Abstract—Service Oriented Architectures and service-centric models arose in the last years as a standard de-facto in IT enterprises for offering practically any kind of services to users world-wide. In particular Cloud-based models provide means for realizing and distributing everything-as-a-service, including infrastructures, hardware and software platforms and services. Even if at the moment 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 methodology based on Multi-Agent Models which allows for description, composition and verification of requirements of Cloud-based services. The methodology uses a modeling profile able to describe services as agents in a multi-agent environment and it is based on Model Driven Engineering (MDE) techniques. The proposed methodology includes a verification process for requirements that exploits formal methods during the whole life cycle of services.

I. INTRODUCTION

Nowadays, researches on service-centric system engineering mainly focuses on Cloud Services, since they have Cloud Computing has recently emerged as a new paradigm for hosting and delivering services. 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. In addition they have to enact proper monitoring actions able to assure that services are provided with promised QoS also at run time. A methodology to analyze reachability of goal services with given QoS is appealing if it can be used to build automatically the target service [1], [2].

In this scenario Models based on Multi-Agent Systems (MAS) Models are promising formalisms for the analysis of composite Cloud services. MetaMORP(h)OSY framework is based on Papyrus [4] and defines profiles for the definition of a modelling language for real-time MAS description. Verification at every life-cycle step is performed by implementing translation algorithms which translate design, simulation and run-time description into formal models [5], [6], [7], [2].

This paper shows how it is possible to use MDE and MAS models in order to describe composition of cloud services meeting given availability QoS and how it is possible to verify requirements at design phase while configuring (existing) services to offer a VAS. A simple example will be reported in order to show the application of the presented approach.

This paper is organized as follows: section II contains an analysis of the state-of-the-art in Model Driven Engineering methodologies and tools applied to Cloud systems development and validation; the MetaMORP(h)OSY overall architecture is described in section III. Section IV shows how it is possible to use MetaMORP(h)OSY methodology to describe and validate cloud services composition. The methodology is applied to a simple example in section V. Finally section VI contains some concluding remarks.

II. 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 [8].

Anyway, Agent-Based paradigm in defining and implementing cloud services is arising because it copes well with characteristics of Cloud systems [9]. In this scenario an increasing number of cloud platforms are trying to provide services by using autonomous, self-organizing MAS. For example, Authors in [10] 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 [11]. 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. A similar approach for SLA enforcement, based on classes of clients with different priorities, is presented in [12]. The need for a managing guaranteed Service Level Agreement becomes larger in the last years. Authors in [13] proposes a new Cloud Model, entirely based on the provision of Cloud services which fulfill measured properties at run-time. 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 [14] 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 which 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 [15]. 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 [16], where are outlined 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.

III. MetaMORP(h)OSY

In the following both the Architecture of the MetaMORP(h)OSY framework and the MDE methodology on which the Architecture is based will be described.

A. MetaMORP(h)OSY Modelling Methodology

The MetaMORP(h)OSY Modeling Methodology (MMM) is based on AML[24], that describes MAS by using a UML-based language. As described in [25], the original support for describing timed behaviors of agents in AML is poor. Agents behaviors are defined by using Beliefs, Desires, Intentions (BDI) logic. Agents structures are defined in terms of beliefs of agents, goals they want to achieve and the available plans to reach the goals. The language defined in MMM for Agent description is called RT-AML (RealTime AML) since it was originally intended for describing real-time agent.

As described before, RT-AML is a UML-based language. The UML profile on which the RT-AML metamodel is based is described in the following. RT-AML uses four diagrams for MAS behaviours description: the Class diagrams, 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, the core of the RT-AML profile, are used to describe agents structures, goals and beliefs, their inner states and to declare which actions and plans they can execute; the RT-Activity diagrams that allows for the description of agents plans; and the RT-Sequence diagrams that are used in order to describe agents collaborations (and/or competitions). Their main goal is to define how messages and events (real-time stimuli) are exchanged during agents plans executions.

Main Elements in RT-Agent diagram are: AgentRT, PlanRT, DgoalRT and BeliefRT.

AgentRT stereotype is used to define agents structures. They are managed as classes hence they have to be related
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 modelled as MARTE Resources. In addition, an AgentRT is composed by BeliefRT, PlanRT and DgoalRT. AgentRT stereotype properties include: PlanningSwitchTime, the time needed for an agent to replan actions (if needed); 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 (if defined) when agents are started and terminated; Priority, the priority of agents.

PlanRT allows for the modelling of plans as Steps (i.e. actions) composition as it was introduced before. Plans are associated to goals: each plan is intended as a composition of actions aiming at reaching a goal state.

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 behaviour.

DgoalRT stereotype is used to define decidable goals for AgentRT, where 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 reaching the goal; GoalType, that is used to specify hard or soft real time constraints for goal reachability.

RT-Activity diagrams inherits 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, since usually agents actions are enacted in order to reach their goals. When starting a new plan, an agent has some Beliefs of its environment. Beliefs also composes its initial state and are specified as properties of InitialState stereotype. The usual time-related properties are instead related 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 basically an activity State which is subject to real-time constraints. In fact, this stereotype is characterized by the following properties: the ExecActionTime, the time needed for the execution of the action; 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. Time-related properties are defined also for this stereotype.

Finally, the main elements for Sequence Diagrams are: StimulusRT and SyncType.

The StimulusRT stereotype is used to define stimulus 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 (SyncType).

V. Example

Let us suppose a user needs to implement an high-availability storage service. In order to improve availability services can be hosted and replicated to multiple providers, on multiple locations, reducing chances of failure.

To this purpose, let us assume that a Composed Cloud Services will be build. Replicas or different storage services will be used in the composed service in order to increase availability. Services used in composition are: (a) SimpleStorage: it has the goals (SimpleReadOK and SimpleWriteOK9 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; (3) Connection: this agent models the link connection among services. Fig.1 shows Agent Diagram of Services

Here we show how MetaMORP(h)OSY can be used for describing and analyzing this Composite Service.

In this scenario a storage service with given availability is built. Let us assume that no available vendor can meet requirements and that a Composed Cloud Service is built by using available storage services. In order to increase availability, in this example, data are replicated on different storage services. In a redundant configuration, data are collected from multiple sources (SimpleStorage) and MultipleStorage service reaches its goal if a majority voting is completed on source storage.

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.

In the Agent Diagram the MAS system structure is described. Here classes with proper stereotype represent Agents, their Plans, their Beliefs and relations among them. The Agent Diagram with SimpleStorage, MultipleStorage and Connection agents is shown in Fig.1.

The Agent Diagram in Fig.1 describes agents structures, listing their plans, beliefs and goals related to each plan. In order to complete the model a dynamic description of agents behaviors has to be provided. 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 which describes the action enacted during plan execution. Each action is described in terms of expected execution time, messages awaited from and sent to other agents, resources and beliefs involved in the action.

Activity diagrams describe agents behaviors in regard to 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.2 the Activity related to the plan **MultiRead** is reported.

This 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 date requested by the user ($\text{RequestData}(i), i \in \{0, 2\}$). Each following ActionStateRT has a deadline associated: a retrieval fail if deadline expires. Retrieved Data are then compared by a Vote procedure. If at least 2 retrieved data are identical, the DataRetrieved action state (and the related DgoalRT reported in Fig.2) is reached.

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

For example, interactions among Voter service and Storage agents (which have not been described for brevity) are reported in Fig.3.

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 SimpleStorage Services. **StimulusRT** messages are related to messages sent and received in the actions of the activity diagrams; they represent the messages that are actually 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).

Once the whole model has been defined in MetaMORP(h)OSY, it provides a way to choose the property to analyze and the type of analysis to perform. To this purpose, the MetaMORP(h)OSY modeling profile provides a set of stereotypes which are able to define properties to analyze on the system. They include functional and non-functional properties, like state reachability, schedulability, availability, performance, fault tolerance etc. In addition, the type of the analysis to perform on the model is defined by means of Observers. Depending on the property and on the observer, proper model transformation algorithm is executed in order to translate the RT-AML model into a model that can be exploited for analysis. For example, for what Availability concerns, in the example an Observer is used to translate the model into Fault Trees and the SHARPE[26] framework is used in order to analyze the property.

The produced Fault Tree is depicted in Fig.4.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.95</td>
<td>0.9999</td>
<td>0.95</td>
</tr>
<tr>
<td>2</td>
<td>0.95</td>
<td>0.9999</td>
<td>0.9975</td>
</tr>
<tr>
<td>3</td>
<td>0.95</td>
<td>0.9999</td>
<td>0.999875</td>
</tr>
<tr>
<td>4</td>
<td>0.95</td>
<td>0.9999</td>
<td>0.9999</td>
</tr>
<tr>
<td>5</td>
<td>0.95</td>
<td>0.9999</td>
<td>0.9999</td>
</tr>
<tr>
<td>1</td>
<td>0.98</td>
<td>0.9999</td>
<td>0.98</td>
</tr>
<tr>
<td>2</td>
<td>0.98</td>
<td>0.9999</td>
<td>0.9996</td>
</tr>
<tr>
<td>3</td>
<td>0.98</td>
<td>0.9999</td>
<td>0.9999</td>
</tr>
<tr>
<td>4</td>
<td>0.98</td>
<td>0.9999</td>
<td>0.9999</td>
</tr>
<tr>
<td>5</td>
<td>0.98</td>
<td>0.9999</td>
<td>0.9999</td>
</tr>
</tbody>
</table>

Some results about the analysis of the availability of the replicated service is reported in Tab.I

Notice that the Availability of Broker and Voter in this simple example polarized results since they are a single point of failure in the system.

VI. CONCLUSIONS AND FUTURE WORK

In this paper 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 analyses of high available and fault tolerant scenarios of complex systems.

REFERENCES


