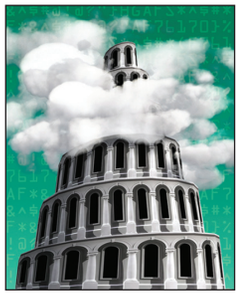
Editor: George Pallis • gpallis@cs.ucy.ac.cy



Enabling Interoperable Cloud Application Management through an Open Source Ecosystem

Nicholas Loulloudes, Chrystalla Sofokleous, Demetris Trihinas, Marios D. Dikaiakos, and George Pallis • *University of Cyprus*

Cloud computing enables on-demand provisioning of computing resources to IT solutions following a utility-based approach. In traditional public utilities, technology and standardization developments enable customers to seamlessly migrate across utility providers without being required to make changes to their home appliances. However, this is not the case with cloud computing, the so-called "fifth utility." Currently, cloud computing customers do not have at their disposal user-friendly tools and mechanisms that can preserve application deployments across different resource providers. Here, the authors present current efforts to develop an open source Cloud Application Management Framework (CAMF) based on the Eclipse Rich Client Platform. This framework facilitates cloud application lifecycle management in a vendor-neutral way.

Vendor neutrality (interoperability) has been identified as one of the main challenges in the cloud application management landscape,[1] with application developers looking for cloud application management frameworks that provide user-transparent mechanisms to handle application deployments across different providers with minimum effort.

While the five design phases of cloud application development[2] – namely functional decomposition, workload consideration, data storage, component refinement, as well as elasticity and resilience support – provide best practices for architecting scalable and interoperable cloud-based solutions, current management tools don't facilitate similar practices when it comes to the application management lifecycle.

Consider the case of an IT expert who follows these design phases to develop a three-tier media streaming service (a Web server, application server, and high-availability data store). Development begins by implementing a desirable feature set using an integrated development environment (IDE) of preference. By considering performance versus cost projections – given advertised resource specifications and pricing schemes – the IT expert registers with one of many available Infrastructure as a Service (IaaS) offerings and initiates an application-specific contextualization workflow to establish a solid runtime environment. The baseline of such workflows is the initial capacity provisioning of virtual resources available (machine images, flavor, network interfaces, persistent storage, and so on) so as to best accommodate the requirements of each component comprising the application. Subsequent processes include, but are not limited to, security policy enforcement, virtual resource instantiation, and the transfer of related artifacts (such as Java archives [JAR], Web archives [WAR], scripts, and libraries) from the local machine to

 Published by the IEEE Computer Society 1089-7801/15/$31.00 © 2015 IEEE

Cloud Application Management Frameworks

Here are examples of current cloud application management frameworks.

AWS CloudFormation (http://amzn.to/1t5rPbU): This enables provisioning of Amazon Elastic Compute Cloud (EC2) deployments by writing JSON template files. Leveraging on Amazon's Auto Scaling service allows the specification of policies for autoscaling the number of EC2 instances in a deployment.

Oracle's Virtual Assembly Builder (OVAB; http://goo.gl/Eetq0V): A GUI-based tool that simplifies the provisioning of multi-tier applications by capturing individual components into self-contained VM appliances. OVAB can instantiate appliances only on Oracle's Exalogic Elastic Cloud and supports horizontal resource scaling.

vCloud Application Director (http://goo.gl/j7LyU7): A solution offered through a Web interface for simplifying the process of designing, customizing, and deploying applications on VMware-compliant and EC2 infrastructures only.

ServiceMesh's Agility Platform (www.servicemesh.com): Enterprise-grade solution that facilitates the automatic deployment of applications on many cloud vendors, and the dynamic lifecycle management through autoscaling rules. It's a feature-rich solution, however, with closed-source connectors to the supported IaaS, and it's subscription based.

remote running instances, and configuration and launching of the Web application itself.

Although the aforementioned procedure is somewhat trivial, it entails significant effort on behalf of the IT expert, as it usually involves working across heterogeneous software (IDE, Web, shell) and operating system (Windows, Linux, and Mac) environments. Things become even more complex in the case of multiple and sometimes repetitive deployments, usually undertaken to achieve a more coherent and functional decomposition or to implement specific workload design imperatives,[2] for instance, when the application logic has been distributed among resources for improved scalability or when various functional components and their respective parameters must be refined for better workload handling. Inevitably, these actions call for a large portion of the deployment workflow to be repeated. Complexity increases even further in situations where key lifecycle operations necessitate the use of vendor-specific mechanisms. For example, the definition of metrics at various points of the application-stack for purposes of performance monitoring requires familiarization with proprietary tools and APIs. Furthermore, when applications offer elasticity and resilience capabilities, developers should get well acquainted with vendor-specific rules and strategies that orchestrate scaling actions. Despite the importance of such governing policies towards efficient workload balancing, they come with the price of a steep learning curve.

All in all, application deployment and management in IaaS Clouds can be a complex venture, typically relying on vendor-specific, proprietary software. Existing IaaS tools (see the related sidebar) don't treat interoperability as a first-class citizen and have yet to provide the right mechanisms for describing what must be conserved across deployments in different environments. Hence, the migration of applications to the cloud or even in-between existing IaaS providers requires substantial (re)contextualization effort and time, often leading to vendor lock-in. In the quest of seeking the infrastructure-deployment profile that best suits their requirements, users will benefit from application management tools that incorporate the "describe once, deploy anywhere" paradigm, thus enabling them to overcome vendor lock-in and test-drive their applications on different IaaS providers.

Enter the Cloud Application Management Framework

The Cloud Application Management Framework (CAMF; www.eclipse.org/camf) is a newly established open source technology project under the Eclipse Foundation. Originating from research activities within the European Union Seventh Framework Programme (EU FP7)-funded project "CELAR,"[3] it aims to provide extensible graphical tools over the Eclipse Rich Client Platform (RCP) that facilitate lifecycle management operations for Cloud applications, in a vendor-neutral manner. To this end, CAMF focuses on three core operations:

1. *Application description* — interoperable representations of cloud applications' structure (blueprints) with high-level depiction of service components, management operations, and interrelationships.
2. *Application deployment* — preparation and submission of descriptions to any cloud infrastructure of preference, while enabling seamless and repeatable contextualization workflows.
3. *Application monitoring* — real-time data from the underlying platform and the application itself, for fine-grained operational and performance monitoring.

The Eclipse platform provides a strong foundation for CAMF primarily because it retains a dominant position in the worldwide IDE market (http://goo.gl/mOG2u3). Its OSGi plug-in-based software architecture enables tight integration of language-specific

IDEs with a plethora of supporting suits providing code testing, analysis and profiling, code management, and so on. Through CAMF, an IT expert can empower her day-to-day development processes with cloud application management operations. Eclipse will not only function as a Web development environment, but also facilitate the preparation, migration, and monitoring of her applications to the cloud. Furthermore, build-in collaboration support can assist when participating in large development teams wherein group members can effortlessly share application descriptions, deployment information, artifacts, or even complete cloud projects with just a few clicks. Notably, all these activities can take place within a unified and intuitive graphical workspace.

Portability Qualities

To hide any inherent complexity and support cloud portability for any of the three management operations, CAMF relies on abstract models for interacting with cloud resources, as well as various open source specifications and toolkits.

IaaS Cloud Abstractions

Building on top of facilities provided by the Apache jclouds toolkit (http://jclouds.apache.org), the cloud resource model lays the groundwork towards portability by establishing a common way of interfacing with different cloud vendors. In particular, it defines a common set of abstractions for virtualized resources (server images, network, security, monitor probes, and elasticity policies and actions) and artifacts (executables, libraries, and configurations) that are key to any application deployment, irrespective of the underlying cloud environment. In addition, it defines a common set of actions on these abstractions (authentication, resource listing, monitor metric listing, configuration, and deployment) that are necessary to complete a management operation. This layer of abstraction is responsible for providing basic interfaces and classes that may be extended by IaaS-specific connectors, enabling CAMF to potentially satisfy application transition in any cloud vendor. Currently, two exemplary connector implementations are available, one for AWS EC2 and one for OpenStack-compliant cloud infrastructures, with plans to support many other IaaS.

Interoperable Application Descriptions

CAMF adopts the Organization for the Advancement of Structured Information Standards (OASIS) Topology and Orchestration Specification for Cloud Applications (TOSCA)[4] specification for blueprinting and packaging cloud applications in a standardized manner. TOSCA provides a vendor-neutral language for describing the topology of cloud applications, along with their management operations. TOSCA adopts a graph-based topology representation, which includes hardware (virtual) and software components involved and their interrelationships. Components implemented by means of *nodes* signify executing entities in a deployment with semantics such as *requirements* against the hosting environment, *capabilities*, and *policies* that govern execution aspects like resource security or elasticity. In the context of the media service scenario, a developer can choose to represent the application server component with an application node that requires from its hosting environment a virtual Ubuntu Linux image with 2 CPUs and 4 Gbytes of RAM, including an installation of the latest Apache Tomcat server. The application server node also requires the presence of a data store connection and is capable of scaling to a number of instances according to monitored performance metrics such as average CPU load and number of incoming network connections.

Automatic and Streamlined Application Configuration

TOSCA provides the necessary grammar to describe management aspects of an application by means of lifecycle operations or by more complex management plans. For each node, we explicitly can define its lifecycle operations (for example, deploy, configure, or start an instance of the corresponding node type). Currently, CAMF supports lifecycle operations and particularly node configuration through native shell scripts. In the near future, we'll incorporate new tools in the framework that allow developers to import third party, or compile their own, open source Chef cookbooks that automate and streamline application configuration processes.

Vendor-Neutral Elasticity Specification

Granted that resource autoscaling is widely regarded as one of the key ingredients of future Web applications, CAMF supports the Simple-Yet-Beautiful Language (SYBL),[5] an open source, directive-based, elasticity requirements specification. SYBL enables developers to define fine-grained scaling policies and corresponding enforcement actions, by leveraging monitoring metrics placed at different levels of the application stack. Metrics can be obtained either from native IaaS monitoring systems or from custom metric collectors (probes) provided by the user during deployment. SYBL elasticity policies are mapped to TOSCA node policies and provide two variances: *Constrains* and *Strategies*. The former express the constraints of an application related to cost, performance, and other quality metrics, and allows the underlying infrastructure to decide the appropriate scaling actions to be taken, if such intelligence is available. The latter variance lets developers take full control of elasticity management by specifying explicit scaling actions.

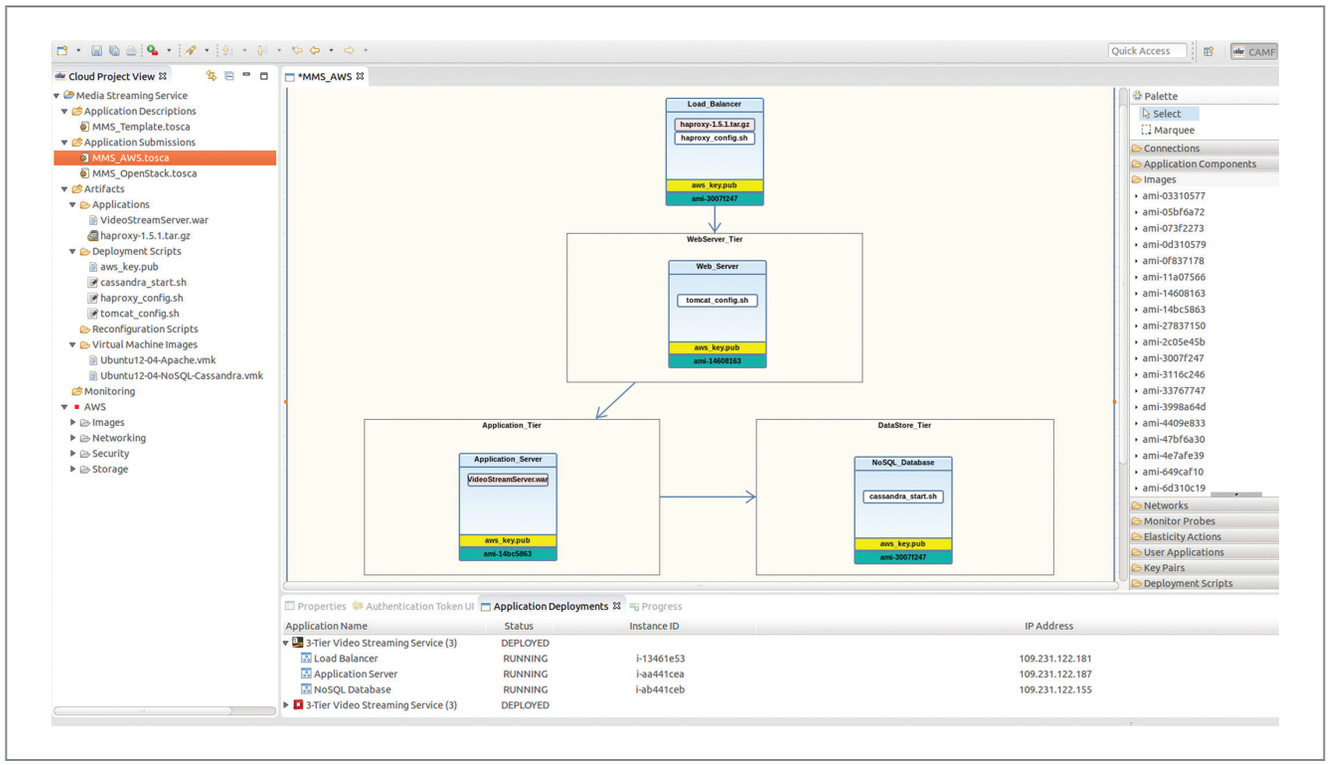


Figure 1. The Cloud Application Management Framework's (CAMF's) graphical environment and a three-tier media streaming service description in the center (a Web server, application server, and high-availability data store). On the left is a structured hierarchy that leverages the platform-independent Eclipse file system. The resource view/palette on the right acts as a front end to an internal information system (IS) and provides a visual representation of all remote resources.

Unified and Intuitive Graphical Environment

CAMF conforms to the Eclipse user interface and workspace guidelines. Thus, it organizes all files related to a cloud application in a structured hierarchy that leverages the platform-independent Eclipse file system (see Figure 1, left). The *cloud project* concept implements this hierarchy and acts as a placeholder for a particular application and its dependencies. Each project has shared access to user-added IaaS profiles that contain important information (communication endpoints, credentials, and permissions) for interfacing, querying, and utilizing any offered resources. Cloud projects provide containers for the following.

- *Application descriptions*: these hold *topology* blueprints defined using TOSCA. Each blueprint can be unique by specifying a different structure and management operations for the application at hand. However, at the same time, each blueprint is generic in that it doesn't contain any IaaS-specific information. In this basic form, application descriptions are portable templates, which upon IaaS selection can be enriched with particular metadata that materialize a component or operation(s) for the target IaaS.
- *Application deployments*: these hold IaaS-flavored blueprints along with important operational records (past and current) regarding different deployments. Among other things, these can include date/time, the target deployment IaaS, version of the application description, operational costs, and so on.
- *Artifacts*: these hold concrete software implementations required for the successful deployment and correct operation of the application. *Local* artifacts include but aren't limited to: executables and/or third party libraries, custom virtual machine images, Chef cookbooks, OS-specific configuration scripts, and so on. The cloud project structure allows for artifacts to be referenced by multiple descriptions, thus avoiding unwanted duplication.
- *Monitoring*: this holds metric collectors prepared by the developer. Currently, CAMF is fully integrated with JCatascopia,[6] an automated, multilayer interoperable monitoring system for elastic cloud environments. Besides the standard probe suite available in JCatascopia, CAMF allows

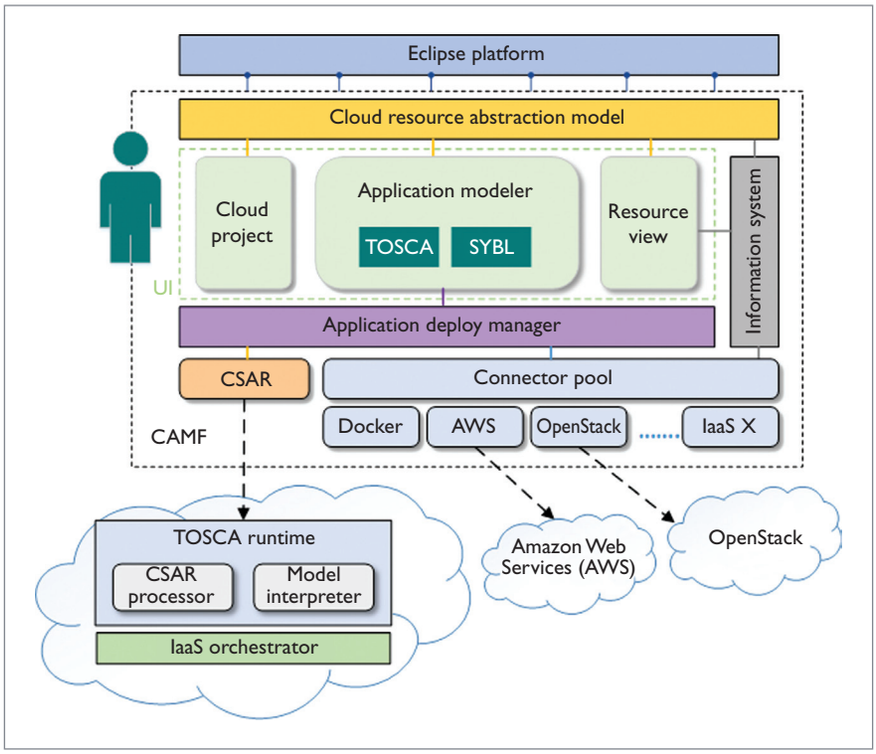


Figure 2. CAMF's extensible architecture. CSAR = Cloud Service Archive, IaaS = Infrastructure as a Service, SYBL = Simple-Yet-Beautiful Language, *and UI = user interface.*

application developers to define and implement custom metric collectors that adhere to JCatascopia's modular architecture. These probes can be placed anywhere on the application stack, ranging from the virtualization layer upwards to the application itself, to obtain meaningful metrics that report the runtime health and application performance.

The resource view/palette (see Figure 1, right) acts as a front end to an internal information system (IS) and provides a visual representation of all *remote* resources (virtual images, networks, storage, key pairs, and so on), available by one or more IaaS. Similar to the cloud project concept, the IS acts as temporary placeholder for storing resource metadata or application artifacts available on the infrastructure side. To do so, the IS relies on the IaaS-specific connectors discussed earlier to fetch from involved IaaS "marketplaces" those metadata that are essential to application deployment processes. The IS indexes resource metadata such as name, type, description, or relevant tags published by the providers. Currently, metadata indices are kept updated through periodic refresh queries; however, we aim to introduce intelligence mechanisms that regulate automatic updates in the near future. Update frequency will depend on a number of parameters, including the metadata change rate on the provider side, or which metadata the user requests more often. The palette structures information under distinct categories and allows fast lookup through standard searching and filtering mechanisms of the metadata indices.

Local and remote information culminate in the application modeling tool (see Figure 1, center), which facilitates the compilation of large and complex TOSCA-based application descriptions simply by following through a user's graphical input. This GUI-based, drag-and-drop tool eliminates the cumbersome and error-prone task of compiling large TOSCA documents manually. It generates diagrams by associating all TOSCA elements with pictorial objects (available through the palette) that are consequently used to model an application schematically. All graphical interactions with available pictorial objects (add, move, connect, rename, delete, and so on) are translated on the fly into the TOSCA XML form by considering the semantics of each underlying utilized element.

Apart from the default semantics that each palette element has, a properties view can provide additional information. Here users can specify image flavors, the minimum/maximum number of instances and elasticity policies for each component separately through a collection of dropdown and tabular input fields. Importantly, the GUI hides compilation complexities from the user and minimizes possible errors that can be introduced via non-valid graphical actions using various visual cues and textual warnings. Also, continuous checks performed on the generated TOSCA XML document assure conformance with the specification.

Easy Application Deployment and Monitoring

When it comes to deploying the application on a cloud provider, the TOSCA specification dictates that application descriptions, along with any artifacts realizing the management operations, should be packaged into a single self-contained archive, called Cloud Service Archive (CSAR). In case of a TOSCA-compliant provider, CAMF enables the submission of CSARs to a dedicated endpoint so as to be processed and interpreted accordingly by a TOSCA runtime environment, such as OpenTOSCA.[7]

According to TOSCA, cloud providers who wish to become TOSCA-compliant host a runtime entity as part of their cloud architecture (see Figure 2). This entity is responsible for communicating with the respective IaaS provisioning service for completing the necessary provider-specific operations, which will satisfy the respective TOSCA description. Nevertheless, given that not many IaaS providers are currently TOSCA-compliant, their respective connectors are used to parse the generated CSAR and make IaaS-specific API calls to satisfy any application deployment. In terms of elasticity, again, it's the responsibility of each IaaS connector to map SYBL policies to the IaaS-specific directives. The cloud abstraction layer ensures that the aforementioned processes are transparent to the end user and only the deployment result is of essence.

After deployment, the application developer can interact with the deployment view (see Figure 1, bottom) to instantly retrieve the applications' operational status without leaving CAMF's workspace. The deployment view provides a snapshot of all application deployments grouped per target IaaS. Each deployment is accompanied with provider-specific properties such as IP addresses of each component, instance IDs, uptime, and cost information, if available. A background polling mechanism refreshes the view and aims to always provide the latest information from each IaaS. Moreover, the CAMF's abstraction layer provides interfaces that enable integration with different monitoring systems, enabling its users to collect performance metrics regarding any deployed application without being required to deal with external software environments.

CAMF is still in its early stages; however, our efforts so far have shown that interoperable application management using the right components available from the open source ecosystem is indeed achievable. With the adoption, support, and valuable feedback from the open source community, CAMF can extend the possibilities of migrating applications to the cloud or across different environments.

Acknowledgments

This work was partially funded by the European Commission through the CELAR FP7 project (contract number 317790). The authors would like to acknowledge Athanasios Foudoulis and Demetris Antoniades for their contributions to CAMF, and all members of the CELAR consortium for their helpful feedback on this work.

References

1. R.J. Colville et al., *Cloud Management Platform Vendor Landscape*, Gartner, 2012; www.gartner.com/doc/2147420/cloud-management-platform-vendor-landscape.
2. C. Fehling, F. Leymann, and R. Retter, "Your Coffee Shop Uses Cloud Computing," *IEEE Internet Computing*, vol. 18, no. 5, 2014, pp. 52–59.
3. C. Sofokleous et al., "c-Eclipse: An Open-Source Management Framework for Cloud Applications," LNCS 8632, *Proc. 20th Int'l Conf. Parallel Processing*, Spring, 2014, pp. 38–49.
4. OASIS, *OASIS: TOSCA Specification Version 1.0*, 2013; www.oasisopen.org/committees/tc_home.php?wg_abbrev=tosca.
5. G. Copil et al., "SYBL: An Extensible Language for Controlling Elasticity in Cloud Applications," *Proc. 13th IEEE/ACM Int'l Symp. Cluster, Cloud, and Grid Computing*, 2013, pp. 112–119.
6. D. Trihinas, G. Pallis, and M.D. Dikaiakos, "JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud," *Proc. 14th IEEE/ACM Int'l Symp. Cluster, Cloud, and Grid Computing*, 2014, pp. 226–235.
7. T. Binz et al., "*Open-TOSCA—A Runtime for TOSCA-Based Cloud Applications*," LNCS 8274, Springer, 2013, pp. 692–695.

Nicholas Loulloudes is a computer science doctoral candidate at the University of Cyprus, Nicosia. His research interests include elastic cloud computing, vehicular computing and complex networks. Loulloudes is an active member in the open source Eclipse ecosystem and currently the project lead of the Cloud Application Management Framework (CAMF) project. Contact him at loulloudes.n@cs.ucy.ac.cy.

Chrystalla Sofokleous works at the University of Cyprus, Nicosia, as a researcher in the FP7 EU project CELAR. Her research interests focus on the Web, cloud computing, and elastic resource provisioning. She has an MSc in Web Technology from the University of Southampton. Contact her at stalosof@cs.ucy.ac.cy.

Demetris Trihinas is a computer science doctoral candidate and researcher at the University of Cyprus, Nicosia. He's interested in cloud computing, distributed systems monitoring, and elasticity behavior analysis. Contact him at trihinas@cs.ucy.ac.cy.

Marios D. Dikaiakos is a professor of computer science at the University of Cyprus and director of the University's Centre for Entrepreneurship. Dikaiakos has a PhD in computer science from Princeton University. His research interests focus on large-scale distributed computing systems. Contact him at: mdd@cs.ucy.ac.cy.

George Pallis is an assistant professor of computer science at the University of Cyprus, Nicosia. His research interests include distributed systems, such as the Web, clouds, and content distribution networks. Pallis has a PhD in computer science from the Artistotle University of Thessaloniki. Contact him at gpallis@cs.ucy.ac.cy.