can be used easily even with non-mOSAIC application interfacing them through queue or through a dedicated component. In Future works we aims at applying such solution in order to control general puropose benchmarks and to compare providers even for evaluating non-mOSAIC uses.

REFERENCES

[1] *Cloudharmony available at.* `http://cloudharmony.com/`.
[2] *Cloudsleuth available at.* `https://cloudsleuth.net/`.
[3] *Linpack available at.* `http://www.netlib.org/linpack/`.
[4] *Nas parallel benchmark available at.* `http://www.nas.nasa.gov/publications/npb.html`.
[5] *Skampi available at.* `http://liinwww.ira.uka.de/~skampi/`.
[6] *Spec soa subcommittee.* `http://www.spec.org/soa/`.
[7] *Spec virtualization benchmark 2010.* `http://www.spec.org/virt_sc2010/`.
[8] *Standard performance evaluation corporation.* `http://www.spec.org/`.
[9] *Top 500 supercomputing sites available at.* `http://www.top500.org/`.
[10] *Tpc, transaction processing performance council available at.* `http://www.tpc.org/`.
[11] R. AVERSA, B. DI MARTINO, M. RAK, S. VENTICINQUE, AND U. VILLANO, *Performance Prediction for HPC on Clouds*, John Wiley & Sons, Inc., 2011, pp. 437–456.
[12] J. DEJUN, G. PIERRE, AND C.-H. CHI, *EC2 performance analysis for resource provisioning of service-oriented applications*, in Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing, Nov. 2009.
[13] J. DEJUN, G. PIERRE, AND C.-H. CHI, *Resource provisioning of web applications in heterogeneous clouds*, in Proceedings of the

2nd USENIX conference on Web application development, WebApps'11, Berkeley, CA, USA, 2011, USENIX Association, pp. 5–5.

[14] C. Evangelinos and C. N. Hill, *Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2*, Cloud Computing and Its Applications, Oct. 2008.

[15] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan, *Optimis: A holistic approach to cloud service provisioning*, 2012-01-01 2012.

[16] D. Kossmann, T. Kraska, and S. Loesing, *An evaluation of alternative architectures for transaction processing in the cloud*, in Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, SIGMOD '10, New York, NY, USA, 2010, ACM, pp. 579–590.

[17] A. Li, X. Yang, S. Kandula, and M. Zhang, *Cloudcmp: comparing public cloud providers*, in Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, IMC '10, New York, NY, USA, 2010, ACM, pp. 1–14.

[18] E. Mancini, M. Rak, and U. Villano, *Perfcloud: GRID services for performance-oriented development of cloud computing applications*, in 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE 2009, Groningen, The Netherlands, 29 June - 1 July 2009, Proceedings, S. Reddy, ed., IEEE Computer Society, 2009, pp. 201–206.

[19] C. Morin, *Open computing infrastructures for elastic services: contrail approach*, in Proceedings of the 5th international workshop on Virtualization technologies in distributed computing, VTDC '11, New York, NY, USA, 2011, ACM, pp. 1–2.

[20] D. Petcu, C. Craciun, M. Neagul, I. Lazcanotegui, and M. Rak, *Building an interoperability api for sky computing*, in 2011 International Conference on High Performance Computing & Simulation, HPCS 2012, Istanbul, Turkey, July 4-8, 2011, W. W. Smari and J. P. McIntire, eds., IEEE, 2011, pp. 405–411.

[21] D. Petcu, C. Craciun, and M. Rak, *Towards a cross platform cloud api - components for cloud federation.*, in CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science, Porto, Portugal, 18 - 21 April, 2012, F. Leymann, I. Ivanov, M. van Sinderen, and B. Shishkov, eds., SciTePress, 2011, pp. 166–169.

[22] D. Petcu, C. Crăciun, M. Neagul, S. Panica, B. Di Martino, S. Venticinque, M. Rak, and R. Aversa, *Architecturing a sky computing platform*, in Proceedings of the 2010 international conference on Towards a service-based internet, ServiceWave'10, Berlin, Heidelberg, 2011, Springer-Verlag, pp. 1–13.

[23] G. Pierre, I. El Helw, C. Stratan, A. Oprescu, T. Kielmann, T. Schütt, J. Stender, M. Artač, and A. Černivec, *Conpaas: an integrated runtime environment for elastic cloud applications*, in Proceedings of the Workshop on Posters and Demos Track, PDT '11, New York, NY, USA, 2011, ACM, pp. 5:1–5:2.

[24] M. Rak and G. Aversano, *Benchmarks in the cloud: The mosaic benchmarking framework*, in Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on, 2012, pp. 415–422.

[25] M. Rak, A. Cuomo, and U. Villano, *A service for virtual cluster performance evaluation*, in Proceedings of the 2010 19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE '10, Washington, DC, USA, 2010, IEEE Computer Society, pp. 249–251.

[26] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, O. Fox, and D. Patterson, *Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0*, 2008.

[27] E. Walker, *Benchmarking Amazon EC2 for high-performance scientific computing*, LOGIN, 33 (2008), pp. 18–23.

[28] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann, *C-meter: A framework for performance analysis of computing clouds*, in Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09, Washington, DC, USA, 2009, IEEE Computer Society, pp. 472–477.

Edited by: Marc Eduard Frîncu
Received: Mar 15, 2013
Accepted: Apr 8, 2013

DOI 10.12694/scpe.v14i1.826
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 1, pp. 47–55. http://www.scpe.org

A MODEL OF AUTOMATED NEGOTIATION BASED ON AGENTS PROFILES

SERBAN RADU, EUGENIA KALISZ, ADINA MAGDA FLOREA*

Abstract. In this paper we present a model of self-interested agents acting in an open environment, which captures the most relevant elements of agents' behaviour related to negotiation with other agents. The agent behaviour is mainly motivated by the gain they may obtain while fulfilling their goals and negotiating. Sometimes, the agents are also motivated by the necessity to cooperate with other agents for achieving their goals. The key element in the agent behaviour is their capability to develop a set of negotiation profiles: the preference profile, the partner cooperation profile and the group-of-partners negotiation profile; these profiles help the agents to conduct their negotiation.

Key words: agents, multi-agent systems, negotiation, strategy, negotiation profiles

AMS subject classifications.

1. Introduction. Negotiation is essential in settings where autonomous agents have a desire to cooperate, but also conflicting interests. Automated negotiation among intelligent agents has thus become increasingly important in applications that require computer supported decision making, like e-commerce, distributed resource allocation, or virtual enterprises.

The environments of such applications are inherently open, as they are populated with self-interested agents designed and/or owned by different people and there is no complete information about the preferences or decision-making processes of the participating agents. In order to be really autonomous and achieve performance when conducting a negotiation, an agent should be able to anticipate both the outcome of the negotiation and the best potential partner with which to start a negotiation. Machine learning approaches can contribute to adapt the agent's strategy during negotiation and trading, achieve better outcomes and increased payoffs.

A series of negotiation strategies and studies already exist in the literature, for example [1, 2, 3, 4, 5], in which the agents are able to choose different strategies. The automated negotiation process is usually taking place in open environments. These environments don't have a way to control the agents' behaviour and it is also possible to have humans in these environments, whose behaviour is unpredictable [4]. Some negotiation strategies are based on agent profiles, which can define statically or develop dynamically agents' preferences. There are advantages for creating agents negotiation profiles. Using these profiles, agents obtain better results than those in case of fixed negotiation strategies, i.e. increase the agents' gain from negotiation [6, 7, 8].

In this paper we present a framework for automated negotiation, based on negotiation profiles, and rules that encode the agents' negotiation strategy.

The set of negotiation profiles the agents are able to evolve consists of: the preference profile, which specifies the agent negotiation strategy, the cooperation profile, which keeps track of the agent interaction with the other agents in the system, and the group-of-partners negotiation profile, which clusters the profiles of several agents.

The outcome of negotiation is evaluated for different strategies, encoded in the preference profile. In the system there are different agents and a facilitator, which can be used by the agents for registering their capabilities and optionally for facilitating the negotiation.

The three types of negotiation profiles are described and discussed, as well as the uninformed and informed negotiation strategies. Negotiation strategies are implemented in the form of production rules. In our approach, preference coefficients can be assigned to these rules and dynamically modified, according to the negotiation situation.

The current system is under development and has been tested on some simple cases of electronic transactions. When completed, the system is aimed to support general e-commerce transactions.

The paper is organized as follows. Section 2 presents the agent framework for automated negotiation, Section 3 presents the agent structure, and Section 4 details the preference profile and different agent strategies for negotiation. Section 5 presents related work and Section 6 contains conclusions and several future developments.

2. Framework for Automated Negotiation. The negotiation agent behaviour that we propose is defined in a framework of a multi-agent system in an open environment. The system includes the facilitator,

*Department of Computer Science, University Politehnica of Bucharest, Bucharest, Romania (serban.radu, eugenia.kalisz, adina.florea@cs.pub.ro).

which can be used or not by the agents during negotiation, depending on the preference profile. Because the environment is open, other agents can enter or exit the system dynamically, cf. Fig. 2.1. When entering the system, the agent has to register with the facilitator, and the facilitator will inform other agents about the new agent arrival.

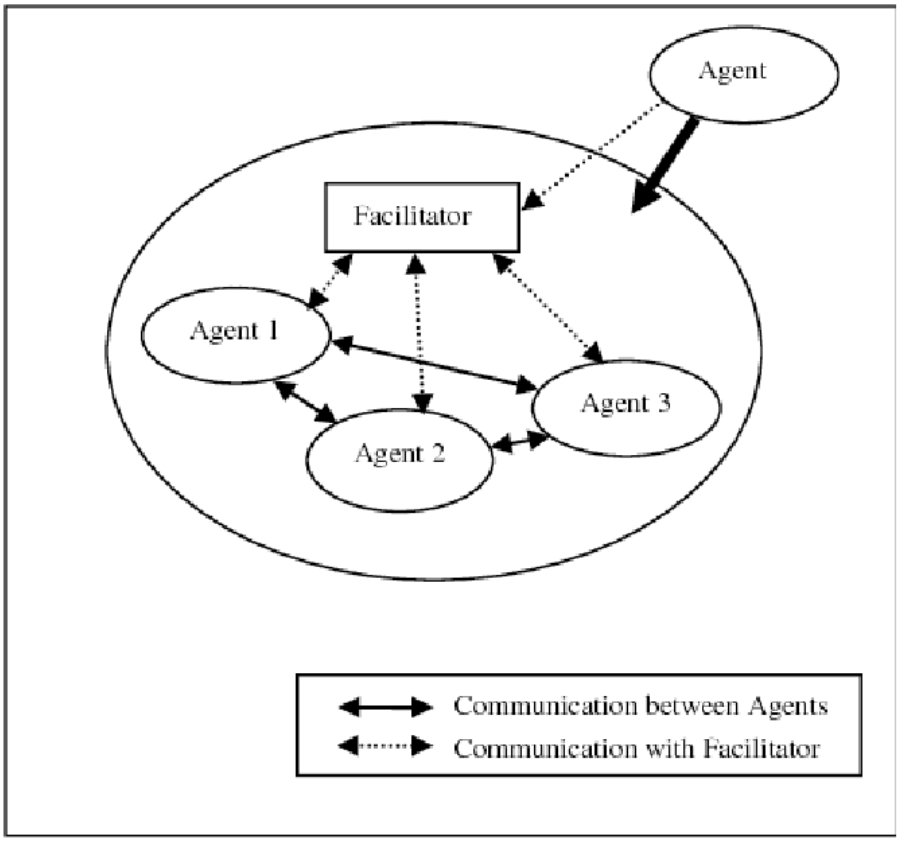


FIGURE 2.1. *The MAS Open Environment.*

When an agent wants to negotiate, it sends a message to the facilitator, asking it to find other agents appropriate to negotiate with. In this message, the agent sends to the facilitator two types of information: a part of the message presenting its abilities, for example which tasks it can do, which objects it possesses, etc., and the other part of the message asking help for a negotiation, e.g., to buy or sell an item. It is the facilitator which will query the agents with the required abilities, and will inform the agent about the outcome. The advantage of this approach is that not all the agents are queried, but the disadvantage is that the facilitator has a lot of messages to process and can become a bottleneck in the system.

The negotiation between agents is a single issue negotiation, where the price of the products is bargained. The type of negotiation implemented is a heuristic negotiation, in which the agent computes the gain, as compared to its private value for an object. The agents negotiate following the Contract Net protocol. Each agent takes into account a private value for any item (product or service) to trade (sell or buy). The Contract Net protocol may be applied conveniently in B2B models, where one company wants to acquire a product or service (and becomes the initiator of the protocol) and also in some specific cases of B2C where the initiator is the seller of a service he wants to be promoted.

There is an upper limit for the number of negotiation rounds. In a buying negotiation, an agent will look for a lower value than its private one, while in a selling negotiation, its main goal is to obtain more than the items private value. The private value is established by the internal reasoning mechanism of the agent and depends on the specific domain.

The agent behaviour is mainly motivated by the gain, but also, depending on a specific context, by the desire to achieve cooperation with other agents. For example, as a general rule, it will not accept a price lower than its private value. However, when the agent wants to cooperate, it can accept a lower price. The agent behaviour is set up by the negotiation strategy.

The negotiation model we propose in this paper can be easily extended to multi-issue negotiation. In this case, the agent will trade the negotiation object (**NO**). A negotiation object is the range of issues over which agreements must be reached, as defined in [5]. The object of a negotiation may be an action which the negotiator agent A asks another agent B to perform for its benefit, a service that agent A asks to B, or, alternatively, an offer of a service agent A is willing to perform for B, provided B agrees to the conditions of A. An agent A may have a plan to achieve one of its desires, but may be unable to carry out some of the necessary actions, therefore it will ask B to execute these actions for it, if it believes the actions belongs to B's abilities [7]. The agent A may ask agent B to perform a service for him, for example to paint its house, or A may be a communication company offering to an agent B a competitive long distance calls service.

3. Agent Structure. The agents have different reasoning capabilities, as specified above, designed to conduct successful negotiation and the fulfillment of agents' goals. Any agent from the system uses a set of behaviour rules, which defines how the agent fulfills the goals and carries on the tasks assigned to it, and a set of rules, which defines its negotiation strategies, cf. Fig. 3.1. During negotiation, the agents gather information about the partner agents, and store it in the associated cooperation profiles.

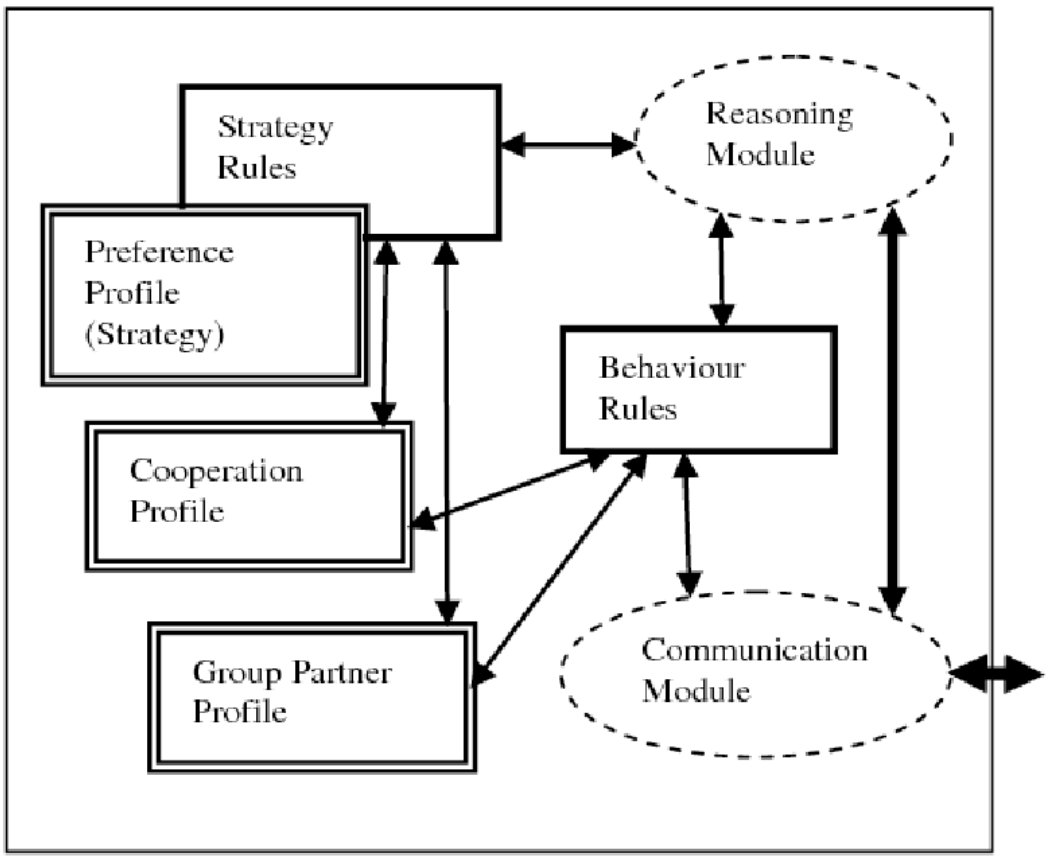


FIGURE 3.1. *The Agent Main Components.*

Each agent is endowed with a set of negotiation profiles, cf. Fig. 3.1:

- **the preference profile**, which specifies the agent negotiation strategy;

- **the partner cooperation profile**, which keeps track of the agent interactions with the other agents in the system;

TABLE 3.1
Agent Cooperation Profile.

Agent Name	No. Negotiations with Partner	No. Successful Negotiations	Gain
John	5	3	2
Gain Percent	No. Negotiation Rounds	Interest Degree of Partner	Classification
40	6	3	cooperative

- **the group-of-partners negotiation profile**, which deals with a group of negotiation partners.

The partner cooperation profile describes the preferences regarding the agents with which an agent prefers to cooperate. The cooperation profile of an agent is implemented as a structure which evolves dynamically. For each agent encountered during the negotiation process, knowledge about the outcome of different negotiations is stored in this structure.

The partner cooperation profile is a characteristic of each agent and is stored as a matrix. Each row contains the agent name and a set of associated attributes, upon which the agent dynamically changes its preferences. The partner cooperation profile is updated during the negotiation process, at the end of each negotiation. Table 3.1 presents an example of partner cooperation profile for an agent.

An entry in the matrix, defining the cooperation profile for an agent, has the fields presented below, representing the cooperation attributes.

1. The first field represents the name of the partner agent, stored as a string of characters.

2. The second field contains how many times the agent negotiated with its partner, and is represented as a natural number.

3. The third field stores the number of successful negotiations, as an integer number.

4. The fourth field points out the connection between the outcome of negotiation and the private value of the agent for the negotiation object, showing the total gain obtained. This is stored as a positive or negative integer number.

Based on heuristic criteria, an agent can accept a price greater than its private value in order to buy a product, or can sell a product at a price lower than its private value. These decisions are based on the information stored in the agent cooperation profile and are specified by the strategy rules.

5. The fifth field shows the gain percentage, namely how much is the gain obtained while negotiating with the partners, as a percentage of the agent total gain.

6. The sixth field is the number of negotiation rounds, during the last negotiation, and is an integer number.

7. The seventh field tries to capture the agents' beliefs about the partner abilities and/or credentials. We call this attribute the interest degree of the partner, which is quantified as: very interesting, interesting, moderately interesting, and not interesting (4...1). For example, if the partner has an ability to perform a task, which is lacking to the agent, then the partner is interesting or very interesting to the agent. Moreover, if the negotiation is successfully concluded in a small number of steps, and the gain is positive, then the partner is very interesting. This field is either set in the beginning by the designer if the partner agent is known, or it is computed based on the agent abilities using a function provided by the designer.

8. The eighth field contains the classification of the partner agent, which represents the current agent belief about the cooperation potential of the partner. The partners are classified in six cooperation classes: highly cooperative, very cooperative, cooperative, slightly cooperative, non cooperative and unknown.

The attributes 1 to 7 described above are updated by the agent after each negotiation with a specific agent. The last attribute (8) will be filled in by a more elaborate process, to be described further on.

The representation of the agents' cooperation profiles as a set of attributes is able to characterize the cooperation potential of a partner, with a high degree of granularity. We would like to be able to characterize the cooperation potential of a partner agent in a broader sense; to this aim, we proposed the clustering of partners into cooperation classes, specified in field 8. We achieve this classification by using a C4.5 learning algorithm [9], in which the fields from 2 to 7 are used as classification attributes and field 8 represents the class. We can have two approaches to generate training examples for the learning algorithm. In the first approach, we let the system run and collect gathered information. In the second approach, we generate a training set of virtual agents, with which an agent has virtual negotiations. After these virtual negotiations are performed, the matrix is filled with the results of these negotiations, according to the fields described before.

The information stored in this way in the matrix will allow the classification of real negotiations. The

classification is improved, as more negotiations take place and the matrix is filled with the correct real values. After that, the C4.5 algorithm will classify correctly the negotiation instances.

Some premises guide the C4.5 algorithm, such as the following:

- if all cases are of the same class, the tree is a leaf and so the leaf is returned labeled with this class;

- for each attribute, compute the potential information provided by a test on the attribute, based on the probabilities of each case having a particular value for the attribute. Also, compute the gain in information that would result from a test on the attribute, based on the probabilities of each case with a particular value for the attribute being of a particular class;

- depending on the current selection criterion, find the best attribute to branch.

4. Agent Behaviour. The preference profile of an agent describes its strategy. This is represented as a set of negotiation rules. These rules can be heuristic negotiation rules, which refer to cooperation profiles, or rules which encode certain negotiation strategies.

The negotiation strategy of an agent may show how much and how quickly the prices are decreased and if the price is lower than the agent's private value. An agent can use a tradeoff. In the case of cooperative agents, an agent can gain more once, and then can sell cheaper. This is a kind of global evaluation on previous deals.

The negotiation strategy is implemented in the form of production rules, each agent keeping a history of its interactions. If in a given situation several rules are eligible, then the negotiation strategy decides which rule from the conflicting set to apply. A possible approach to solve the conflicts is to assign priorities between rules. The solution is to apply the rule with the highest priority. Using a feedback, it is possible to apply the same rule or another rule.

The rule's priorities are dynamically modified, and the preference coefficients are not built-in, they are dynamically changed, according to the negotiation situation. The preference coefficients are updated, using the Q-learning algorithm. Each preference coefficient is indexed on a state and an action. The states from the Q-learning algorithm represent the internal states of an agent and the actions are described by the rules applied by an agent when negotiating.

The group of partner negotiation profile is defined by grouping into classes the partner agents, with which the agent interacted in the system. For each of the six values of the cooperation classes, the group negotiation profile contains the list of all agents that belong to a given class. The agents, for which no class was found out yet, belong to the unknown class. During the interaction with other agents, using the C4.5 algorithm, the agents are classified in the appropriate class; moreover, following a negotiation, an agent may migrate from one class into another.

There is a tradeoff between how often we update the agent classes, which may be costly, and the accuracy of classification.

The preference profile is aimed to express the negotiation strategy, more specifically to establish the priority among the negotiation rules.

Each rule has an associated preference coefficient (**PC**), which indicates, in case two or more rules apply in a given situation, which rule should be preferred. In this way, the strategy can be explicitly set up by the system designer by specifying the values of PC when writing the rules, and can be changed from one use of the system to another, simply by changing these coefficients. Another use of these coefficients is that they can be dynamically modified by a reinforcement learning algorithm, in a similar manner to a learning classifier system [10]. Learning classifier systems are a machine learning technique that may be categorized in between symbolic production systems and sub-symbolic connectionist systems.

There are two categories of rules used in the strategy. The first category of rules is used in the beginning of the negotiation and represents **uninformed strategy rules**. The second category of strategy rules is used when there is enough information about the negotiation partner and represents **informed strategy rules**. In this part of the negotiation, it is also possible to apply the uninformed strategy rules. Depending upon the success or failure of the negotiation, the preference coefficients of the strategy rules are changed accordingly.

It is possible to have more than one strategy in the system, for instance, at each negotiation step, the price is decreased by 1, or is decreased by 3. Another strategy rule tells what happens when the price increases with 10 % above the private value, if the offer is instantly accepted or not.

In order to show an example of strategy rules, we consider the following Contract Net protocol:

a) ***cfp(A, X, NO, P)*** is the communication primitive, which represents a call for proposals from agent A to all the acquaintances X, regarding a negotiation object NO, with an associated cost P

b) ***propose(X, A, NO, P, Step)*** is the communication primitive, which represents the response of agent X to the *cfp*, with the negotiation object NO, price P and negotiation step *Step*

c) ***accept(X, NO)*** indicates the acceptance of a proposal issued by X, for the NO

d) ***reject(X, NO)*** indicates the rejection of a proposal issued by X, for the NO

e) ***counterpropose(A, X, NO_1, P_1, Step)*** defines a communication primitive, which represents the counterproposal of agent A to the proposal of agent X, with the negotiation object NO_1, price P_1 and negotiation step *Step*.

As in any other Contract Net protocol, agents may be either buyer or sellers, and the roles can be interchanged.

We consider a Prolog-like language and a set of predicates, defined as follows:

- *propose(X, A, NO, P, Step)* defines a predicate, which is true when agent A receives the response *(b)* of agent X to the *cfp*, with the negotiation object NO, price P and negotiation step *Step*

- *accept(X, NO)* defines a predicate which, when true, triggers an acceptance message *(c)* of a proposal issued by X, for the NO

- *reject(X, NO)* defines a predicate which, when true, triggers a rejection message of a proposal issued by X, for the NO

- *counterpropose(A, X, NO_1, P_1, Step)* defines a predicate which, when true, represents the counterproposal *(e)* of agent A to the proposal of agent X, with the negotiation object NO_1, price P_1 and negotiation step *Step*

- *tp(Ag_Name, Atr_Name, Value)* is a predicate which selects from the cooperation profile, for a given agent name *(Ag_Name)*, the value *(Value)* of the attribute *(Atr_Name)* in the associated field.

Considering the above described predicates, an example of an uninformed strategy rule is:

propose(John, Tom, House, 1000, 1),
private_value(House)=p, p≥1000 → accept(John, House)

and some examples of informed strategy rules of an agent are given below:

r_1 :
- *propose(John, Tom, House, 1000, S),*
  tp(John, No_Successful_Negotiations, v_1),
  tp(John, No_Negotiations_with_Partner, v_2), $v_1 > v_2 - 2$,
  tp(John, Interest_Degree_of_Partner, v_3), $v_3 > 3$,
  tp(John, Gain_Percent, v_4), $v_4 > 20$
  $\rightarrow$ *accept(John, House)* PC_1

r_2 :
- *propose(John, Tom, House, 1000, S),*
  tp(John, No_Successful_Negotiations, v_1), $v_1 > 5$,
  tp(John, Gain, v_2), $v_2 > 100$,
  tp(John, Interest_Degree_of_Partner, v_3), $v_3 = 4$
  $\rightarrow$ *accept(John, House)* PC_2

r_3 :
- *propose(John, Tom, House, 1000, S),*
  tp(John, No_Successful_Negotiations, v_1),
  tp(John, No_Negotiations_with_Partner, v_2), $v_1 < 0.5 * v_2$,
  tp(John, Interest_Degree_of_Partner, v_3), $v_3 < 2$
  $\rightarrow$ *reject(John, House)* PC_3

r_4 :
- *propose(John, Tom, House, 1000, S),*
  tp(John, No_Successful_Negotiations, v_1),
  tp(John, No_Negotiations_with_Partner, v_2), $v_1 < 0.5 * v_2$,
  tp(John, Interest_Degree_of_Partner, v_3), $v_3 > 3$
  $\rightarrow$ *counterpropose(Tom, John, Car, 500, S+1)* PC_4

r_5 :
- *propose(John, Tom, House, 1000, 1),*
  tp(John, Classification, 'highly cooperative')
  $\rightarrow$ *accept(John, House)* PC_5

r_6 :

- *propose(John, Tom, House, 1000, S),*
  tp(John, Classification, v_1*),* $v_1 ='$ *unknown',*
  Price $\geq$ *private_value(House)*
  $\rightarrow$ *reject(John, House)* PC_6

r_7 :

- *propose(John, Tom, House, 5000, 1),*
  tp(John, Classification, v_1*),* $v_1 ='$ *unknown',*
  private_value(House) $= p, p \leq 5000$*,*
  tp(John, Gain, v_2*),* $v_2 = 5000 - p$
  $\rightarrow$ *accept(John, House),*
  tp(John, No_Negotiations_with_Partner, 1),
  tp(John, No_Successful_Negotiations, 1) PC_7

During negotiation steps, the C4.5 learning algorithm can classify the partner in another cooperation class, if the criteria used as attribute in the algorithm are changed.

So, instead of having a negotiation strategy which applies all the rules that match at a certain moment, the classification with the C4.5 learning algorithm decreases the number of eligible rules.

5. Related Work. The work reported here is connected to our previous developments of a multi-agent system framework for negotiation. In [7], we have presented a system of self-interested agents that are endowed, besides the widely accepted beliefs-desires-intentions notions, with goals, preferences, obligations, and norms. The inference rules that guide negotiation are based both on cost and gain, and on the cooperation profile the agent develops during previous interactions with other agents in the system. However, the cooperation profile is unique and considers another set of attributes, than the ones in our current model. In [7], we have defined several types of agents, by varying their behaviour according to obligation compliance, self-interestedness, desire to develop good cooperation relations with other agents in the system or simply desire to obtain the maximum gain.

In [8], we have reported on a model of heuristic negotiation between self-interested agents, which allows negotiation over multiple issues of the negotiation object, comprises different types of negotiation primitives, including argument based ones, and a set of rules to conduct negotiation. In order to negotiate strategically and to adapt negotiation to different partners, the agents use rewards associated to negotiation objects and the notion of regret to compare the achieved outcomes with the best possible results that could have been obtained both in a particular negotiation and in selecting the partner agent.

The decision process during the negotiation is modeled as an adversarial bandit problem with partial information and uses the computed probabilities of negotiation rules to select the best rule to be used at a certain moment during negotiation. Rewards were defined depending on the attributes of the negotiation object at a given negotiation round or, in case adaptation of partner selection is sought, depending on the negotiation object with which the negotiation is concluded. Moreover, we have shown how the problem can be modeled if not all rules can be selected at a given decision point (equivalent with not all experts being available for consultation).

In [11], the authors apply the Q-learning algorithm to analyze and learn customer behaviours and then recommend appropriate products. As compared to our approach, the user profile is not used for negotiation, but to personalise the information to the user interests. The authors use weighting features to recommend products to the user; we use weights to represent the preference coefficients.

In [12], the authors propose a software framework for negotiation, in which the negotiation mechanism is represented by a set of rules, as in our case. The rules are organized in a taxonomy, and can be used in conjunction with a simple interaction protocol; the negotiation language is based on OWL-Lite. Although the rules allow flexible definition of several negotiation strategies, there are no negotiation profiles and the possibility to modify the negotiation, according to these profiles, as in our case.

An implementation of automated negotiations in an e-commerce modeling multi-agent system is described in [13]. A specific set of rules is used for enforcing negotiation mechanisms. An experiment involving multiple English auctions performed in parallel is discussed.

A system for automated agent negotiation, based on a formal and executable approach to capture the behaviour of parties involved in a negotiation is shown in [14]. The negotiation strategies are expressed in a declarative rules language, defeasible logic, and are applied using the implemented system Dr-Device.

In [4], the authors present a system, called GENIUS (General Environment for Negotiation with Intelligent multi-purpose Usage Simulation), that supports the design of different strategies for agent negotiation, and the evaluation of these strategies in a simulated environment. The system allows the negotiation between automated agents, but also between agents and humans. The designer of a strategy can select from a repository a negotiation domain and a preference profile for the agent. Both are represented in a tree-like structure, which enables to specify priorities related to outcomes of negotiation. As compared to this system, in our approach, there are several negotiation profiles, which are evolved during interactions, and the negotiation domain is specified by the agent rules.

6. Conclusions and Future Work. In this paper, we have presented a model of negotiating agents that aimes to combine the agents' beliefs about the other agents in the system, with the possibility to explicitly represent and modify the negotiation strategy, expressed in a set of rules. In order to achieve this, we have defined three negotiation profiles: the preference profile, the partner cooperation profile and the group-of-partners negotiation profile. The last two, partner and group-of-partners, are profiles developed during interactions and they are gradually built, as the agent is taking part to more and more negotiation rounds. The group profile is obtained by applying C4.5 algorithm, and allows the classification of negotiation partners in different classes. Once these classes are available, the agent can decide much quicker on the behaviour to adopt, regarding a partner agent, than in case of the single preference profile.

The negotiation strategy is explicitly represented as a set of rules, together with the preference coefficients, associated to the rules. The preference profile is formed of these coefficients, which can be fixed by the system designer or can be evolved, using a reinforcement-like algorithm.

The behaviour of the agents is motivated by the gain they obtain when realizing their goals or by the necessity to cooperate with other agents, in order to achieve these goals. During negotiation, the agents beliefs on the other agents are updated, as the agent comes to know more about the others.

Because the agents' preferences are based on their interests, the preference coefficients can be modified in time. The agents can modify their preferences over negotiation outcomes, when receiving new information.

The current system is under development and has been tested on some simple cases of electronic transactions. The system prototype has been implemented in Jade; the preliminary results have shown good performances.

The work reported here is an extension of our previous research presented in the Workshop on Agents for Complex Systems (ACSys 2012), Timisoara, Romania.

A future research direction is to use an alternate approach to develop group profiles. Namely, the k-means clustering algorithm, which will remove the necessity to determine in advance the cooperation classes.

Another future line of work is to investigate an alternate approach to update the preference coefficients. This may be done using genetic algorithms. Moreover, genetic algorithms that use rule-specific genetic operators can be used to evolve new strategy rules, based on the existing ones.

Testing the system with a large number of agents and in a real-world environment is the next future work.

Acknowledgments. This work was supported by the project ERRIC No. 264207, FP7-REGPOT-2010-1.

REFERENCES

[1] T. Ito, H. Hattori, and M. Klein, *Multi-issue negotiation protocol for agents: Exploring nonlinear utility spaces*, in Proceedings of the 20th International Joint Conference on Artificial Intelligence, 2007, pp. 1347–1352.

[2] C. Jonker, V. Robu, and J. Treur, *An agent architecture for multi-attribute negotiation using incomplete preference information*, Autonomous Agents and Multi-Agent Systems, 15 (2007), pp. 221–252.

[3] R. Lin, Y. Oshrat, and S. Kraus, *Investigating the benefits of automated negotiations in enhancing peoples negotiation skills*, Proceedings of the 8th International Conference on Automated Agents and Multi-agent Systems, 2009, pp. 345–352.

[4] R. Lin, S. Kraus, J. Wilkenfeld, and J. Barry, *Negotiation with bounded rational agents in environments with incomplete information using an automated agent*, Journal of Artificial Intelligence, 172 (2008), pp. 823–851.

[5] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge, *Automated negotiation: prospects, methods and challenges*, International Journal of Group Decision and Negotiation, 10 (2001), pp. 199–215.

[6] S. Radu and A. M. Florea, *An adaptive multi-agent system for e-commerce*, Proceedings of Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems, 2012, pp. 297–300.

[7] A. M. Florea and E. Kalisz, *Conceptual models of multi-agent systems*, Proceedings of the 14th International Conference on Control Systems and Computer Science, 2003, pp. 394–399.

[8] A. M. Florea and E. Kalisz, *Adaptive Negotiation Based on Rewards and Regret in a Multi-agent Environment*, IEEE Computer Society Press (CPS), 2007, pp. 254–259.

[9] J. Quinlan, *C 4.5: Programs for machine learning*, Morgan Kaufmann, 1992.

[10] M. V. Butz, *Learning classifier systems*, Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation, 2010, pp. 2331–2352.
[11] A. Srivihok and P. Sukonmanee, *E-commerce intelligent agent: personalization travel support agent using Q Learning*, Proceedings of the 7th International Conference on Electronic Commerce, 2005, pp. 287–292.
[12] C. Bartolini, C. Preist, and N. R. Jennings, *A software framework for automated negotiation*, Proceedings of the 3rd International Workshop on Software Engineering for Large-Scale Multi-Agent System, 2005, pp. 213–235.
[13] C. Badica, A. Badita, and M. Ganzha, *Implementing rule-based mechanisms for agent-based price negotiation*, Proceedings of the 2006 ACM Symposium on Applied Computing, 2006, pp. 96–100.
[14] T. Skylogiannis, G. Antoniou, N. Bassiliades, G. Governatori, and A. Bikalis, *Dr-Negotiate A system for automated agent negotiation with defeasible logic-based strategies*, Journal Data and Knowledge Engineering, 63 (2007), pp. 362–380.

Edited by: Marc Eduard Frîncu
Received: Jan 13, 2013
Accepted: Apr 6, 2013

DOI 10.12694/scpe.v14i1.827
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 1, pp. 57–67. http://www.scpe.org

NEW PERFORMANCE ESTIMATION FORMULA FOR EVOLUTIONARY TESTING OF SWITCH-CASE CONSTRUCTS

GENŢIANA IOANA LAŢIU, OCTAVIAN AUGUSTIN CREŢ, AND LUCIA VĂCARIU*

Abstract. Evolutionary structural testing is a technique that uses specific approaches based on guided searches algorithms. It involves evaluating fitness functions to determine whether test data satisfy or not various structural testing criteria. For testing multi-way decision constructs the nested *If-Then-Else* structure and Alternative Critical Branches (ACBs) approaches are generally used. In this paper a new evolutionary structural approach based on Compact and Minimized Control Flow Graph (CMCFG) which uses two different formulas for evaluating the performance of test data, is presented. The CMCFG approach is derived from the concept of Control Flow Graph (CFG). Experiments on different *Switch-Case* constructs with different nesting levels have demonstrated that CMCFG yields significantly better results in finding test data which cover a particular target branch in comparison with the previous approaches.

Key words: Evolutionary structural testing, control flow graph, switch case structures, fitness function.

AMS subject classifications. 68N15, 68N19, 68N01

1. Introduction. The main idea behind the evolutionary testing process is to automatically generate test data through the use of optimizing search techniques [1]. The search space which corresponds to the evolutionary process is represented by the specific domains of the input variables of the software program under test. Evolutionary structural testing has been intensively used for automatically generating test data by many researchers. M. Harman and P. McMinn present in [9] a vast theoretical exploration of global search techniques embodied by Genetic Algorithms. Other approaches related to evolutionary testing with flag conditions are presented in [5], [16], and [4]. Different transformations techniques were applied and reported in the literature for Evolutionary Testing (ET) in order to improve the fitness function calculation, because a well-defined fitness function is essential for the efficiency of the evolutionary search process ([12], [8], and [13]).

The main constructs (sequence structures, selection structures and repetition structures) of a software program were studied and tested in the literature using evolutionary search techniques, but less work has been done on the *switch-case* constructs which are used to express multiple branch selection statements. This type of construct was studied in [15], where it was tested using the concept of Alternative Critical Branches (ACBs). ACBs consist of all case branches that can prevent the execution of the target branch. The ACBs consist of one element that is the alternative branch of the target branch if it is leaving a two-way decision node. Each control-dependent node has only one ACB assigned to it. All the ACBs with respect to the target branch constitute a set. It forms the Critical Branches Set (CBS) which is extended from the single critical branch concept. This concept refers to the branch which prevents the target branch to be reached when the current test data is executed. If any element from CBS corresponding to the target branch is taken, then there is no chance to generate test data which cover the target branch.

The focus in this approach is on the structural testing of multi-way decision statements, in particular on branch coverage. The ACBs are used for determining the approximation level, which is used by the fitness function formula that evaluates the performance of each individual. The approximation level has been calculated by subtracting one from the number of ACBs which are in the CBS and which are lying between the node from which the test data diverge away and the target node.

The rest of this paper is organized as follows: Section II describes the evolutionary testing methodology and the switch-case constructs. Section III describes different fitness function calculation approaches used for structural testing in case of switch-case constructs. Section IV presents the experimental results obtained for different level of imbrications and Section V presents the final conclusions and future work.

2. Evolutionary testing methodology and switch case construct. Evolutionary testing (ET) is a meta-heuristic approach by which test data can be generated automatically through the use of optimization search techniques. It is usually used for testing complex systems which may involve many components with correlated activity between them. The ET process tries to improve the effectiveness of the traditional testing process by transforming the testing objectives in search problems which will be solved using evolutionary

*Technical University of Cluj-Napoca, Computer Science Department, 26-28 George Baritiu street, 400027 Cluj-Napoca, Romania

algorithms. In the ET process the search space is represented by the variation domain of the input variables of the software under test, in which test data fulfill the specific test objectives. The ET process' phases are presented in Fig. 2.1:

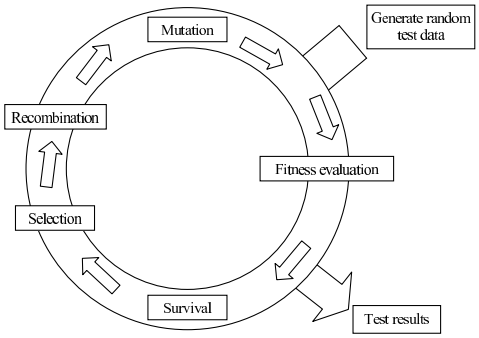


FIGURE 2.1. *Evolutionary testing process*

ET is used in many search problems in software testing, because it has a very good capacity of adapting itself to the system under test. The ET process is an iterative procedure which combines good test data in order to achieve better test data. ET was successfully reported in the literature and applied for different forms of testing, called: specification testing [14], unit testing [7], and extreme execution time testing [17].

During the ET process the initial test data are randomly generated. Each individual from the population represents the test data using which the test is executed. Test data take values from the domains of the software under test's input variables. The performance of each individual is evaluated and the fitness value corresponding to the current individual is determined. Next, the population members are selected with respect to their performance. The chosen individuals are subject to *crossover* and *mutation* processes to generate new individuals, called *offsprings*. *Crossover* is used to combines two parents to produce a new offspring, while *mutation* is used for randomly altering a gene value (for instance, switching from 1 to 0 in case of binary individuals) from the individual. By reuniting the new created offsprings with their parents a new population is formed. The evolutionary process repeats all the above described steps until the established testing criteria are met. Then the process stops and the best solution found will be the testing solution.

The goal of this research was to study the *switch-case* construct in the context of structural path oriented testing, aiming to find test data which executes a particular branch in a program that contains *switch-case* constructs. In order to automatically generate test data which trigger the execution of a particular branch of the program, every possible solution is evaluated with respect to the test objective. The *switch-case* construct is a multi-way selection control mechanism which is used as a substitute for the nested *if-then-else* structure. It is extensively used in software programs because it improves the readability of the source code and reduces repetitive coding. The general structure of a *switch-case* construct is presented in Fig. 2.2:

```
Switch (expression) {
Case expression: //some code
        Jump, return or break statement
Default: //some code
        Jump, return or break statement
}
```

FIGURE 2.2. *General switch-case conditional construct*

The *switch-case* construct, as presented in 2.2, gives the developer the possibility of choosing between many statements, by passing the flow control to one of the *case* statements within its body. The *switch* statement evaluates the expression and executes the *case* branch that corresponds to the expression's value. A *switch-case* construct can include any number of *case* statements. Each *case* statement is followed by an optional *break*, *return* or *goto* statement (called breaking statements). The breaking statements are used either to return a

value and exit the *switch* body, or to break out of the *switch* construct when a match is found, or go to a specific location in the code.

If *break*, *return* and *goto* options are not present after a *case* statement then the control flow is transferred to the next *case* statement until it will meet one of the breaking statements. If an expression transmitted to the *switch-case* construct does not match any *case* statement, the control will go to the *default* statement. If no default statement exists, the control will go outside the *switch* body. A simple example of a *switch-case* construct is presented in Fig. 2.3.

```
Switch (x) {
        Case 'b': y = 'B'; break;
        Case 'f': y='F'; break;
        Case 'c': y='C'; break;
        Case 'a': y='A'; break; //Target branch
        Default: y='Z';
        }
```

FIGURE 2.3. *Simple switch-case conditional construct*

A previous work [15] has argued that for a particular case branch, the CBS should be constructed. This set is composed of all the case branches that cause the target to be missed. For instance, the CBS that corresponds to the target branch from the source code listed in Fig. 2.3 is composed by *case* 'b', *case* 'f', *case* 'c, and *default*. The target branch is definitely missed when the execution of test data diverges away at any branch within the CBS.

The fitness function used for evaluating each test data is calculated using the sum between two metrics:

1. The *approximation level*. This is calculated by subtracting 1 from the number of ACBs located between the node from which the test data diverge away and the target branch itself (in the example from Fig. 2.3, the branch that corresponds to case 'a').
2. The *branch distance*. This is calculated using the following expression: $|expr - C| + 1$, where *expr* is the value of the expression which appears after the *switch* keyword, and C is the constant value for the desired *case* statement. For both operands (*expr* and C) the corresponding ASCII code for each character is used. The value 1 which appears at the end of the formula is the positive failure constant [14]. For instance, if $x =$ 'f', then the branch distance which corresponds to the target branch specified in Fig. 2.3 is $|102 - 97| + 1$.

The fitness value indicates how close the test data are to trigger the execution of the target branch located inside the *switch* statement.

3. Different fitness function calculation approaches for switch-case constructs.

3.1. Fitness calculation based on nested if-then-else statements. *Switch-case* constructs are considered to be equivalent to nested *if-then-else* statements with respect to the CFG. The *switch-case* construct presented in Fig. 2.3 is equivalent to the nested *if-then-else* construct shown in Fig. 3.1:

```
if (x=='b') {y='B';}
else if (x=='f') {y='F';}
else if (x=='c') {y='C';}
else if (x=='a') {y='A';} // Target branch which should be tested
else {y='Z';}
```

FIGURE 3.1. *Transformation of switch-case conditional construct in nested if-then-else statements*

The target branch for which test data should be generated is the case branch x=='a'. Each test data automatically generated by the ET process must be evaluated using the fitness function. The purpose of this function is to guide the evolutionary search process to find the test data that trigger the execution of the target branch. The fitness function evaluation represents the calculation of distances to the target branch.

In structural testing, previous work [2] has demonstrated that the fitness function having the expression illustrated in 3.1 correctly evaluates how close the test data are to cover the target branch:

$$F(test\_data) = Approx\_level + Normalized\_branch\_distance \tag{3.1}$$

The normalized branch distance is computed using formula 3.2 and indicates how close the test data are to take the alternative branch:

$$Normalized\_branch\_distance = 1 - 1.001^{-distance} \tag{3.2}$$

The *approximation level* represents the number of decision nodes lying between the decision nodes where the actual test data diverge away from the target branch itself. In Fig. 3.2, given x = 'b' the control flow takes the Yes branch at node 1. The *approximation level* is 3. The branch distance is computed according to (2) using the values of the variables or constants involved in the conditions of the branching statement [2]. For the branching condition x ='b' the branch distance is $|x - 98|$.

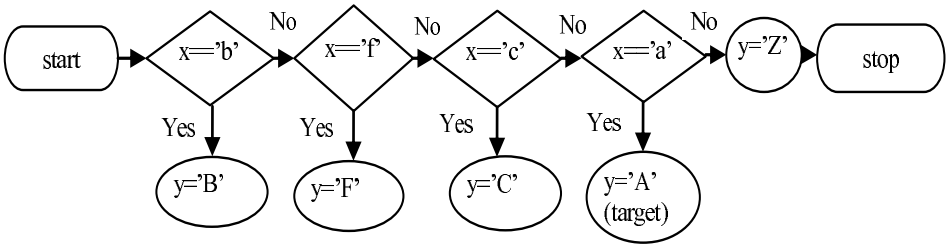


FIGURE 3.2. *CFG for a simple switch-case construct*

As shown in Fig. 3.2 each decision node is control-dependent on the previous decision nodes. For *switch-case* constructs represented as nested *if-then-else* statements, each *case* branch is dependent on the *case* branching node it leaves and all the *case* branching nodes located before it. For instance in Fig. 2.3 the branch corresponding to x=='a' is control dependent on x=='c', x=='f' and x=='b' branches. Considering that the target branch is the branch corresponding to case 'a', then the approximation level will be computed as follows:

- its value will be 0 if test data diverge away at condition node x=='a';
- its value will be 1 if test data diverge away at condition node x=='c';
- its value will be 2 if test data diverge away at condition node x=='f';
- its value will be 3 if test data diverge away at condition node x=='b'.

If the fitness function is computed for two specific values of the x variable, 'c' and 'b', the corresponding *branch distances* are 2, respectively 1 if we consider the traditional approach for computing the *branch distance* based on relational predicates [11]. So for these two values ('c' and 'b') the *approximation level* equals 1 and 3 respectively. Considering that the fitness value is the sum between the *approximation level* and the *branch distance*, the fitness value for x=='c' equals 3 and the fitness value for x=='b' equals 4.

Taking into consideration that better test data have smaller fitness values, the value 'c' is considered to be better than the value 'b' because it is closer to 'a' (which constitutes the target branch). This choice is contrary to the traditional approach, because 'b' is closer to 'a'.

In conclusion the approach with nested *if-then-else* statements is not a perfect one because in the *switch-case* constructs the order in which the clauses are written is not important for the evaluation of the fitness function, while the nested *if-then-else* statements can induce significantly different fitness values depending on the order in which they are written. For instance, the fitness value for x = 'c' is smaller than the fitness value for x = 'b' even though 'b' is closer to 'a' than 'c' is. This approach is not guiding the evolutionary search algorithm in the correct direction, because the dependencies between *case* branches result in an inappropriate approximation level value.

3.2. ACBs-based fitness function calculation. The ACBs-based approach for fitness calculation assumes that all *case* branches in the *switch-case* construct are mutually exclusive in semantics [2]. A special CFG called *Flattened Control Flow Graph* (FCFG) is described in [2]. This graph is extended from the traditional

CFG, with the only difference that the switch node can have more than two successors, all of them being on the same level. In this graph each case branch is control-dependent only on the branching switch node. Fig. 3.3 shows the FCFG corresponding to the simple *switch-case* construct presented in Fig. 2.3:

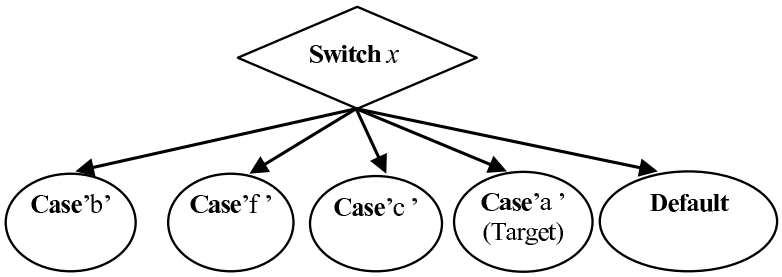


FIGURE 3.3. *Flattened Control Flow Graph for simple switch-case construct*

Based on the FCFG definition ([2]) each node has a set of control nodes on which it depends. This set constitutes the CBS. The target branch execution is definitely not triggered by the test data when the execution diverges away in any node from the CBS. When any node in the CBS is taken by the test data, then there is no chance that the target branch is covered. In the example from 3.3 the CBS attached to the target branch is composed by the following branch cases: 'b', 'f', 'c' and default. If the actual test data object executes one of the *case* statements from the CBS, it has no chance to execute the target branch.

With this concept of CBS and FCFG the approximation level metric (which is part of the fitness function expression) is calculated by subtracting 1 from the number of critical branches located between the node from which the test data diverge away and the target itself. The branch distance metric used for evaluating the test data uses the switch expression's value and the constant for the target branch.

Using this approach for the case when x equals 'b' or 'c', the *approximation level* will be 0 and the *branch distance* will be $|98 - 97| + 1 = 2$ and $|99 - 97| + 1 = 3$, respectively. The fitness calculated based on 3.1 will be 2 when $x ==$ 'b' and 3 when $x ==$ 'c'.

For this simple case it is obvious that the ACBs-based fitness value is guiding the evolutionary search in a correct direction compared to the nested *if-then-else* approach: the fitness value for x=='b' is smaller than the one for x=='c'. If the simple switch-case construct becomes a more complex one, containing case statements without break options and one level of nesting, then it can look like in Fig. 3.4:

```
Switch (x) {Case 'b': value= 'B';
                      break;
            Case 'k': value='K';
                      break;
            Case 'c': value='C';
                      break;
            Case 'a': Switch (y) {Case 'h': value = 'H';
                                             break;
                                  Case 'i':
                                  Case 'n': value = 'N'; //Target
                                             break;
                                  Default: value = 'Z';}
            Default: value='W';}
```

FIGURE 3.4. *Complex switch-case conditional construct*

The corresponding FCFG for the *switch-case* construct with one nesting level presented in 3.4 is shown in Fig. 3.5:

For the complex *switch-case* construct illustrated in Fig. 3.4, if x = 'b' and y = 'h' the *approximation level* is 1 and the *branch distance* is $|1 - 0| + 1 = 2$. The total fitness function value is 3. If x = 'a' and y = 'h', then the *approximation level* is 0 and the *branch distance* is $|7 - 13| + 1 = 7$. The total fitness is 7. So the

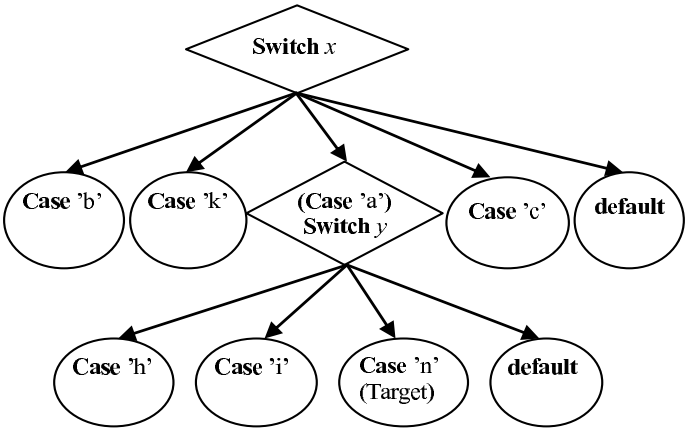


FIGURE 3.5. *FCFG for complex switch-case construct*

pair of values (x = 'b', y = 'h') has a smaller fitness value than (x = 'a', y = 'h'), even though the second pair of values is closer to the solution values (x = 'a', y = 'n'). So it is obvious that the fitness value calculation approach proposed in [15], which is based on ACB approach, misleads the evolutionary search process.

3.3. Fitness calculation based on the CMCFG approach and Korel's distance formula. To correctly guide the evolutionary search algorithm in the right direction we propose a new approach based on Compact and Minimized Control Flow Graph (CMCFG). As shown in Fig. 3.5 every switch node has as descendants several *case* branches. For the target branch, one or more *case* branches can lead to the target branch being not executed by the test data.

In the CMCFG approach each *switch* statement is represented on a different level. The *approximation level* is calculated based on the number of switch nodes from which we subtract 1. The numbering of the approximation level starts in the CMCFG in a top-down manner. As shown in Fig. 3.5, if test data diverge away from target branch at the first switch node, it will have an approximation level of 1, while if they diverge away from the target branch at the second switch node, it will have an approximation level of 0.

All the *case* branches which prevent the target branch from being executed are the *case* branches which have one of the following breaking options: *break*, *return* or *goto* statement. All these branches stop the execution of the *switch-case* structure and force the exit from this multi-way decision construct. All the *case* branches which don't have a *jump* or a *break* option are considered as not preventing the target branch to be missed and they are merged in the CMCFG graph with the next *case* branches which have a *break* option.

The CMCFG that corresponds to the complex *switch-case* construct presented in Fig. 3.4 is shown in Fig. 3.6. The node which corresponds to the *case* 'i' branch has no *break* or *return* statements and therefore it is merged with the node which corresponds to the *case* 'n' branch. In Fig. 3.6 the *case* 'n' node has resulted by merging *case* 'i' and *case* 'n' nodes. So it doesnt matter whether the test data is 'i' or 'n', because the target is prevented to be executed only when the *break* statement is encountered.

In CMCFG the *case* branches having no breaking options are not represented. Instead of these *cases*, the next *case* branch which has a *break* or a *return* statement is displayed. Compared to the approach based on critical branches, this one is more compact because it can be successfully used for modeling different type of *switch* constructs and the decision nodes which don't prevent the target to be covered are not present in the graph. The processing time of this new graph is smaller compared to the processing time for the FCFG, because the graph has fewer nodes.

The new proposed fitness function used for evaluating each test data is:

$$F(test\_data) = Approx\_level + \sum Normalized\_branch\_distance \tag{3.3}$$

The sum that appears in 3.3 refers to the sum of the normalized branch distances computed for each gene of the individuals using 3.4:

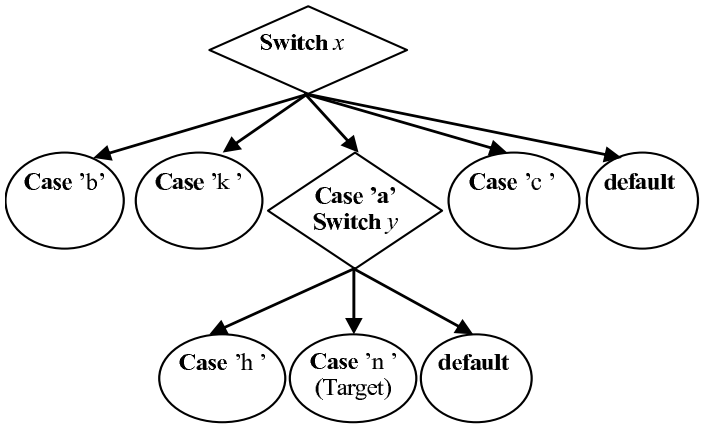


FIGURE 3.6. *CMCFG for complex switch-case construct*

$$Normalized\_branch\_distance = \frac{Branch\_distance}{Branch\_distance + 1} \tag{3.4}$$

When computing the fitness function, the normalized branch distance is chosen because the *approximation level* is more important than the *branch distance*. We use equation 3.4 for normalizing the *branch* distance according to the study presented in [3].

The *branch distance* is calculated using the functions based on relational predicates introduced by Korel in [10], for which the *switch* expression value and the target case value are used: $|switch\_expr - target\_case|$.

The test data values x = 'b' and y = 'h' will diverge away at the level of the node *case* 'b'; therefore the approximation level will be 1. The fitness function will be $(|98-97|\ /\ |98-97|+1) + (|104-110|\ /\ |104-110|+1) + 1 = 2.35$.

The second test data values x = 'a' and y = 'h' will diverge away at the level of the node case 'h'; therefore the approximation level will be 0. The fitness function will be $(|104-110|\ /\ |104-110|+1) = 0.85$.

Taking into consideration that better test data always attain smaller fitness values, by comparing the previous pairs of test data, it is easy to find out that the second one is closer to the desired test data values (x = 'a' and y = 'n'). This means that the approach based on CMCFG and *branch distance*, introduced by Korel in [10], gives a better guidance to the evolutionary search process than the evolutionary approaches based on nested *if-then-else* and ACBs.

3.4. Fitness calculation based on the CMCFG approach and Euclidian distance. In the previous section the formula proposed by Korel [10] was used for calculating the branch distance. The fitness function used for evaluating each test data was composed of the *approximation level* and the normalized *branch distance*.

Another new fitness calculation approach based on CMCFG representation of the *switch-case* construct and the Euclidian distance is proposed hereinafter. It uses for evaluating each test data the CMCFG model for representing the entire *switch-case* construct in combination with the fitness function formula presented in 3.5:

$$F(test\_data) = Approx\_level + \sum Euclidian\_distance \tag{3.5}$$

The Euclidian distance formula is the most commonly used formula for calculating distances. It examines the square root differences between coordinates of a pair of objects. Considering two points in an n-dimensional search space, the Euclidian distance formula is shown in 3.6:

$$Euclidian\_distance = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2} \tag{3.6}$$

Formula 3.6 is adapted for calculating distance in evolutionary testing by making the following changes:

1. Each x variable is the value of the node which is located along the target path in CMCFG. For instance, for the simple *switch-case* construct presented in Fig. 2.3 the x variable from the Euclidian distance formula has the value 5.

2. Each y variable form Euclidian distance formula represents the actual test data values. In case of the complex *switch-case* construct presented in Fig. 3.4 for a set of test data equal to ('b', 'h'), the y values for 3.6 will be replaced with 98 and 104 (the ASCII codes for 'b' and 'h').

For the simple *switch-case* construct presented in Fig. 2.3, in case $y1$ == 'c' the Euclidian distance will be: $\sqrt{(97-99)^2}$. The fitness value according with 3.5 for test data equal to 'c' is: 0 + 2 = 2.

For the most complex *switch-case* construct presented in Fig. 3.4, in case of pair values (y1 == 'b', y2 == 'h') the Euclidian distance will be: $\sqrt{(97-98)^2+(110-104)^2}$. The fitness function value according to 3.5 for the test values x == 'b' and y == 'h' is: 1 + 6.08 = 7.08.

The performance for the new fitness calculation approach based on approximation level and Euclidian distance will be compared with the other three approaches (nested *if-then-else*, ACBs and CMCFG with normalized branch distance) in order to be able to find out easy which one is the best for generating test data.

4. Experimental Results. The experiments using the two new estimation performance formulas in conjunction with the CMCFG model were executed on eight different *switch-case* constructs having different nested levels - from 0 to 7. All these switch-case constructs were also tested using the nested *if-then-else* approach and the Alternative Critical Branches approach.

The software program used for testing the *switch-case* constructs was written in C# and all the experiments were performed using a system equipped with an Intel I3 processor running at 2.2 GHz, and Windows 7 Operating System.

For all the four approaches, ten runs were performed for testing each *switch-case* construct and the results were compared. For testing the switch-case constructs an evolutionary framework was designed and implemented in C#. The high level architecture of the software program used for generating test data which cover the target is presented in Fig. 4.1. It has a separate component which implements each evolutionary method described in Section III and three layers which together form an application able to automatically generate test data for a particular path from the software under test. The software program is a desktop application which has a very easy to use interface.

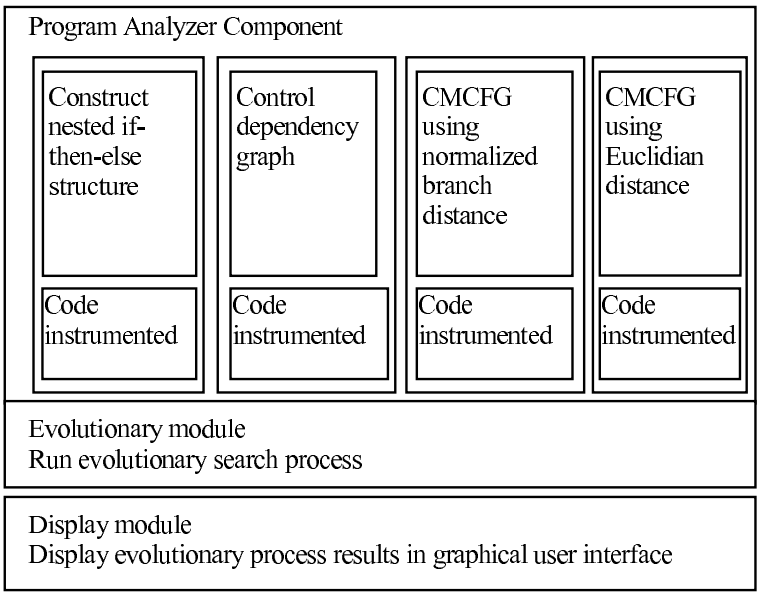


FIGURE 4.1. *High-level architecture of the evolutionary framework*

The software program used for experiments is composed of three parts: a static analyzer module, a module for running the evolutionary process and a module for displaying the graphical results.

The module that performs the static analysis consists of four sub-modules which build the nested *if-then-else* structures, or build the dependency graph flow and the CBS, or build the two CMCFG approaches (depending on which approach is to be executed). The static analyzer component instruments the code with the information needed for calculating the fitness function.

The module that executes the evolutionary process uses the data provided by the program analyzer module and runs the genetic algorithm which represents the evolutionary search method in the ET process. This module runs the evolutionary process for 100 generations and uses an initial population composed of 40 randomly generated individuals. Each individual from the population represents a different test data which are applied for testing the *switch-case* constructs.

The graphical module takes the results provided by the evolutionary module and displays them in a user interface. The best individual from each generation is displayed in a data grid. For the current generation, the software application displayed the individual genes values, the fitness function value and the computational time needed for each generation.

Table 4.1 presents the best run for each of the four evolutionary approaches out of ten runs for each. It shows that the iteration number at which the evolutionary algorithm is able to find test data which covers the target branch is smaller for the two CMCFG approaches compared to the ACB and nested *if-then-else* approaches.

TABLE 4.1
Experimental results - the iteration number at which the solution is found

Nested level	**Evolutionary process**			
	IF-THEN-ELSE (nested if-then-else structure)	ACB (Alternative Critical Branches Approach)	CMCFG+ Normalized branch distance	CMCFG+ Euclidian distance
0	27	18	7	12
1	90	56	30	36
2	98	65	40	46
3	100	79	52	59
4	>100	83	60	72
5	>100	91	68	76
6	>100	96	80	88
7	>100	100	89	95

The test data were generated for unstructured switch-case constructs having case branches with no break or return options. The processing time for the CMCFG-based methods was smaller compared to the processing time needed for the CBS-based and the nested if-then-else structures approaches.

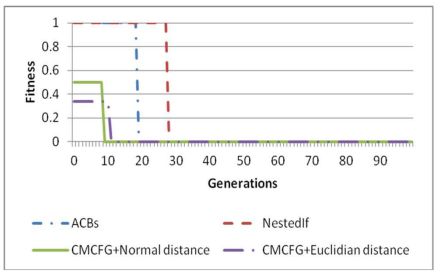
The processing time strongly depends on the number of nodes in the control flow graph. If the CMCFG has one branch node less than the normal control flow graph, then from our experiments the processing time resulted to be significantly smaller compared to the processing time for a normal control flow graph. From the experiments that were run, it came out that for each nested level our proposed methods are faster with about 1 millisecond per iteration in comparison with other two approaches.

Fig. 4.2 shows the results obtained for each nested level of the tested *switch-case* construct. All four approaches are displayed on the same graphic in order to facilitate their comparison.

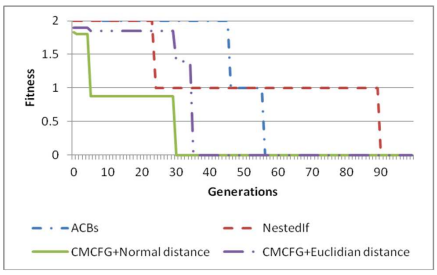
As shown in Fig. 4.2, the proposed CMCFG-based approaches which are using the normalized distance and the Euclidian distance for calculating fitness function converge faster than the two other approaches. In the previous figures one can notice that for each switch-case construct the CMCFG based approaches converge to 0 in a smaller number of generations in comparison with the other two evolutionary approaches.

The nested *if-then-else* approach is not able to generate test data for the target branch in 100 generations for a *switch-case* construct with 4 nested levels. The CBA-based approach converges much slower in comparison with our CMCFG approach that is using the two different fitness function formulas that we proposed. This means that the fitness function formulas used in CMCFG and introduced in this paper improve the guidance of the testing process based on evolutionary searches, compared to the two other approaches that were also tested. The process improvement has been illustrated in Table I, which indicates the iteration at which each approach is able to find test data for the target branch.

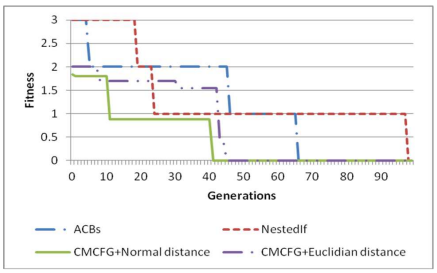
5. Conclusions and Future Work. In this paper, the approach based on nested *if-then-else* constructs and the one based on ACBs have been pointed out to be problematic because of a poor guidance of the search algorithm. The new performance estimation formulas introduced in this paper are used in conjunction with a



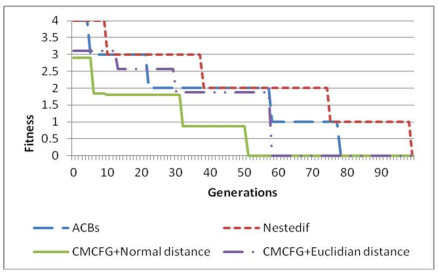
(a) Test data generation Switch-case construct with Nested level 0



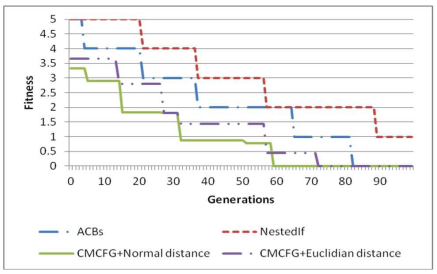
(b) Test data generation Switch-case construct with Nested level 1



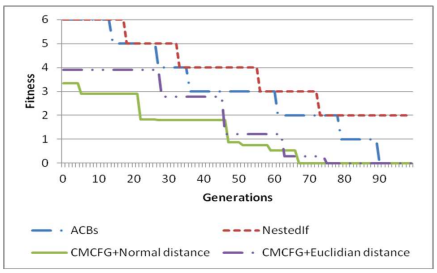
(c) Test data generation Switch-case construct with Nested level 2



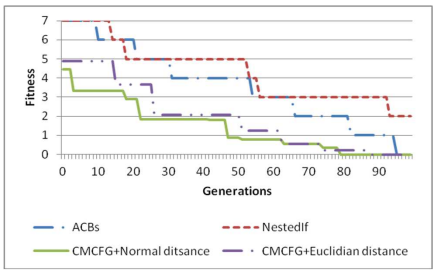
(d) Test data generation Switch-case construct with Nested level 3



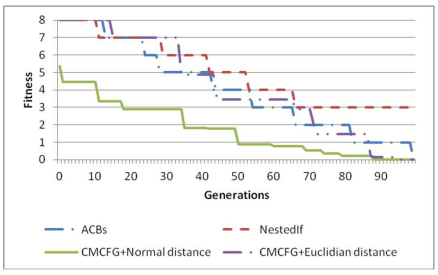
(e) Test data generation Switch-case construct with Nested level 4



(f) Test data generation Switch-case construct with Nested level 5



(g) Test data generation Switch-case construct with Nested level 6



(h) Test data generation Switch-case construct with Nested level 7

FIGURE 4.2. *Test data generation Switch-case construct with different Nested levels*

new representation model of *switch-case* constructs, which is the CMCFG.

Our proposed approaches using CMCFG representation in conjunction with normalized branch distance and Euclidian branch distance are able to find test data for a given target path in a smaller number of iterations in comparison with the other evolutionary approaches. From the practical experiments it came out that the fastest is CMCFG with normalized branch distance.

Table I illustrates that our proposed formulas for calculating fitness function are 40 iterations faster in finding test data than the *if-then-else* approach and 20 iterations faster than the ACB approach. Since the best evolutionary approach finds test data in a smallest number of iterations, Table 1 clearly shows that the best method is the CMCFG-based one with normalized *branch distance*. The two new fitness function formulas used in conjunction with the CMCFG approach are two original approach proposed, implemented and tested here.

Future work will involve using evolutionary algorithms for generating test data that cover a particular target branch in larger projects from the applicative area. In order to be able to completely automate test data generation process, a complete software framework should be implemented. This software framework should offer the human tester the possibility of choosing between different approaches and evolutionary algorithms, make suggestions concerning the best strategy to adopt for different classes of software programs etc.

REFERENCES

[1] Phil Mcminn and Mike Holcombe. *The State Problem for Evolutionary Testing.* In *Genetic and Evolutionary Computation Conference (GECCO)*, Springer-Verlag, 2003, pp. 2488-2498

[2] AnkurPachauriand Gursaran. Program test data generation branch coverage with genetic algorithm: Comparative evaluation of a maximization and minimization approach. *International Journal of Software Engineering and Applications*, 3(1):207-218, January 2012.

[3] A. Arcuri. It does matter how you normalise the branch distance in search based software testing. In *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, pages 205–214, 2010.

[4] André Baresel, David Binkley, Mark Harman, and Bogdan Korel. Evolutionary testing in the presence of loop-assigned flags: a testability transformation approach. In *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, ISSTA '04, pages 108–118, New York, NY, USA, 2004. ACM.

[5] André Baresel and Harmen Sthamer. Evolutionary testing of flag conditions. In *Proceedings of the 2003 international conference on Genetic and evolutionary computation: PartII*, GECCO'03, pages 2442–2454, Berlin, Heidelberg, 2003. Springer-Verlag.

[6] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. *ACM Trans. Program. Lang. Syst.*, 9(3):319–349, July 1987.

[7] Nirmal Kumar Gupta and Mukesh Kumar Rohil. Using genetic algorithm for unit testing of object oriented software. In *Proceedings of the 2008 First International Conference on Emerging Trends in Engineering and Technology*, ICETET '08, pages 308–313, Washington, DC, USA, 2008. IEEE Computer Society.

[8] Mark Harman, Lin Hu, Robert M. Hierons, André Baresel, and Harmen Sthamer. Improving evolutionary testing by flag removal. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 1359–1366, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[9] Mark Harman and Phil McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Trans. Software Eng.*, 36(2):226–247, 2010.

[10] B. Korel. Automated software test data generation. *IEEE Trans. Softw. Eng.*, 16(8):870–879, August 1990.

[11] P. McMinn. Search-based software testing: Past, present and future. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, pages 153–163, 2011.

[12] Phil McMinn, David Binkley, and Mark Harman. Empirical evaluation of a nesting testability transformation for evolutionary testing. *ACM Trans. Softw. Eng. Methodol.*, 18(3):11:1–11:27, June 2009.

[13] Phil McMinn and Mike Holcombe. Evolutionary testing of state-based programs. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1013–1020, New York, NY, USA, 2005. ACM.

[14] Nigel Tracey, John Clark, and Keith Mander. Automated program flaw finding using simulated annealing. In *Proceedings of the 1998 ACM SIGSOFT international symposium on Software testing and analysis*, ISSTA '98, pages 73–81, New York, NY, USA, 1998. ACM.

[15] Yan Wang, Zhiwen Bai, Miao Zhang, Wen Du, Ying Qin, and Xiyang Liu. Fitness calculation approach for the switch-case construct in evolutionary testing. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1767–1774, New York, NY, USA, 2008. ACM.

[16] Stefan Wappler, Andre Baresel, and Joachim Wegener. Improving evolutionary testing in the presence of function-assigned flags. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, TAICPART-MUTATION '07, pages 23–34, Washington, DC, USA, 2007. IEEE Computer Society.

[17] Joachim Wegener and Matthias Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Syst.*, 15(3):275–298, November 1998.

Edited by: Marc Eduard Frîncu
Received: Sep 26, 2012
Accepted: Apr 8, 2013

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2_ε using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.