Exploiting Model Profiles in Requirements Verification of Cloud Systems

Francesco Moscato
DiSciPol,
Second University of Naples,
Italy
E-mail: francesco.moscato@gmail.com

Abstract: Cloud Systems arose in the last years as a standard de-facto in IT enterprises for offering practically any kind of services to worldwide users. They provide means for realizing and distributing everything-as-a-service, including infrastructures, hardware and software platforms and services. Even if now, Service-centric models and technologies are mature in the IT scenario, composition, analysis and validation of Cloud services are open research challenges. In this work, we describe a Modeling Profile that enables Model Driven Engineering (MDE) analysis of systems and requirements verification of Cloud-based services. The verification process exploits formal methods during the whole life cycle of services. We show the application of the proposed methodology in a simple example.

Keywords: Model Driven Engineering, Cloud Systems, Formal Methods, Requirement Verification

Reference ***

Biographical notes: Francesco Moscato graduated, summa cum laude, in Computer Science Eng. at Univ. of Naples in 2002 and received his Ph.D. in Electronic Eng. in 2005. He is Researcher in Computer Systems at the Second Univ. of Naples. His research interests are in the area of formal verification of critical and distributed systems.

1 Introduction

Cloud Computing has recently emerged as a new paradigm for hosting and delivering services and nowadays researches on service-centric systems mainly focus on Cloud Services. Following the NIST’s definition of Cloud computing[1]:

“Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability...”

it can be realized that Cloud Computing has made processing and storage resources more accessible and cheaper. It provides elastic services, low-cost computing infrastructure (even for building scientific clusters), and scalable data storage to users whose number is increasing each day[2].

In this scenario, users demand more and more complex services, requesting some properties on services execution (Qualities Of services - QoS): providers must build and configure services in order to verify required QoS. At run-time, they have to enact proper monitoring actions able to assure that services execute with promised QoS. The growing of complexity leads to several problems in Cloud Service definition. One of the key features of Cloud Computing is the capability of providing resources on-demand. Automated service provisioning able to allocate and manage resources satisfying service goals is an open research challenge[3]. A methodology to analyze reachability of goals with given QoS is appealing particularly for automatically building target services [4, 5].

In addition, assuring on-demand provision of resources leads to the problem of building Value Added Services (VAS) by using existing ones: given QoS are required also on composed services and a way to extend requirements verification and modeling on VAS is needed.

On the other hand, Multi-Agent Systems (MAS) Models are promising formalisms for description and analysis of composite Cloud services. Both Cloud services and multi-agents systems are distributed systems. Heterogeneity (of platforms, systems and vendors) and complexity in cloud services involve the implementation of some kind of intelligence in order to make them more flexible, adaptive and autonomous[6, 7].

Multi-Agent Systems (MAS)[8] paradigms are a good choice for designing and developing complex systems [9,
Since they seem to cope with their increasing complexity, MAS can provide models of Cloud System where several autonomous components and resources cooperate for implementing complex services. Several methodologies have been proposed for MAS design and development [13, 14]. However, software engineering has not provided yet any approach to model and verify their dependability during all the life cycle. Because of their criticality, Model Driven Engineering (MDE) approaches are appealing when dealing with complex systems: producing designs correct by construction[15] where requirements are validated during all life cycle is useful. The need of MDE methodology in design, verification and even run-time monitoring of cloud systems is growing up [16, 17, 18].

The framework MetaMORP(h)OSY (Meta-modeling of Mas Object-based with Real-time specification in Project Of complex SYstems) is based on formal modeling and analysis of MAS systems. A UML profile compliant with MAS models allows for modeling Cloud components. Model Transformation algorithms translate design models into formal one for analysis purposes. MetaMORP(h)OSY is based on Papyrus [19]. It defines profiles for the definition of a modeling language for real-time MAS description. Previous works ([20, 21, 22, 23]) describe the architecture of the frameworks and the methodology on which it is based. The papers describe the application of the methodology to several example. This is the first work describing the modeling profile used for services description and it deepens the results reported in [20]. Verification at every life-cycle step is performed by implementing translation algorithms which translate design, simulation and run-time descriptions into formal models [24, 25, 26, 5].

UML-based diagrams and modeling profiles enable the description of Cloud services components and behaviors. The models allows for:

- requirements verification (i.e. QoS) of composed services at design phase;
- generation of stub code from design models for implementation aims;
- monitoring and verification of Cloud VAS at run-time.

This work describes a novel modeling profile for design, verification and development of Cloud Services. It describes the profile and its integration into the framework. Another contribution is the description of a methodology able to design composite cloud services and to verify their requirements. We report a simple example in order to show the application of the presented approach.

The paper is organized as follows: section 2 introduces the software architecture of MetaMORP(h)OSY framework; section 3 describes its modeling profile; the methodology is applied to a simple example in section 4. Section 5 contains an analysis of the state of the art. Finally section 6 reports some concluding remarks.

2 MetaMORP(h)OSY

MetaMORP(h)OSY[20, 21, 22, 23] is a framework that supports MDE modeling and verification of complex systems. It aims at following all design and development steps of their life cycles. It offers a methodology for verification of requirements by means of Model Transformation, formal modeling and analysis.

The software architecture of MetaMORP(h)OSY framework is depicted in Fig.1.
Modeling Profile

As described before, MetaMORP(h)OSY allows for the modeling of systems by using a MAS paradigm. In addition, the framework allows for the use of the same formalism for system description and requirements specification: it provides a graphical language to users at design phase and requirements specification becomes part of the system model. This language is formally defined by a meta-model[32]. The name of the meta-model, with all the rules used to compose correct models, is (inheriting the UML jargon) modeling Profile.

3 Modeling Profile

MetaMORP(h)OSY modeling profile must allow for the definition of:

- Structural and behavioral views of the system, as well as interactions among components;
- Properties and Requirements to verify on the system (both at design and run time);
- Methods, Techniques and metrics for analyzing or measuring properties and requirements;
- Expected and measured workloads of the systems.

In addition, the modeling profile is able to specify temporal behavior of agents, as well as real-time properties. In fact, the name of the modeling profile is RT-AML (Real-Time Agent modeling Language) since it was originally intended for describing real-time agent.

The modeling methodology in MetaMORP(h)OSY is based on AML[33], that describes MAS by using a UML-based language. As described in[34], the original support for describing timed behaviors of agents in AML is poor. MetaMORP(h)OSY uses a Beliefs, Desires, Intentions (BDI) logic in order to describe Agents. Beliefs of agents, Goals they want to achieve and the available Plans to reach the goals defines agents structures.

For properties and requirements definition, RT-AML inherits and extends MARTE[27] profile. MARTE is an extension of UML 2.0 to support MDE of real-time systems. In particular, it addresses time schedulability and performance description.

Making RT-AML compliant with MARTE is appealing for several reasons: first, models developed with the new profile are interoperable with other frameworks supporting MARTE standard; furthermore, MARTE explicitly addresses the problem of choosing the kind of analysis to perform, since it allows the definition of different kinds of analysis models.

MARTE divides profile into several packages, containing a set of stereotypes used for modeling real-time systems. The packages inherited by the RT-AML profile are: the Core Elements, which contain the main classes for real-time systems design; the Generic Quantitative analysis modeling (GQAM), that contains elements related to models analysis; the Schedulability analysis modeling (SAM), where the elements needed to define schedulability analysis on models are defined; Performance analysis modeling (PAM) which is similar to SAM, but it is used to enable performance modeling; and the Generic Resource modeling (GRM), used to model resources.

The Marte GQAM package defines agents behaviors (i.e.: plans). In particular, one of the elements contained in this package is the Action stereotype. The RT-AML profile inherits this stereotype to define real-time actions during the execution of real-time activities.

The RunTimeContext sub-package, is used in RT-AML to define real-time constraints and timing behaviors of agents.
GQAM, SAM and PAM contain three sub-packages: Workload, Resources and Observers. MetaMORP(h)OSY uses the elements in these packages to define agent resources, state and beliefs (Resources). When users requests requirements verification, the verification process requires the analysis of one or more properties on the model. The framework instances an Observer to monitor and/or evaluate a property. Finally, Workloads elements defines expected or measured loads of the modelled systems. Fig.3 depicts the structure of this last package.

SAM elements are specialized for Schedulability analysis. As described late, SAM observer on RT-AML models analyze the reachability of a goal under real-time constraints. The same for PAM, but for performance.

3.1 Structural and behavioral description

RT-AML profile uses four diagrams for MAS behaviors description: Class diagrams, RT-Agent Diagrams, RT-Activity diagrams and RT-Sequence diagrams. The Class diagrams, are the same of UML diagrams. They are used when it is not useful to describe objects as agents (for example, for passive entities). The RT-Agent diagrams are the core of the RT-AML profile. They describes agents structures, goals and beliefs. In addition, they declares the actions and the plans they can execute. The RT-Activity diagrams allow for description of agents plans. The RT-Sequence diagrams describe agents collaborations (and/or competitions). Their main goal is to define exchanges of messages and events (real-time stimula) during execution of agents plans.

The main elements of RT-Agent diagram meta-model are depicted in Fig.4
They are: AgentRT, PlanRT, DgoalRT and BeliefRT.

AgentRT stereotype defines agents structures. They are classes hence they refer to Class and Classifier UML base classes. In addition relationships with ObjectNode and ObjectFlow have to be defined in order to apply the stereotype to classes and objects in the RT-Activity and RT-sequence diagrams. RT-Agent inner state and other
resources (like sensors etc.) are modeled as MARTE Resources. AgentRT stereotype includes some useful properties like PlanningSwitchTime (the time needed for an agent to replan actions), Thinking Time (the time usually spent by agents when a choice must be taken), Action Time (the time used to begin actions executions), Reaction Time (the time elapsed between thinking and action time), Start and Finishing Times (the times when agents are started and terminated) and Priority of agents.

PlanRT stereotype inherits the MARTE GaScenario: this allows the modeling of plans as Steps composition. Plans are associated to goals: each plan is intended as a composition of actions (MARTE steps) aiming at reaching a goal state. Hence, PlanRT has some references (Pl.dgref) to DgoalRT stereotype. Some properties for this stereotype are: Arrival Time, the time when the Plan is activated; Execution Time: the time needed to execute a plan; Start and Finish Time, the times at which the plan is started and finished; the Deadline, the deadline for the plan execution; PlanType, used to specify hard or soft real time behavior.

DgoalRT stereotype defines decidable goals for AgentRT. A decidable goal is a goal whose reachability can be determined under real-time constraints. Some DgoalRT properties are the following: MaintainingGoal Time, the time the goal has to be maintained active; Deadline, the deadline for the reaching the goal; GoalType, that is used to specify hard or soft real time constraints for goal reachability.

Stereotypes needed to model agents plans are defined in the RT-Activity diagram profile, that is show in Fig.5. RT-Activity diagrams inherit all elements from UML activity diagrams, and re-define states and transitions to enable real-time constraint specification. In particular, ActionStateRT, InitialState, FinalState, TransitionRT, SendObjectRT and ReceiveObjectRT are the new stereotypes for this diagram.

InitialState and FinalState are the initial and the final states for the Activity Diagram. Usually FinalState refers to a DgoalRT. When starting a new plan, an agent has some Beliefs of its environment. Beliefs also compose its initial state and are specified as properties of InitialState stereotype. The usual time-related properties relate to the final (goal) state.

ActionStateRT is a step that composes the whole agent plan. It inherits the classic UML Activity State, but also GaStep from MARTE profile. An Action State is an activity State, which is subject to real-time constraints. In fact, this stereotype includes the following properties: the ExecActionTime, the time needed for the execution of the action and the TimeOut.

Two particular types of ActionStateRT are the SendObjectRT and ReceiveObjectRT, which are used when messages and events are sent and received to and from other agents.

TransitionRT is a stereotype used to define transitions in activity diagrams with timing constraints. This stereotype too includes Time-related properties.

Finally, the main elements for Sequence Diagrams are: StimulusRT and MultiStimulusRT (see Fig.6). The StimulusRT stereotype is used to define stimuli with real time constraints in Sequence diagrams. The main properties for this element are: the StartTime, the time when the decision of sending a message is taken by an agent during the execution of a plan; the SendTime, the time when the message
is sent; the TransmissionTime, the time needed for messages transmission; the StartReceiveTime, the time when the receiver begins to receive the message; the StartEndTime, the time when the receiver ends to receive the message; the Deadline, the deadline for transmission; the MessageSync, it is used to specify if the message is synchronous or not. MultiStimulusRT inherits the StimulusRT stereotype and it is used when defining redundancy in messages reception.

### 3.2 Properties, Requirements and Metrics

Properties-Requirements and Structural-behavioral profiles are separated: definition of system models are independent from requirements specification and verification. This allows for requesting different requirements on the same system at different time and this is the reason for defining requirements and properties specification outside Agents stereotypes.

Fig.7 depicts the Property profile.

A Requirement is a list of Properties that Observers will analyze on the models (or at run-time). Properties include Functional and NonFunctional ones. This profile declares stereotypes for several properties like Availability, Reliability, Schedulability, Performances etc. Metrics are associated to non functional properties. In fact it is possible to specify, in a MetaMORP(h)OSY model, the metrics to use for properties definition, evaluation and monitoring. Users can extend the profile if necessary adding more metrics and properties. For example, Fig.8 shows how Property Profile (but also
Agent Profile) can be extended in order to new properties or to specialize the existing ones.

3.3 Measuring Properties and Requirements Verification: the Observers

The bottom of Fig. 7 reports the part of the profile that defines Observers stereotypes. The profile divides Observers in Design_Time and Run_Time Observers. They can be further specialized depending on translation algorithm and analysis methods used to analyze properties and requirements on system components. Observers can be associated to Structural and behavioral components and obviously to the property to analyze or to monitor, as well as to the metrics used to collect and produce results.

4 Example

Let us suppose a user needs to implement a high-availability storage service. In order to improve availability multiple providers can host and replicate services, reducing chances of failure.

To this purpose, let us assume that we need to realize a Composed Cloud Services with replicas to improve availability. Services used in composition are:

(a) SimpleStorage: it has the goals (SimpleReadOK
and SimpleWriteOK) of completing a read or a write operations (SimpleRead and Simple Write). (b) MultipleStorage which has the goals of performing a Read or Write operation (ReadOk and WriteOk Goals respectively) in a redundant configuration of storage; (c) Connection: this agent models the link connection among services. Fig. 9 shows Agent Diagram of Services.

Here we show how MetaMORP(h)OSY can be used for describing and analyzing this Composite Service. Notice that even if it is not specified by user, the Composite Service has to assure a given response time to Users’ Request. Hence, we need to check two properties: a deadline in the response time of the composite service and the availability of the service when replicas are introduced. The analysis of this section involves modeling Phase.

In this scenario, we model a high available storage service. Let us assume that no available vendor can meet requirements and that composition of existent services with replicas is the solution we need. Since storage services may belong to different vendors, three Storage Wrappers services are used as common interface towards true storage services (in the following the term Storage will be used to address Storage Wrappers). In a redundant configuration, data comes from multiple sources (SimpleStorage) and MultipleStorage service reaches its goal if source storages pass a majority data check. A Voter service allows for simple error correction. This is used to collect data from the three Storage

---

**Figure 8** Extending Profiles
to perform a 2 on 3 voting on retrieved data; if data retrieved from storage are the same at least for 2 storage services (this includes also unavailable services which provide no data at all), it is forwarded to the user as correct data.

In order to model these agents and their interactions in MetaMORP(h)OSY, three kinds of diagrams have to be provided: the Agent Diagram, the Activity Diagrams for agents plans, and a Sequence Diagrams for describing interactions among agents.

The Agent Diagram describes the MAS system structure. Here classes with proper stereotype represent Agents, their Plans, their Beliefs and relations among them. Fig.9 shows the Agent Diagram with SimpleStorage, MultipleStorage and Connection agents.

The stereotype AgentRT applies to classes in the diagram. It is defined in the MetaMORP(h)OSY modeling profile and it is used to define properties that can specify (real-time) agents in a UML model. By associating this stereotype to a class in a MetaMORP(h)OSY model, it is possible to specify agent’s temporal characteristics. Composition associations exist between Agent classes and their plans, beliefs and goals. In particular, PlanRT, BeliefRT and DGoalRT stereotypes are used. In Tab.1 agents with their plans, and with beliefs and goals are listed.
Table 1  Agents, Plans, Beliefs and Goals

<table>
<thead>
<tr>
<th>AgentRT</th>
<th>PlanRT</th>
<th>BeliefRT</th>
<th>DGoalRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voter</td>
<td>RetrieveData: When a user requests for data for the voter sends three</td>
<td>Data[i]: it includes resource, request Storage services URI temporary</td>
<td>DataRetrieved: this goal is reached when the 2 on 3 voting on retrieved data reports</td>
</tr>
<tr>
<td></td>
<td>parallel requests to the Storage services. when data arrive, a 2 on</td>
<td>retrieved data etc. a success</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 voting procedure is enacted in order to send correct data to users.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SimpleStorage</td>
<td>simpleRead/Write: this plan is enacted when users request for stored data</td>
<td>BkdData: this contains the users, certificates for storage accesses etc.</td>
<td></td>
</tr>
</tbody>
</table>

The Agent Diagram in Fig.9 describes agents structures, listing their plans, beliefs and goals related to each plan. In order to complete the model, we need a dynamic description of agents behaviors. In MetaMORP(h)OSY this is done by mean of particular activity and sequence diagrams.

The MetaMORP(h)OSY profile defines several stereotypes for messages and timed activities in the activity diagrams. This allows for the analysis of timed behavior and interaction of agents. At the state, each PlanRT in the AgentDiagram is associated to an Activity Diagram that describes the action enacted during plan execution. Each action includes the declaration of expected execution time, messages awaited from and sent to other agents, resources and beliefs.

Activity diagrams describe agents behaviors concerning their plans, but different execution paths are possible depending on different use cases. In order to define which events and messages are involved in a particular use case, a Sequence Diagram is reported in MetaMORP(h)OSY. The main purpose of this diagram is the definition of agents interactions in a use case.

In Fig.10 the Activities related to the plan MultiRead are reported.

MultiRead is the first plan enacted by a MultiStorage agent in the use case described in this paper. It first sends three parallel requests to three SimpleStorage services in order to retrieve data requested by the user \((RequestData[i], i \in \{0, 2\})\). Each following ActionStateRT has a deadline associated: a retrieval fail if deadline expires. A Vote procedure compares retrieved Data. If at least 2 retrieved data are identical, the DataRetrieved action state (and the related DgoalRT reported in Fig.10) is reached. TransitionRT elements reports Guars. Those ending with exclamation mark, are related to messages sent during the plan execution to other agents; those ending with a question mark are related to messages that are awaited by ActionsRT before firing the Transition. In Fig.10, during the MultiRead Plan, Three requests for data are sent to three different SimpleStorage agents, and then three responses are awaited from the same agents. Notice that timeouts are properties of TransitionRT stereotype.

It is clear that Plans require communication among agents. Agent Collaborations are Modeled by using RT-AML Sequence diagram.

For example, interactions among agents are reported in Fig.11.

All messages in the sequence are StimulusRT which is a MetaMORP(h)OSY stereotype able to represent the timed behavior of messages exchanged during the execution of the use case with three SimpleSotage Services. StimulusRT messages relate to messages sent and received in the actions of the activity diagrams; they represent the messages sent considering the use case in analysis. Filled Lifeline under the MultipleService agent indicates that not all the messages from Storage services have to reach the MultipleServices. If for some reason (deadlines expiration, broken communication links etc.) it does not receive all messages, the Vote action is executed (probably leading to an operation failure).

The next step is the definition of an Observer Diagram for the declaration of analyses and monitoring activities to enact. Fig.12 shows the definition of an Observer that will perform an availability analysis on the model described in this section. The Availability Observer uses the stereotype of a Sharpe Observer. It intends to evaluate the availability of the MultiStorage agent in terms of percentage. Since Transmission Availability of Connection Agent, and Storage Availability of Simple Storage Agent are not linked to any Observer, they represent simple (input) availability values for the previously mentioned agents. The Sharpe Observer translates the design model into Fault Trees for analyzes purposes. The translation in this case is Horizontal and the SHARPE[35] framework is used in order to analyze the property.

The produced Fault Tree is depicted in Fig.13.
Some results about the analysis of the availability of the replicated service is reported in Tab.2
Notice that the Availability of MultiStorage in this simple example polarized results since they are a single point of failure in the system.

5 State of the art

Combining Cloud services to build VAS in Cloud environments is a recent problem and an open research challenge. Interoperability is still a problem for Cloud services of different providers, and composition is usually approached with the standard methods (i.e. Orchestration) used in Web Services and Service Oriented Architectures [36].

Anyway, Agent-Based paradigm in defining and implementing cloud services is arising because it copes well with characteristics of Cloud systems[6]. In this scenario an increasing number of cloud platforms are trying to provide services by using autonomous, self-organizing MAS. For example, Authors in [37] use a
MAS system to provide cloud services. The need for Choreography in composition in this work is clear, but a methodology and a language have not been yet defined for modeling composition of services.

In addition, the other problem in modeling and providing Cloud services is that users request given level of services in terms of Service Level Agreements (SLA), but existing public clouds provide very few guarantees in terms of performance and dependability [38]. Several approaches have been proposed in the last years in order to manage performance and dependability in terms by signing SLA between providers and users. In [39], authors propose the automation of SLA establishment based on a classification of cloud resources in different categories with different costs. However, this approach does not provide guarantees in terms of performance, dependability, etc. A similar approach for SLA enforcement, based on classes of clients with different priorities, is presented in [40]. Here again SLA is guaranteed only in terms of best-effort policies. Other approaches for SLA management which uses heuristics are proposed in [41], where again the work provides best-effort without strict guarantees on SLA. A SLA management approach with measured Level of service is presented in [42], but it addresses only Service as a Service (SaaS) Clouds. The need for a managing
guaranteed Service Level Agreement becomes larger in the last years. Authors in [43] proposes a new Cloud Model, entirely based on the provision of Cloud services which fulfill measured properties at run-time. The main problem of these approaches is that SLA negotiation starts from pre-prepared SLA by providers. There is no attempt on the part of providers to configure existing services based on the service levels required by the end-user. Furthermore, nothing has been done for negotiating SLA of composed Cloud Services. This because most of the systems for the management of SLAs are based on monitoring of components at run-time, while the configuration (and possibly the creation) of services that verify requirements is a process that involves the use of appropriate analysis techniques already at design phase. Model Driven Engineering (MDE) Techniques sounds good to reach the goal of SLA verification and monitoring since they cover all the life cycle of the services (from design to run-time). There have been several attempts to apply MDE techniques to the study and development of Cloud systems and services. In [44] authors propose to implement Model Driven Engineering environments as a Service to use on the Cloud. Although it is only a proposal, the idea of providing services that are build based on Design requirements and not already deployed by providers is appealing. The need for verification of requirements by using formal methods at design phase is reported in [18]. There an optimization technique based on formal methods for performance evaluation is described. What is missing in this work is the enactment of the optimization based on users requests and SLA managements. Furthermore, once the services are optimized at design phase, there is no way to reconfigure the system monitoring services execution. The need of MDE techniques for defining Cloud Systems and services is reported in [45]. The work outlines the lack of support for heterogeneous Cloud providers, the limited matchmaking between application requirements and Cloud capabilities, the lack of meaningful cross-platform Cloud resource descriptions, the lack of lifecycle management of Cloud applications, the inadequate cross-layer monitoring and adaptation based on event correlation.

In this context MetaMORP(h)OSY tries to overcome all the reported limitations in applying MDE techniques to the management of Cloud Systems and Services. First of all, MetaMORP(h)OSY is able to address different kind of requirements (reliability, availability, performances, security etc.). Requirements verification is enacted in all the life cycle of services, hence it is not limited only at design phase. MetaMORP(h)OSY components can be embedded at run-time in order to allow for requirement verification also during services execution, acting not only as a development tool, but also providing a mean for produce run-time monitors of services depending on users requests. The use of MetaMORP(h)OSY methodology, it is possible to build and configure services depending on users requirements (i.e. SLA) and not only to choose the “best-coping” service for user.

6 Conclusions

This work describes a methodology that supports the design, verification and validation of requirements. The methodology covers the entire life cycle, going beyond the abstraction, and providing a mean to analyze the system at design and run time. The MetaMORP(h)OSY methodology and framework have been introduced and applied to the design and the analysis of a simple composite Cloud Service. It has been shown how MetaMORP(h)OSY suites well the MAS nature of cloud components and it is possible to define Observers on models for system analysis. Future works include the design and the analyzes of high available and fault tolerant scenarios of complex systems.

References


