Improving Security in Cloud by Formal Modeling of IaaS Resources

Flora Amato[a], Francesco Moscato[b], Vincenzo Moscato[a], Francesco Colace[c]

[a] *DIETI, University of Naples Federico II, Italy*
[b] *DiSciPol, University of Campania Luigi Vanvitelli, Italy*
[c] *DIIN, Universitá degli Studi di Salerno, Italy*

Abstract

Nowadays, it is a matter of fact that Cloud is a "must" for all complex services requiring great amount of resources. Big-Data Services are a striking example: they actually perform many kind of analysis (like analytics) on very *big* repositories. Many File Systems and middlewares exist for efficient distribution and management of data and they usually use Cloud Resources. Anyway Several problems arose about Security of data: Virtualization is the base of Cloud resources and, even if we consider data storage as *virtually* separated elements, security issues exist if privilege escalation allows for gaining control on any data on *physical* hosts. In this paper we show how it is possible to cope Model Driven Engineering techniques to security analysis and monitoring of Cloud infrastructures. For reducing overhead, we provide a formal profile of hosts thermal behaviors. Depending on services input workloads, we detect and forecast malicious actions by comparisons with real thermal data.

Keywords: Cloud services, Verification, Big-Data, Security

1. Introduction

In the age of Big-Data, Cloud Computing is the best solution to the problem of acquiring enough computational, network and storage resources in order to

Email addresses: flora.amato@unina.it (Flora Amato), francesco.moscato@unina2.it (Francesco Moscato), vmoscato@unina.it (Vincenzo Moscato), francesco.colace@unisa.it (Francesco Colace)

manage and analyze Big-Data Information. Cloud architectures allow for improvement of performances and costs trade-off, as well as for availability and reliability enhancements[1, 2]. Anyway, it was clear that several *new* risks and treats, including security problems, are arising [3].

Some surveys[4, 5] outline new security issues in Cloud Computing. In addition to well known security problems derived from services execution and protection, as well as from data storage management, the introduction of many layers of services (including all elements in IaaS, PaaS and SaaS stacks) led to the problem of propagation of malicious faults from a layer to all the other. Because of the heterogeneous and distributed nature of middlewares and interfaces (they can eventually belong to different vendors and standards), it is really difficult to design monitors for the detection of security flaws.

In particular, Cloud Virtualization is creating new security problems. It was usually addressed as a mean to implement separation from physical and virtual resources, but through virtual resources it is possible to exploit malicious escalation to gain control over hosting resources[6]. This obviously overcomes the false belief about isolation of virtual resources[1]. Privilege escalation from guest to hosting systems may result in the access to resources (and in particular to storage) belonging to other virtual hosts, producing severe security problems.

Someone can argue [7] that it is possible to use cryptography in order to protect data on virtual resources, but sometimes, the trust in separation by virtualization and the need of fast (but cheap) resources lead to insecure management of data. As example, we can mention a recent Dropbox scandal[2] that was discovered five years after the exploitation of a security breach.

Virtual-aware security is required when dealing with Cloud infrastructures: since the spread of Crisis/MORCUT malware[8], that was able to infect virtual resources through the underlying infrastructure, it is clear that direct monitoring of virtualized resource is a real problem.

[1] a nice examples can be found on Bugtraq forum: `http://www.securityfocus.com`

[2] `http://www.bbc.com/news/technology-37232635`

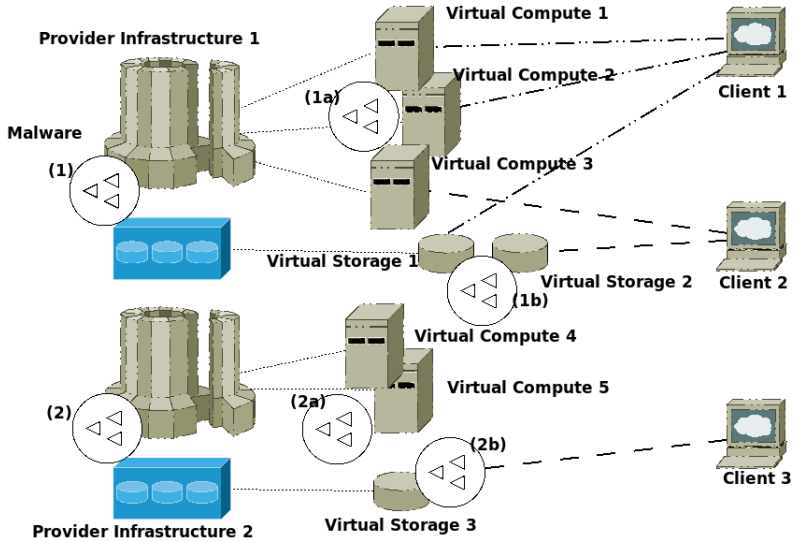


Figure 1: Example Scenario

Let us imagine the scenario depicted in Fig.1. A provider of Cloud Services has two different physical infrastructures, where it dispatches several virtual resource (Virtual Compute 1 to 5 and Virtual Storage 1 to 3) for different clients (in figure dotted lines link clients to their own Cloud resources). If a Malware like Crisis/MORCUT infects one of the physical infrastructure (1) can easily infect the other one (2) and than it can propagate to virtual resources (1a, 1b,, 2a, 2b) and to Clients. Even if a the malware can be detected on virtual resources thanks to active monitoring, some problems arise.

First of all, active monitoring of physical resources can be performed by providers or by clients: in the first case, privacy issues have to be considered and proper resources have to be used for monitoring; in the second case, clients should install and execute monitoring programs, loosing computing power of virtual resources.

In any case, physical infrastructure must run during monitoring. Anyway, the phenomenon of *sprawl* usually leads to the lack of oversight of virtual resources. Sprawl is an uncontrolled proliferation of virtual resources that are often *forgotten* by clients. This happens often when clients buy extra-resources

for some specific purpose then loosing track of them. Traditional agent-based monitoring and antivirus do not match with this scenario because of the dynamic behaviors of virtual resources. security is a poor choice for virtual environments because of the dynamic nature of virtual machines. If a guest machine is not online, traditional approaches is not enough. In addition, if Virtual Compute 4 and 5 are frozen and than reactivated even months later, clients will deal with not monitored and not patched resources and malware can further propagate to new clients and resources.

Hence, the problem is: Cloud Virtualization introduces some new security problems[9, 10]; in addition, especially in federated and multi-cloud environment, monitors are complex and expensive to implement and it is difficult to detect flaws in a whole cloud infrastructure.

In this context, a way to model and detect malicious behavior of resources is appealing, in particular for storage management. Anyway, in order to increase detection probability (even increasing false positives), it is useful to address indirect monitoring of resources.

In this work, we want to relate power management and power consumption of Cloud infrastructures in order to define models as *oracles* of attacks.

Security perspective of power management is a novel research topic: several cyber-attacks can be related to unusual power consumption or thermal behaviors of attacked systems. For example, in [11, 12, 13], different types of Denial of Services (Dos) attacks are related to energy consumption.

In this scenario, where complexity of Software and Hardware systems is growing more and more day by day, a way to apply design and monitoring methodologies also to consider *green* requirements[14, 15] is appealing.

Model-Driven Engineering (MDE) methods and techniques try to solve this problem by facilitating design, implementation and verification of complex systems.

Usually in MDE methodologies, system design refers to models (high-level description of systems) defined in formal languages; formal rules transform high level descriptions to low level ones preserving models soundness from early stages

and during all system life cycle. This leads to implementation of systems *correct by construction* where requirements are validated during all life cycle. In MDE, functional and non functional properties have to be verified at early design phase, on the system high-level models[16, 17] in order to produce correct implementations through model transformation. Different techniques used in MDE depends on the languages used for describing high-level models, and on the type of model transformations used to validate models and to implement systems components.

In this paper we use of MetaMORP(h)OSY, (Meta-modeling of Mas Object-based with Real-time specification in Project Of complex SYstems) framework for modeling thermal requirement of a (simple) Cloud infrastructure. Formal models are used both to verify properties at design time and to generate and execute monitor at run-time in order to identify unusual working conditions. The Modeling profile of MetaMORP(h)OSY is extended here in order to consider thermal requirements and behaviors. It is based on formal modeling and analysis of Multi-Agent Systems (MAS)[18].

Several methodologies have been proposed for MAS design and development [19, 20] and we use this approach here in order to face the complexity of Cloud Infrastructures.

Verification at every life-cycle step is performed by implementing transformation algorithms, which translate design, simulation and run-time description into formal models [21, 22, 23].

In this work we describe MetaMORP(h)OSY framework and methodology in Section 2; Section3 contains the definition of the model profile used to generate oracle models. Section 4 discusses some experimental results. A comparison with other similar works is reported in section 5 and, finally, Section 6 reports concluding remarks.

2. Modeling Profile

In this work, we define a profile for modeling and monitoring thermal behavior of Cloud Infrastructures in order to implement *Oracles* able to forecast malicious actions on Cloud resources. We use MetaMORP(h)OSy[24] to define the modeling profile and to implement proper model transformations[25] able to face with the forecast problem we mentioned before.

MetaMORP(h)OSY is based on a profile (i.e. a meta-model) we use to define models based on Multi-Agent Systems (MAS). It describes MAS by using an extension of Beliefs, Desires, Intentions (BDI) logics[26]. From a Designer perspective, the Framework interface allows for definition of diagrams that extend Unified Modeling Language (UML) meta-model.

Fig.2 shows the main components of MetaMORP(h)OSY. The graphical interface of MetaMORP(h)OSy is based on Papyrus[3]. Hence, designers are able to produce UML-like diagrams with stereotypes defined by MetaMORP(h)OSY profile.

The modeling profile allows both for description of systems and for definition of their requirements (in terms of QoS properties). A deeper description of Requirements profile is in [27, 28].

The framework consists of an editor (to draw design models and to specify requirements); of a set of *Observers* used to evaluate properties (and hence to verify requirements) on MetaMORP(h)OSY models, and of a set of *Translators* that enacts model transformation on models.

Observers evaluate properties both on (abstract) models and (real) running systems.

Translators implement both vertical and horizontal transformation [25]. Horizontal transformations preserve the same level of abstraction and they generate analyzable models. Vertical transformations produce models at different level of abstraction. *Abstraction* is enacted when vertical translation generates a more

[3] `https://eclipse.org/papyrus/`

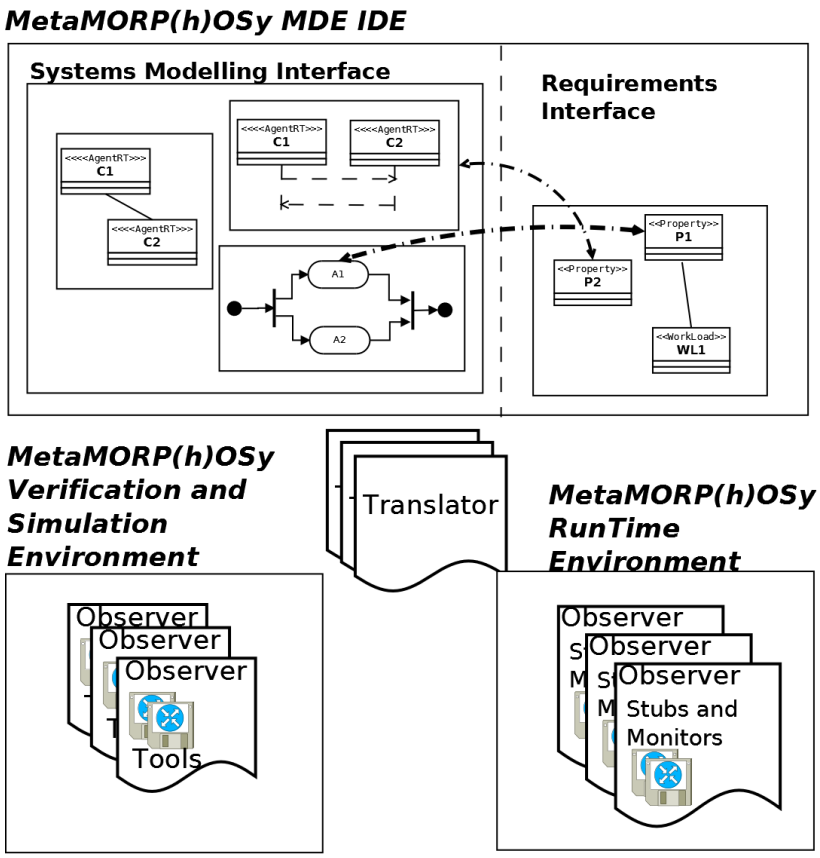


Figure 2: MetaMORP(h)OSy framework

abstract model from another one; translations producing finer grained models are called *Refinements*. Usually refinements are not fully automated and only stubs can be created. Requirements tracing in vertical transformations requires the creation of proper *Observers* able to verify abstract requirements on finer grained models. On the other hand, *Abstraction* can be used to exploit at run-time models from design phase as will be shown in Section4.

UML is the language used for definition of design models. MAS systems can be described by extending UML language. UML extension is realized by definition of proper meta-models called *modeling profiles*.

Requirements are defined by providing a MARTE-compliant[29] profile for specification of properties to analyze.

From design models, *Translators* produce analyzable formal models, or models at different abstraction level (i.e. simulation or running components). They maintain correspondences among translated models, original design models and requirements (i.e. they *trace* requirements), in order to use the proper Observer for verification at different stages.

In particular, the system enacts verification when a requirement is linked to a component of the system model. Proper modules called Observers (whose structure is defined in the modeling profile too), choose the best suitable *Translator* in the framework to execute proper Model Transformations[25] and to verify requirements. Since MetaMORP(h)OSY covers all system life cycle, it uses particular Observers at run-time for monitoring and testing purposes[30].

3. Modeling and Evaluation

In this section we describe the application of MetaMORP(h)OSY methodology to the detection of malicious actions on a Cloud infrastructure by analyzing thermal behavior of physical hosts. The problem here is the design and the analysis of Cloud components from the thermal perspective. We have to consider here that virtualization introduces new processes and workloads that physical hosts must schedule, manage and execute.

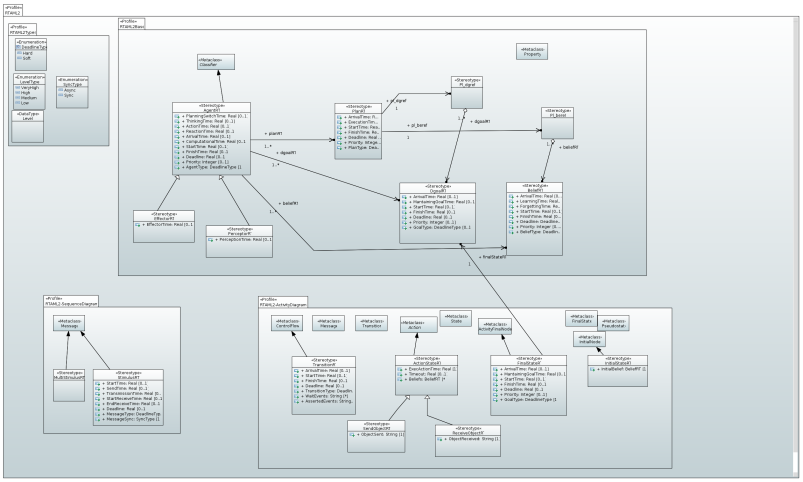


Figure 3: MetaMORP(h)OSY Agent Profile

Actors on each hosts consist in both system processes and virtualized processes: the MetMORP(h)OSy profile must manage both. In addition, we have to define profiles for properties to measures, their metrics, and we have to define an Observer Profile too in order to enact Model Translation as introduced in the previous section.

Following a MDE approach, the design model drives the definition of a runtime monitor of the thermal requirements specified at design time. Section 4 reports some experimental results. Notice that the model we present here is simple to analyze and expressive enough to obtain useful results.

In order to define a model of a Service Provider with MetaMORP(h)OSY, it must include proper stereotypes in its modeling profile. Stereotypes are defined by using UML Meta-Meta modeling language and in MetaMORP(h)OSY they appear in UML graphic diagrams.

Figure 3 shows part of the modeling profile. In particular, it reports stereotypes for definition of Agents and their properties. They include all the basic stereotypes for definition of BDI agents: agent (*AgentRT*), belief (*BeliefRT*), desires (*DGoalRT*) and intentions in terms of plans to reach goals (*PlanRT*). Here, stereotype definitions include all properties for definition of real-time behaviors

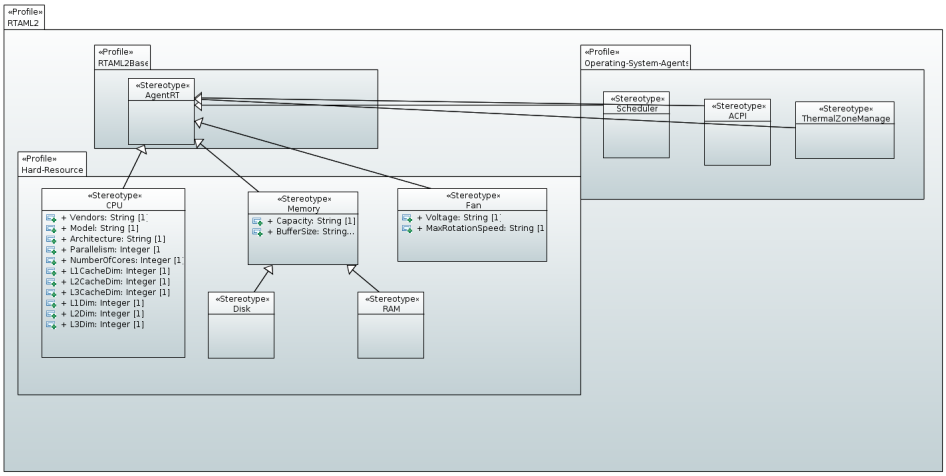


Figure 4: MetaMORP(h)OSY Physical Components Profile

and properties.

Fig.4 contains some of the stereotypes used for definition of specialized agents of the "host node" domain: hardware and software components related to thermal and functional behavior of web services and providers. CPU, Memory, Fan, ACPI, Operating system Scheduler, Service provider and Service are some of the agents defined in this part of the profile.

Fig.5 draws some properties and requirements we need to monitor temperatures, and some metrics (single absolute value in a given scale, value range etc.) used to specify their values. Properties can be used to specify one or more characteristics of a given component of a system model, or they can be defined as Requirement to analyze (or to monitor at run-time). We distinguish requirements from properties when we connect an Observer to a property for analysis.

The bottom-right part of the figure shows some Observer stereotypes. Design-time and Run-time are the two main categories of Observer in MetaMORP(h)OSY. The first kind enacts horizontal transformation in order to create proper analyzable models; the latter enacts vertical transformation in order to create run-time monitors and to use design (analyzable) models to compare monitored behaviors

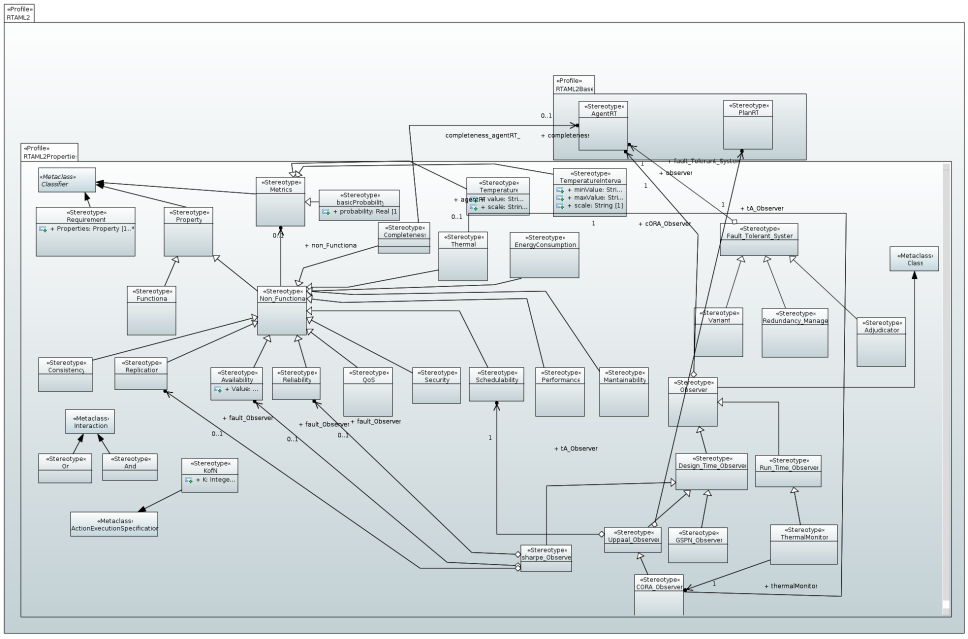


Figure 5: MetaMORP(h)OSY Properties Profile

with the expected ones.

Fig.6 contains some of the profile elements for definition of IaaS elements: Compute, Network and Storage elements; services API; elements schedulers and Operating Systems Image manager etc. The modeling profile defines several properties for each elements (like number and type of virtual processors and memory on Compute nodes, Network throughput and dimension of storage elements) but we do not report them in figure for the sake of readability.

Finally, Fig.7 contains connections among virtual and physical elements at IaaS.

System Models definition exploits MetaMORP(h)OSY modeling profiles in order to define design models.

In this example, we want to model an IaaS Cloud provider with some resources running on it. The IaaS provider consists in a Hardware configuration (with a number of physical CPU, RAM, disks and fans) with an operating system (with some processes installed on it), and with a hypervisor used to create

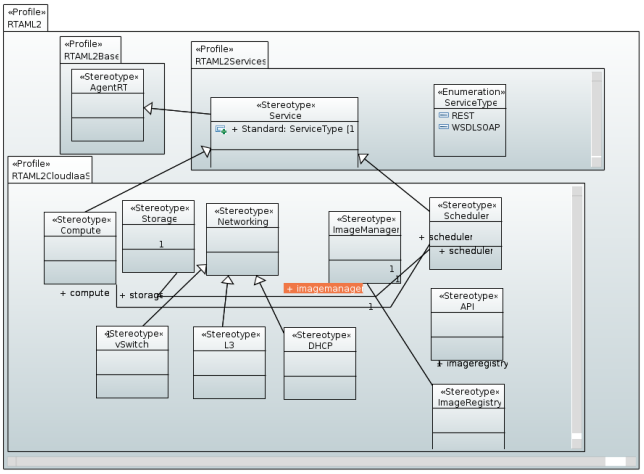


Figure 6: MetaMORP(h)OSY IaaS Profile

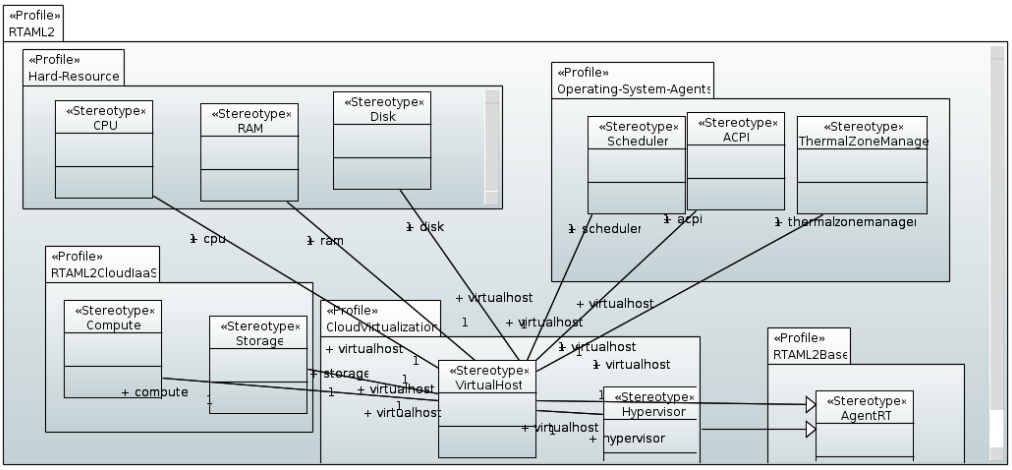


Figure 7: MetaMORP(h)OSY Profile for Iaas and Host Connections

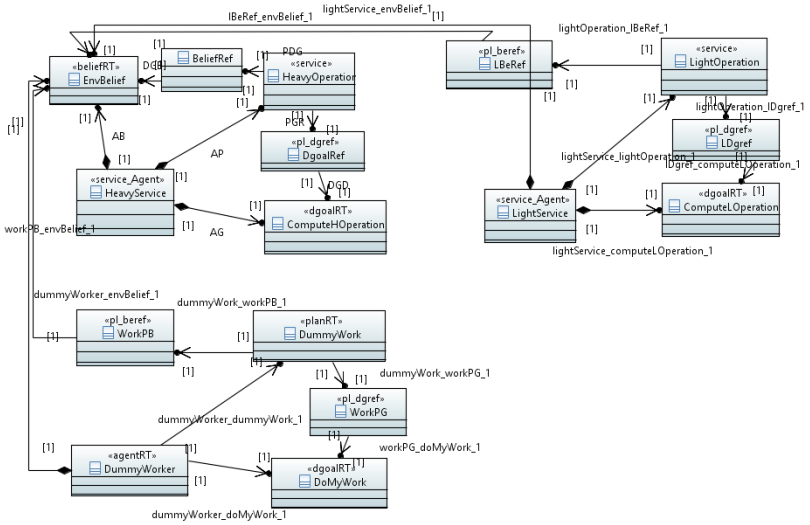


Figure 8: Agent Diagram

virtual machines.

The only perspective we analyze here is the thermal behavior of the systems when system load varies. For brevity, we do not report here the full models. Anyway, the design model is composed by four kinds of diagrams: the Agent Diagrams describe the structure of system components; the Activity diagrams (one for each plan in the Agent diagrams) describe agent behaviors while enacting their plans to accomplish their goal; the Sequence Diagrams describe Agents collaboration in reaching common goal; finally, the Observer Diagram specifies requirements to analyze and monitor under given workloads.

Fig.8 shows the agent diagram of the services implemented on each *Virtual Compute Node*. In the following example, two kind of different services are used: a *Heavy Service* and a *Light Service*. The execution time of the first service (when considering no other loads) is 5 seconds; the service time of the second is 2 seconds. Each service type has one goal (ComputeHOperation and ComputeLOperation respectively) and one plan (HeavyOperation and LightOperation). The other components in the agent diagrams are used to properly

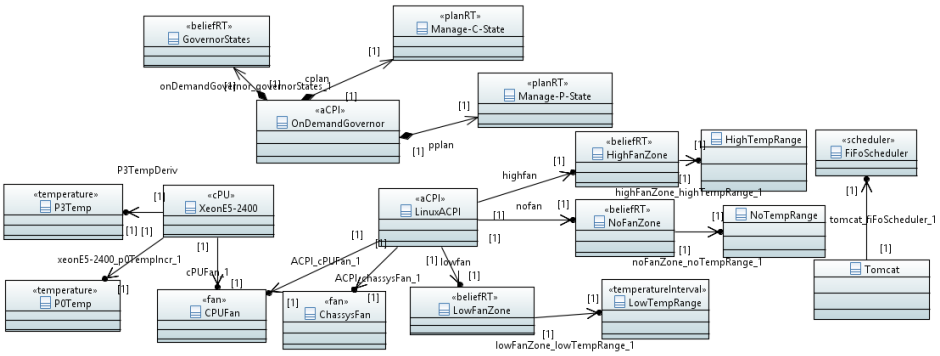


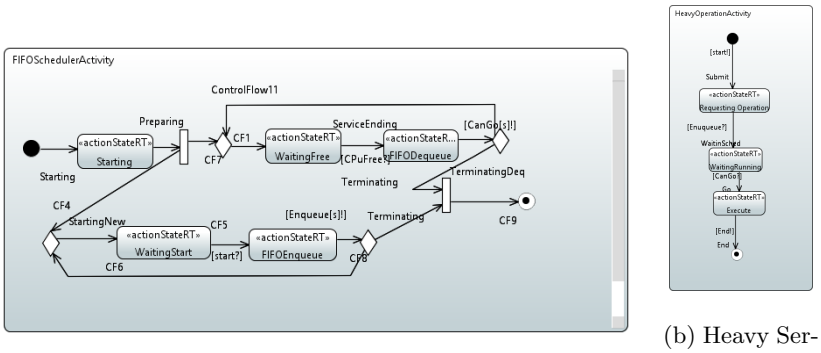
Figure 9: Provider Diagram

connect Agents, Plans and Goals and are not discussed here for brevity's sake. Another Agent (DummyAgent) is here modeled in order to represent the average workload of the system when no services are invoked.

The hardware and the operating system components of the IaaS physical infrastructure are described in the Agent Diagram in Fig.9: the ACPI with thermal zones for Fan activation, Scaling Governor[4] with their states, the Tomcat Application Server and the Scheduler used to provide requested services are defined in this diagram. All information contained here will be used to evaluate thermal behavior by using the CoraObserver depicted in Fig.5. In particular P3Temp and P0Temp are two temperature properties related to the CPU: they represent respectively the increment of the temperature when CPU works in performance and eco states.

Fig.10 reports some activity diagrams. In particular, part of the behavior of the IaaS node scheduler is reported on the left: the scheduler waits the *start* event from another agent before putting the request in its queue. A new request service is scheduled when service manager threads in Tomcat application server are free (i.e. when running services terminates). For what this example concerns, Services behaviors consist in waiting for application server scheduling,

[4] `https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt`



(a) Fifo Scheduler Activity

(b) Heavy Service

Figure 10: Activity Diagrams

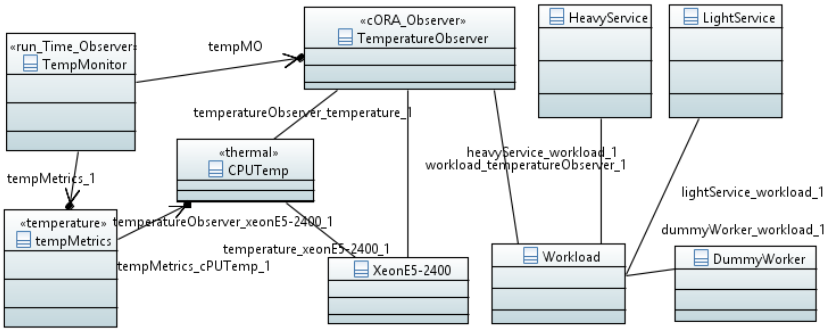


Figure 11: Observer Diagram

executing their job and finishing.

We must define other diagrams to declare on which Physical resource (nodes) we allocate virtual IaaS components[31, 32] (i.e. compute and storage nodes) but we omit their description here for brevity. Workload profile is omitted too: it is simple: a workload stereotype is linked to each *Service* agent (i.e.: normal services on physical machines, virtual hosts scheduling and management and virtual services on virtual hosts).

Workloads specify the mean number of requests and their frequencies. Timing behavior is inherited directly from *Agent* Stereotype.

At the end of structural and behavioral descriptions of the system, Users

must prepare an Observer Diagram to declare the type of analyses and of monitoring activities they want to perform on the system. In Fig.11 two Observers are defined: the first one (TemperatureObserver) is a design Observer. It translates the design model into Timed and Priced Automata in order to analyze thermal behavior of the system under a given Workload, expressed in terms of Services executing on the application Service. Workload description uses a proper language in MetaMORP(h)OSY but its definition is out of the scope of this work. The second is a Run Time Observer and it is generated by using vertical transformation. In addition, analyzable models from TemperatureObserver are used at run time to compare real behaviors with the predicted ones. Notice that there is no connection between run-time observer and Workload: in fact, the real workload is detected at run time by TempMonitor and, the prediction of Temperature behavior requires the setting of new parameters in the Observer Diagram for Workload Element and the use of the same TemperatureObserver to generate new analyzable model at run-time.

Horizontal Transformation enacted by CORA Observer is reported in Fig.12. For example, the TA_Sc model for the FIFO scheduler is reported in Fig.13 while the Linear Priced Automaton $LTPA\_Governor$ is reported in Fig.14.

In particular, Only two thermal zones are considered in the LPTA model of the governor, because only two different Fan Zones are defined in Fig.9. Hence, the top of Fig.14 models the states of the system where new services are submitted and the system works with CPU fan turned off (Fan0), then with at low speed (Fan1), and finally with at full speed (Fan2 and MaxTemp). Temperature Increments (and decrements) here depends on CPU and CPUFan properties (defined in the provider Diagram). The bottom of Fig.14 reports the behavior of the system (cooling down) when no load is submitted.

4. Experimental Results

This section reports some experimental results on an experimental testbed. It was composed by a Blade server HP CS 250-HC Store Virtual, with 4 blades,

for all Virtual Resources **do**

generate a Timed Automaton $TA\_Pl_V[plan]$ *for all Plan of resources related to virtual agents*

generate a Timed Automaton TA_Sc_V *for (Virtual) Scheduling behavior;*

for all *seq* ∈ Sequence Diagrams **do**

generate a Timed Automaton $TA\_Sq_V[seq]$ *for other TAs interactions;*

end for

Build the Product Automaton of $TA\_Pl_V[]$, TA_Sc_V, $TA\_Sq_V[]$;

Build a LPTA ($LTPA\_Workload_V$*) for Workload;*

Solve the Product TA to retrieve results on timed behavior of services;

Tune $LTPA\_Workload_V$ *with last results;*

end for

Generate Timed Automaton TA_Sc *for (Physical) Scheduling behavior;*

for all *seq* ∈ Sequence Diagrams **do**

generate a Timed Automaton $TA\_Sq[seq]$ *for other TAs interactions;*

end for

Build the Product Automaton of $TA\_Pl[]$, TA_Sc, $TA\_Sq[]$;

Build a LPTA ($LTPA\_Governor$*) for Governor and Thermal Behavior;*

Build a LPTA ($LTPA\_Workload$*) for Workload;*

Solve the Product TA to retrieve results on timed behavior of services;

Tune $LTPA\_Workload$ *with last results;*

Build the Product Automaton of $LTPA\_Governor$ *and* $LTPA\_Workload$, $TA\_Pl[]$, TA_Sc, $TA\_Sq[]$, $TA\_Pl_V[]$, TA_Sc_V, $TA\_Sq_V[]$;

solve and predict system behavior by optimal analysis and simulation of LPTA product Automaton;

Figure 12: CoraUPPAAL Observer Transformation

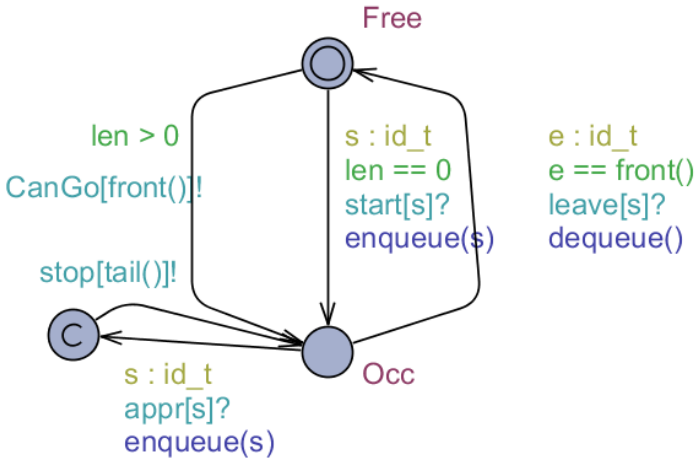


Figure 13: FIFO Scheduler TA

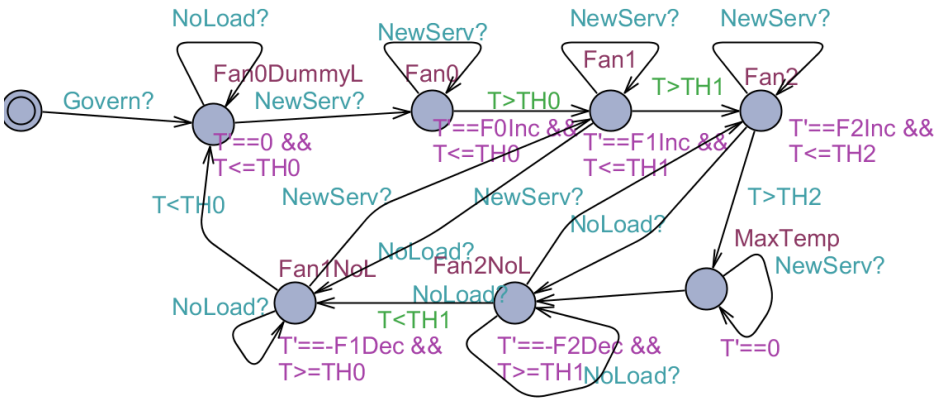


Figure 14: Governor Temperature LPTA

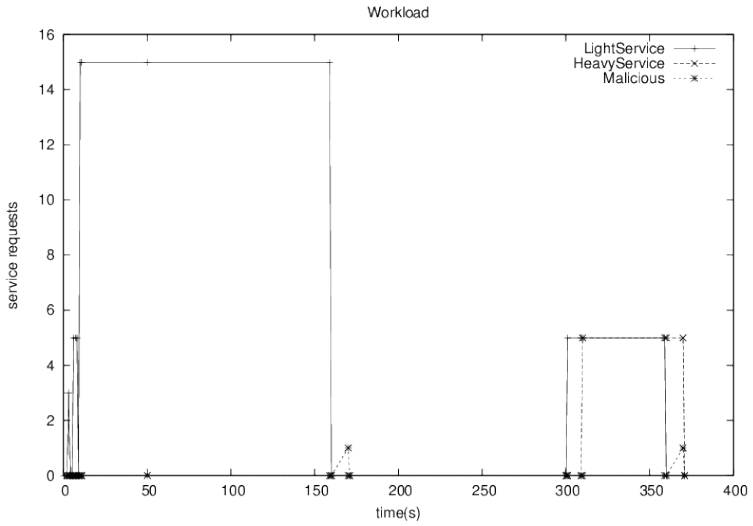


Figure 15: Input Workload

each equipped with 2 CPUs Xeon-E5-2680 CPU with 64 GBytes of RAM, a RAID 5 disk with 3 TBytes of available storage. On the System it was installed VMWare vSphere on a Windows 7 node embedded in the Blade for the creation of virtual machines of physical nodes. We implemented there an OpenStack Cloud Infrastructure. Compute nodes use *nova* default image and we created *cinder* volumes of 10 Gbytes.

The malicious code we injected on virtual machine services was devoted to implement a privilege escalation attack in order to retrieve contents of whole physical disks and to send it through an SSH connection to a target machine in the same network.

In order to enable the attack, we installed bugged VMware and Windows version. The malicious code can be retrieved here[5].

Fig.15 shows the workload used to obtain results reported in this section: it reports the number of services requests (for Heavy and Light services) during

[5] `https://docs.google.com/document/d/1sIYgqrytPK-CFWfqDntraA_Fwi2Ov-YBgMtl5hdrYd4/mobilebasic?pli=1`

the time. In addition, a malicious service request is considered at time 170 and 370. It uses an exploit in Light Service to execute privilege escalation and to harvest a small amount of data from the compromised storage node.

The malicious service is not executed continuously since this lead to trivial results: continuous execution of the service leads to an evident increase of temperature.

Execution time of malicious service is estimated to be 60s.

Warkload is repeated continously with a period of 500s.

The experiment reports results of two different runs: the first without malicious requests, the second running the exploit.

We tuned MetaMORP(h)OSy models by running systems without malicious interventions. This does not limit our approach: system models can be easily computed off-line or estimated during the first executions of the services.

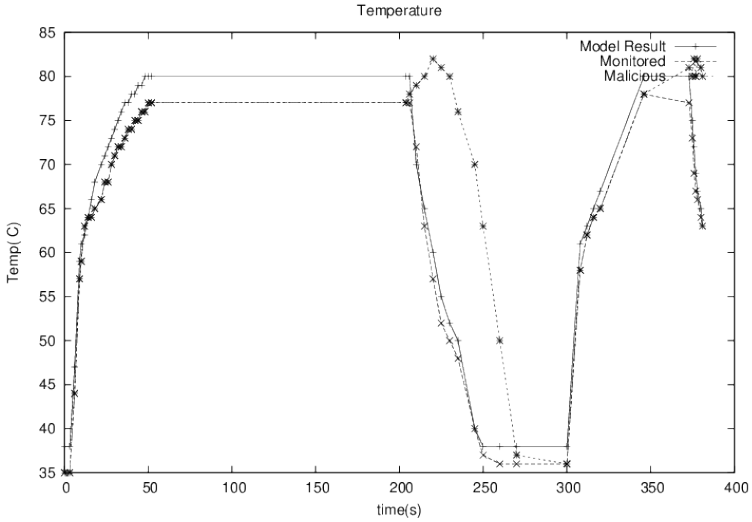


Figure 16: Evaluated and Monitored Temperatures

MetaMORP(h)OSY generates a Run-Time Observer for thermal properties evaluation which basically monitors kernel interfaces of governors and thermal zones on a remote system. The monitor uses models generated by horizontal transformation to evaluate if current thermal behavior is compliant with the

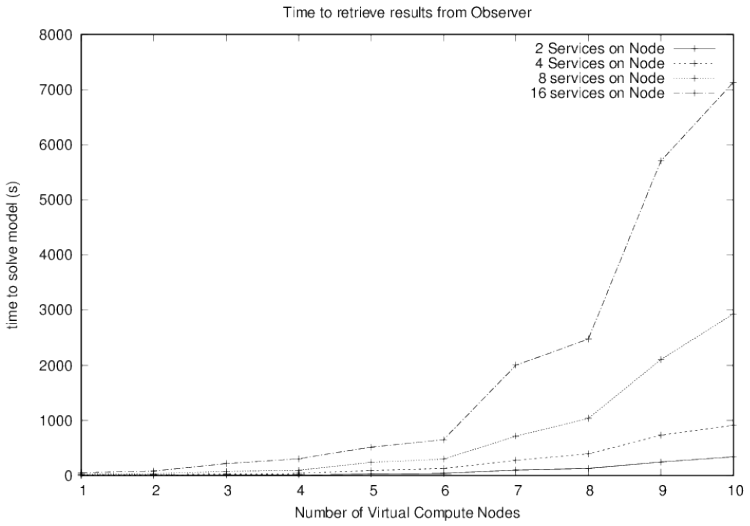


Figure 17: Time to Produce Results from Models

expected one. It evaluates new workload parameters every 2 seconds and it analyze the LPTA described in the previous section in order to check differences between expected and real thermal behaviors of the system.

Fig.16 depicts thermal behavior obtained by the analysis of design model (Model Request), the behavior monitored at run time when no malicious intervention (Monitored) is enacted and the behavior during malicious attacks (Malicious).

Notice that Monitored and Modeled behaviors are similar (they differs in few degrees), while Malicious behavior is far distant from expected behavior: this is a symptom of incorrect, unpredictable or malicious execution.

We obtain similar results when deploying different compute nodes on the same physical Blade. The introduction of more virtual nodes and services increases the complexity of the LPTA model to analyze.

Finally, Fig.17 reports the time in seconds elapsed to produce results from models, depending on the number of virtual nodes on a physical blade, and on the number of virtual services deployed on the node. Models have been analysed on an Intel i5 4400 Processor, with 4 GByte of RAM and a solid state HD of

128 GBytes, running Ubuntu Linux 16.04.

Time increases with number of nodes more than linearly because of product automata creation. Anyway, due to hardware resource saturation, it is really unusual the allocation of more than 20 virtual machines on the same node.

In addition, remember that models have to be evaluated once for all, and that they have to be re-analyzed only when services or their loads change: in a stable environment, this event is not so frequent.

5. State of The Art

Some recent works exist discussing the problem of securing virtual machines from malicious behaviors. Authors of [33] realized a system able to detect attacks by monitoring system calls on host systems realizing promising results. Anyway, we think it is difficult to distinguish massive attacks performed with *legal* interactions due to the overhead of monitors. Authors of [34] describe some interesting methods to detect malware execution in Cloud infrastructures. They manage malwares and DDoS attacks, by using dedicated monitoring components at hypervisor level. The novelty of their approach is in coping both network and system level data, but again we think they suffer for overhead when managing real Cloud infrastructures. In [35], data theft in IaaS Clouds is addressed. Authors use an expert system in order to detect data theft in Clouds. Their monitoring system analyzes patterns of network messages used to transfer data.

Anyway at the best of our knowledge, all these systems fail when dealing with real infrastructures, due to the high number of resources to monitor. In addition, due to their characteristics, they usually fail in *forecasting* malicious behaviors since they are not based on any fast and useful forecasting model.

For what our MAS methodology concerns, many agent-oriented modeling languages and methodologies have been proposed in the literature. While MDE should make easier the development of services, the effort of design and implement proper languages for models description and analysis is demanded to Domain Specific Modeling Languages (DSML)[36]. Usually UML could be con-

sidered as a standard design language, but it is not the best tool for modeling agent based systems[37]. This is why several DSML for Agent Based models have been developed like Agent-UML [38], or the one defined for the Prometheus methodology [39].it has been integrated within INGENIAS[40] and it consists of a meta-model describing elements of MAS from different perspectives like environment, goals, tasks specifying the behavior of each agent. Anyway, INGENIAS focuses only on the generation of executable software and no verification issues have been addressed especially in complex systems. A tool that uses an approach similar to MetaMORP(h)OSY is SCADE[41]. It enacts model checking to verify properties at design time and implements designed models into embedded systems. Unlike MetaMORP(h)OSY, it implements only verification of state reachability properties and it focuses on embedded and control systems. Another framework that focuses on the generation of executable embedded code in TOPCASED [42, 43, 44]. On the other hand, Model Driven Engineering techniques have not been applied yet for modeling *green* requirements[45, 2]. Few approaches have been reported in literature where usually formal models like differential equations, automata or Markovian models are used directly to model system behaviors and properties. Very frequent is the study of thermal or voltage behavior of systems by means of dynamic equations like in [46] or [47], but models are usually used to tune scheduling or optimal allocation algorithms. Something similar to the approach presented here is in [48], where a model based on priced timed automaton is used to analyze *green* properties of a system. MetaMORP(h)OSY differs from this last approach (and from the other presented in this section) because it offers a *true* MDE integrated approach to system models. The MetaMORP(h)OSY modeling methodology is oriented especially towards requirements verification. Differently from other proposed approaches, it uses both vertical and horizontal transformation. It is a more versatile framework and at the state of the art it has been used both for verification of properties and generation of running embedded systems, as well as for supporting Cloud Services design and provisioning [23, 49]. It also supports different verification techniques (and new ones can be plugged in) including

multi-formalisms and multi-solution approaches [22].

6. Conclusions and Future Works

This work describes a methodology that supports the design, verification and validation of thermal properties of a Cloud IaaS infrastructure. The methodology covers the entire life cycle of IaaS infrastructures, going beyond abstraction, and providing a mean to analyze systems at design and run-time. The use of model transformation techniques allows IaaS managers to use design models as oracles in the case of unpredicted and malicious behaviors. Future works include the extension of the methodology to consider more complex governor, thermal and energy management, as well as more IaaS middlewares.

7. References

[1] Y. Xiang, I. Stojmenovic, P. Mueller, J. Zhang, Security and reliability in big data, Concurrency and Computation: Practice and Experience 28 (3) (2016) 581–582.

[2] A. Castiglione, A. Santis, A. Castiglione, F. Palmieri, U. Fiore, An energy-aware framework for reliable and secure end-to-end ubiquitous data communications, in: Proceedings - 5th International Conference on Intelligent Networking and Collaborative Systems, INCoS 2013, 2013, pp. 157–165.

[3] K. Dahbur, B. Mohammad, A. B. Tarakji, A survey of risks, threats and vulnerabilities in cloud computing, in: Proceedings of the 2011 International conference on intelligent semantic Web-services and applications, ACM, 2011, p. 12.

[4] D. A. Fernandes, L. F. Soares, J. V. Gomes, M. M. Freire, P. R. Inácio, Security issues in cloud environments: a survey, International Journal of Information Security 13 (2) (2014) 113–170.

[5] A. Singh, K. Chatterjee, Cloud security issues and challenges: A survey, Journal of Network and Computer Applications 79 (2017) 88–115.

[6] K. Hashizume, D. G. Rosado, E. Fernández-Medina, E. B. Fernandez, An analysis of security issues for cloud computing, Journal of Internet Services and Applications 4 (1) (2013) 5.

[7] A. Castiglione, G. Cattaneo, A. De Santis, F. Petagna, U. Ferraro Petrillo, SPEECH: Secure Personal End-to-End Communication with Handheld, Vieweg, Wiesbaden, 2006, pp. 287–297.

[8] S. Mansfield-Devine, Security review: the past year, Computer Fraud and Security 2013 (1) (2013) 5 – 11.

[9] L. Jiang, H. Tian, J. Shen, S. Maharjan, Y. Zhang, Quality of protection in cloud-assisted cognitive machine-to-machine communications for industrial systems, MONET 21 (6) (2016) 1032–1042.

[10] J. Abawajy, G. Wang, L. T. Yang, B. Javadi, Trust, security and privacy in emerging distributed systems, Future Generation Comp. Syst. 55 (2016) 224–226.

[11] F. Palmieri, S. Ricciardi, U. Fiore, Evaluating network-based dos attacks under the energy consumption perspective: New security issues in the coming green ict area, in: Broadband and Wireless Computing, Communication and Applications (BWCCA), 2011 International Conference on, 2011, pp. 374–379.

[12] F. Palmieri, S. Ricciardi, U. Fiore, M. Ficco, A. Castiglione, Energy-oriented denial of service attacks: an emerging menace for large cloud infrastructures, The Journal of Supercomputing 71 (5) (2015) 1620–1641.

[13] U. Fiore, A. Castiglione, A. De Santis, F. Palmieri, Exploiting Battery-Drain Vulnerabilities in Mobile Smart Devices, IEEE Transactions on Sustainable Computing 2 (2) (2017) 90–99.

[14] Z. Zhou, M. Dong, K. Ota, G. Wang, L. T. Yang, Energy-efficient resource allocation for D2D communications underlaying cloud-ran-based LTE-A networks, IEEE Internet of Things Journal 3 (3) (2016) 428–438.

[15] D. Li, M. Zhou, P. Zeng, M. Yang, Y. Zhang, H. Yu, Green and reliable software-defined industrial networks, IEEE Communications Magazine 54 (10) (2016) 30–37.

[16] S. Wen, W. Zhou, J. Zhang, Y. Xiang, W. Zhou, W. Jia, C. C. Zou, Modeling and analysis on the propagation dynamics of modern email malware, IEEE Trans. Dependable Sec. Comput. 11 (4) (2014) 361–374.

[17] S. Wen, W. Zhou, J. Zhang, Y. Xiang, W. Zhou, W. Jia, Modeling propagation dynamics of social network worms, IEEE Trans. Parallel Distrib. Syst. 24 (8) (2013) 1633–1643.

[18] Z. Guessoum, J.-P. Briot, N. Faci, O. Marin, Towards reliable multi-agent systems: An adaptive replication mechanism, Multiagent Grid Syst. 6 (2010) 1–24.

[19] K. M. Kavi, M. Aborizka, D. Kung, N. Texas, A framework for designing, modeling and analyzing agent based software systems, in: in Proc. of 5th International Conference on Algorithms and Architectures for Parallel Processing, 2002, pp. 23–25.

[20] V. T. Da Silva, C. J. P. De Lucena, From a conceptual framework for agents and objects to a multi-agent system modeling language, Autonomous Agents and Multi-Agent Systems 9 (2004) 145–189.

[21] G. Franceschinis, M. Gribaudo, M. Iacono, S. Marrone, F. Moscato, V. Vittorini, Interfaces and binding in component based development of formal models, in: IEEE proc. of VALUETOOLS 09 conference, 2009, p. 44.

[22] F. Moscato, V. Vittorini, F. Amato, A. Mazzeo, N. Mazzocca, Solution workflows for model-based analysis of complex systems., IEEE T. Automation Science and Engineering 9 (1) (2012) 83–95.

[23] F. Moscato, B. D. Martino, R. Aversa, Enabling model driven engineering of cloud services by using mosaic ontology, Scalable Computing: Practice and Experience 13 (1).

[24] R. Aversa, B. Di Martino, F. Moscato, Critical systems verification in metamorp (h) osy, in: Computer Safety, Reliability, and Security, Springer, 2014, pp. 119–129.

[25] T. Mens, P. V. Gorp, A taxonomy of model transformation, Electronic Notes in Theoretical Computer Science 152 (0) (2006) 125 – 142, proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005), Graph and Model Transformation 2005. doi:10.1016/j.entcs.2005.10.021.
URL http://www.sciencedirect.com/science/article/pii/S1571066106001435

[26] M. Wooldridge, Agent-based software engineering, in: IEE Proceedings on Software Engineering, 1997, pp. 26–37.

[27] F. Moscato, F. Amato, A. Amato, R. Aversa, Model-driven engineering of cloud components in metamorp (h) osy, International Journal of Grid and Utility Computing 5 (2) (2014) 107–122.

[28] F. Moscato, Model driven engineering and verification of composite cloud services in metamorp(h)osy, in: Proc. of 6th, International Conference on Intelligent Networking and Collaborative Systems INCoS-2014, IEEE, 2014.

[29] M. Faugere, T. Bourbeau, R. de Simone, S. Gerard, Marte: Also an uml profile for modeling aadl applications, Engineering of Complex Computer Systems, IEEE International Conference on 0 (2007) 359–364.

[30] D. Di Domenico, F. Moscato, Automatic monitor generation for cloud services, Proceedings - 2015 9th International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2015 (2015) 547–552.

[31] L. He, L. T. Yang, Z. Du, F. Pop, Resource management in virtualized clouds, Scientific Programming 2016 (2016) 1407940:1–1407940:2.

[32] G. Nan, Z. Mao, M. Li, Y. Zhang, S. Gjessing, H. Wang, M. Guizani, Distributed resource allocation in cloud-based wireless multimedia social networks, IEEE Network 28 (4) (2014) 74–80.

[33] P. Mishra, E. S. Pilli, V. Varadharajan, U. Tupakula, Securing virtual machines from anomalies using program-behavior analysis in cloud environment, in: 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016, pp. 991–998.

[34] M. R. Watson, A. K. Marnerides, A. Mauthe, D. Hutchison, et al., Malware detection in cloud computing infrastructures, IEEE Transactions on Dependable and Secure Computing 13 (2) (2016) 192–205.

[35] J. Nikolai, Y. Wang, A system for detecting malicious insider data theft in iaas cloud environments, in: Global Communications Conference (GLOBECOM), 2016 IEEE, IEEE, 2016, pp. 1–6.

[36] Z. Hemel, L. C. Kats, D. M. Groenewegen, E. Visser, Code generation by model transformation: a case study in transformation modularity., Software and Systems Modeling 9 (2010) 375–402.

[37] B. Bauer, Uml class diagrams revisited in the context of agent-based systems, in: Agent-Oriented Software Engineering II, Springer, 2002, pp. 101–118.

[38] B. Bauer, J. P. Mller, J. Odell, Agent uml: A formalism for specifying multiagent software systems, Int. Journal of Software Engineering and Knowledge Engineering 11 (2000) 91–103.

[39] J. M. Gascuea, E. Navarro, A. Fernndez-Caballero, Model-driven engineering techniques for the development of multi-agent systems, Engineering Applications of Artificial Intelligence 25 (1) (2012) 159 – 173.

[40] A. Fernndez-Caballero, J. Gascuea, Developing multi-agent systems through integrating prometheus, ingenias and icaro-t, in: J. Filipe, A. Fred, B. Sharp (Eds.), Agents and Artificial Intelligence, Vol. 67 of Communications in Computer and Information Science, Springer Berlin Heidelberg, 2010, pp. 219–232.

[41] P. Abdulla, J. Deneux, G. Stalmarck, H. Agren, O. Akerlund, Designing safe, reliable systems using scade, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Vol. 4313 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 115–129.

[42] J. Farines, M. De Queiroz, V. da Rocha, A. Carpes, F. Vernadat, X. Cregut, A model-driven engineering approach to formal verification of plc programs, in: Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on, 2011, pp. 1–8. doi:10.1109/ETFA.2011.6058983.

[43] F. Palmieri, U. Fiore, S. Ricciardi, A. Castiglione, GRASP-based resource re-optimization for effective big data access in federated clouds, Future Generation Computer Systems 54 (2016) 168 – 179.

[44] A. Castiglione, M. Gribaudo, M. Iacono, F. Palmieri, Modeling performances of concurrent big data applications, Software: Practice and Experience 45 (8) (2015) 1127–1144.

[45] A. Castiglione, F. Palmieri, U. Fiore, A. Castiglione, A. De Santis, Modeling energy-efficient secure communications in multi-mode wireless mobile devices, Journal of Computer and System Sciences 81 (8) (2015) 1464–1478.

[46] M. Bao, A. Andrei, P. Eles, Z. Peng, Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling., in: DATE, IEEE, 2010, pp. 21–26.

[47] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.

[48] E.-Y. Kang, G. Perrouin, P. Schobbens, Towards formal energy and time aware behaviors in east-adl: An mde approach, in: Quality Software (QSIC), 2012 12th International Conference on, 2012, pp. 124–127.

[49] F. Moscato, R. Aversa, B. D. Martino, T.-F. Fortis, V. I. Munteanu, An analysis of mosaic ontology for cloud resources annotation, in: IEEE Proc. of FedCSIS 2011 Conference, 2011, pp. 973–980.