Scalable Computing: Practice and Experience
Volume 13, Number 1, pp. 29–44. http://www.scpe.org
ISSN 1895-1767

ENABLING MODEL DRIVEN ENGINEERING OF CLOUD SERVICES BY USING MOSAIC ONTOLOGY

FRANCESCO MOSCATO*AND B. DI MARTINO AND R. AVERSA†

Abstract.
The easiness of managing and configuring resources and the low cost needed for setup and maintaining Cloud services have made Cloud Computing widespread. Several commercial vendors now offer solutions based on Cloud architectures. More and more providers offer new different services every month, following their customers needs. A way to provide a common access to Cloud services and to discover and use required services in Cloud federations is appealing. mOSAIC project addresses these problems by defining a common ontology and it aims at developing an open-source platform that enables applications to negotiate Cloud services as requested by users. Anyway the increasing complexity of services required by users in Cloud Environments usually needs the definition of composite, value added services (VAS). Usage patterns and Use Cases definitions help in defining VAS, but a way to assure that new services reach the required goals with proper qualitative and quantitative properties has to be provided in order to validate design and implementation of composite services. In this paper mOSAIC Ontology is described and the MetaMORP(h)OSY methodology and framework are introduced. The methodology uses Model Driven Engineering and Model Transformation techniques to analyse services. Due to the complexity of the systems to analyse, the mOSAIC Ontology is used in order to build modelling profiles in MetaMORP(h)OSY able to address cloud domain-related properties.

Key words: Cloud, Ontology, Model Driven Engineering, Validation And Verification.

1. Introduction. Cloud Computing is an emerging model for distributed systems. It refers both to applications delivered as services and to hardware, middleware and other software systems needed to provide them. Nowadays the Cloud is drawing the attention from the Information and Communication Technology (ICT) thanks to the appearance of a set of services with common characteristics which are provided by industrial vendors. Even if Cloud is a new concept, it is based upon several technologies and models which are not new and are built upon decades of research in virtualization, service oriented architecture, grid computing, utility computing or distributed computing ([16, 26, 43]). The variety of technologies and architectures makes the Cloud overall picture confusing [16]. Cloud service providers make resources accessible from Internet to users presenting them *as a service*. The computing resources (like processing units or data storages) are provided through virtualization. *Ad-hoc* systems can be built based on users requests and presented as services (*Infrastructure as a Service*, IaaS). An additional abstraction level is offered for supplying software platforms on virtualized infrastructure (*Platform as a Service*, PaaS). Finally software services can be executed on distributed platforms of the previous level (*Software as a Service*, SaaS). Except from these concepts, several definitions of Cloud Computing exist ([34, 5, 17, 13, 30, 25]), but each definition focuses only on particular aspects of the technology. Cloud computing can play a significant role in a variety of areas including innovations, virtual worlds, e-business, social networks, or search engines but it is actually still in its early stages, with consistent experimentation to come and standardization actions to effort. In this scenario, vendors provide different Cloud services at different levels usually providing their own interfaces to users and Application Programming Interfaces (APIs) to developers. This results in several problems for end-users that perform different operations for requesting Cloud services provided by different vendors, using different interfaces, languages and APIs. Since it is usually difficult to find providers which fully address all users needs, interoperability among services of different vendors is appealing.

Cloud computing solutions are currently used in settings where they have been developed without addressing a common programming model, open standard interfaces or adequate service level agreements or portability of applications. Neglecting these issues current Cloud computing forces people to be stranded into locked, proprietary systems. Developers making an effort in Cloudifying their applications cannot port them elsewhere.

In this scenario the mOSAIC project (EU FP7-ICT programme, project under grant #256910) aims at improving state of the art in Cloud computing by creating, promoting and exploiting an open-source Cloud application programming interface and a platform targeted for developing multi-Cloud oriented applications. One of the main goal is that of obtaining transparent and simple access to heterogeneous Cloud computing resources and to avoid locked-in proprietary solutions.

*Second University of Naples, Dep. of European and Mediterranean Studies, Via del Setificio 15, 81100 Caserta, Italy (francesco.moscato@unina2.it).
†Second University of Naples, Dep. of Information Engineering, Aversa, Italy

In order to attain this objective a common interface for users has to be designed and implemented, which should be able to wrap existing services, and also to enable intelligent service discovery. The keystone to fulfil this goal in mOSAIC is the definition of an ontology able to describe services and their (wrapped) interfaces.

In addition, users also require given qualities of services (QoS) and Cloud providers have to build services *on demand* depending on specified QoS. The provisioning should be coupled with proper monitoring systems which assure that services are provided with promised QoS also at run time.

In a Cloud scenario, Service Level Agreement (SLA) is a way to establish a contract between users and service providers where providers assure the execution of services with specified requirements. SLA is a way to express these requirements formally and formal methods can be exploited in order to verify them.

Automated service provisioning able to allocate and manage resources satisfying service goals is an open research challenge [46]. A methodology to analyse reachability of goal services with given requirement is appealing if it can be used to build automatically the target service.

Multi-agent systems(MAS) represent a model for designing and developing complex systems [6, 8, 47, 15] since it seems to cope with their increasing complexity. MAS can be successfully used to provide a model of Cloud System where several components and resources cooperate for providing complex services to users. Several methodologies have been proposed for MAS design and development [18, 9]. 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 approaches are appealing when dealing with complex systems. Producing designs *correct by construction* where requirements are validated during all life cycle is useful. The methodology introduced here, MetaMORP(h)OSY, (Meta-modelling of Mas Object-based with Real-time specification in Project Of complex SYstems) inherits, improves and extends the one described in [28]. It is based on formal modeling and analysis of MAS systems. Cloud components for each service are modeled by using and extended UML profile compliant with MAS models. The main model is then analyzed by means of formal models that are obtained from the UML model with model transformation algorithms.

As explained below, the capability of representing Cloud services components and behaviours with UML-based diagrams is appealing since several services description and use cases are expressed by UML diagrams [36]. In addition, requested QoS may be represented as requirements that can be validated in a MDE methodology.

MetaMORP(h)OSY framework is based on Papyrus [41] and defines profiles for the definition of a modelling language for real-time MAS description. The language is compliant with OMG MARTE [12] specification, in order to make *MetaMORP(h)OSY* compliant with other tools supporting the standard. Verification at every life-cycle step is performed by implementing translation algorithms which translate design, simulation and run-time description into formal models.

Even if MetaMORP(h)OSY is used for (real-time) MAS modelling, it is based on a general methodology. When dealing with particular domains a way for generating modelling profiles based on the transformation of formal models describing the domain of interest is appealing. In mOSAIC, the Cloud ontology is the domain model that can be exploited in order to build missing information in MetaMORP(h)OSY allowing the framework for modelling and verifying requirements on Cloud components.

In particular, the intrinsic hierarchical organization of an ontology is useful when dealing only with particular aspects of the domain. In particular this work focuses on SLA modelling and verification of Cloud services. It will be shown how the mOSAIC Ontology can be used in order to build the part of the MetaMORP(h)OSY modelling profile required for mOSAIC components elements.

This paper is organized as follows: Section 2 shows some motivation for using mOSAIC Ontology for build a modelling profile in MetaMORP(h)OSY. Section 3 introduces the mOSAIC project, Section 4 contains a description of the mOSAIC Ontology; Section 5 describes MetaMORP(h)OSY modelling methodology and Section 6 reports a description of its modelling Profile and describes how mOSAIC Ontology is used in order to enhance it. Section 7 shows how the created profile can be exploited in modelling and verification of QoS of Cloud Services. Finally Section 8 contains some concluding remarks.

2. Motivation. mOSAIC promotes interoperability of cloud services. Building mOSAIC-compliant services requires the use of proper API and components. In addition, mOSAIC allows for the definition of SLA for services. SLA can be considered as requirements that have to be fulfilled at run-time by providers. Using Model Driven Engineering is appealing when dealing with complex cloud services with SLA since MDE allows services components to be built from design model definition. Anyway, the keystone to fulfil interoperability in mOSAIC is the definition of an ontology able to describe services and their (wrapped) interfaces. For searching

and retrieving purposes, mOSAIC services have to be semantically annotated with elements of this ontology. This means that mOSAIC components and SLA definitions have to respect the organization and the structure of the ontology.

Hence an MDE methodology used to design and develop services in mOSAIC should use ontology information in its modelling profile in order to:

- inherit components that are defined as concepts in the ontology with their properties and relationships;
- allow for specification of QoS parameters;
- semantically annotate services components created with the MDE framework.

Using ontology information in a MDE modelling profile is appealing since MDE model transformation techniques enable automatic creation of mOSAIC components and interfaces. In addition, formal verification in MDE assures, at least on the design model, that QoS requirements are respected.

In addition, if proper model transformation techniques are used (like in MetaMORP(h)OSY), verification can also be enacted at run-time by monitors that are created automatically.

In MetaMORP(h)OSY, the mOSAIC ontology has been used to generate part of a modelling profile that is able to describe cloud components in a vision compliant with mOSAIC architecture.

3. mOSAIC Project. The Open Cloud Manifesto [39] identifies five main challenges for Cloud: data and application interoperability; data and application portability; governance and management; metering and monitoring; security.

Actually, the main problem in Cloud computing is the lack of unified standards. Market needs drive commercial vendors to offer Cloud services with their own interfaces since no standards were available at the moment. Vendors solutions have arisen as commonly used interface for Cloud services but interoperability remains an hard challenge, like portability of developed services on different platforms. In addition vendors and open Cloud initiatives spent few efforts in offering services with negotiated quality level.

The mOSAIC project tries to fully address the first two challenges and partially addresses the next two ones by providing a platform which:

- enables interoperability among different Cloud services,
- eases the portability of developed services on different platforms,
- enables intelligent discovery of services,
- enables services composition,
- allows for management of Service Levels Agreement (SLA).

The architecture of mOSAIC platform is depicted in Fig.3.1:

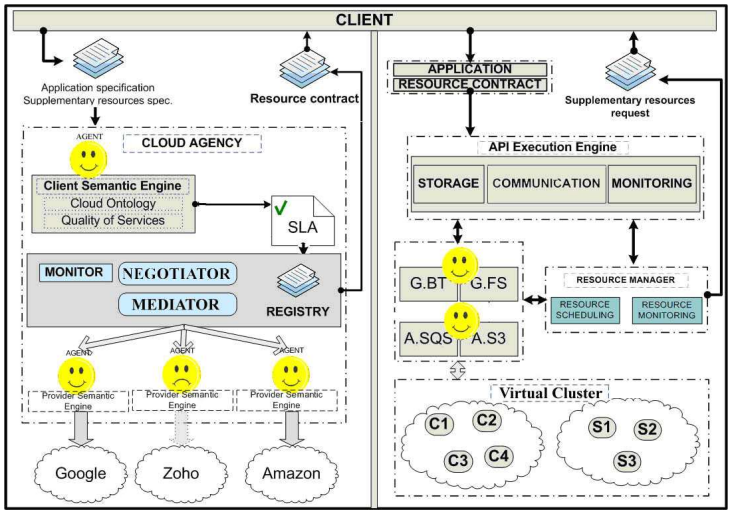


Fig. 3.1: mOSAIC Architecture

it provides facilities both for end-users (at the left of Fig.3.1) and for services developers and managers (depicted on the right side of Fig.3.1)

From the end-users' point of view, the main component is the *Cloud Agency*. This consists in a core set of software agents which implement the basic services of this component. They include:

- negotiation of SLAs;
- deployment of Cloud services;
- discovery and brokering of Cloud services.

In particular, *Client Agent* is responsible for collecting users' application requirements, for creating and updating the SLAs in order to grant always to best QoS. The *Negotiator* manages SLAs and mediates between the user and the broker; it selects protocols for agreements, negotiates SLA creation, and it handles fulfilment and violation. The *Mediator* selects vendor agents able to deploy services with the specified user requirements; it also interfaces with services deployed on different vendors' providers. The *Provider Agent* interacts with virtual or physical resources at provider side. In mOSAIC the Cloud Agency was built upon the MAGDA [2] toolset, which provides all the facilities to design, develop and deploy agent-based services. The *semantic engine* uses information in the Cloud Ontology to implement a semantic-based Cloud services discovery exploiting semantic, syntactic and structural schema matching for searches.

In the Cloud developers and managers perspective, the main components of mOSAIC Architecture are the *API execution engine* and the *Resource Manager*. The first one offers a unique API to use Cloud Services from different vendors when using and developing other services. The API execution engine is able to wrap storage, communication and monitoring features of Cloud platforms. In particular, Virtual Clusters (VC) [11] are used as resource management facility. They are configured by software agents in order to let users to configure required services. A *Resource contract* will grant user's requirements and the Resource Manager will assign physical resources to VC on the basis of the contract.

In this architecture, the bonding element which allows for interoperability and resources description is the *Cloud Ontology*. It is the base for Cloud services and resources description and it contains all information needed to characterize API also from a semantic point of view.

The Cloud Ontology is based on several Cloud taxonomies proposed in literature [1, 38, 19, 7, 20]. It is developed in OWL [24] and OWL-S languages [22]. The benefit of using an ontology language is that it acts as a general method for the conceptual description or modelling of information that is implemented by actual resources [37]. mOSAIC aims at developing ontologies that would offer the main building block to describe services at the three delivery models of Cloud Computing (i.e. IaaS, PaaS, SaaS).

4. mOSAIC Ontology. Ontologies offer the means of explicit representation of the meaning of different terms or concepts, together with their relationships. They are directed to represent semantic information, instead of content. Different languages can be considered for the specification of ontologies, including DAML, OIL, RDF and RDFS, OWL or WSML.

The Web Ontology Language (OWL) is a standard from [24, 4], based on XML, RDF and RDFS. With OWL complex relationships and constraints can be represented in ontologies. With important revisions to the language, OWL 2 became the W3C recommendation in 2009, introducing features to improve scalability in applications. [14]

Different efforts to formalize Semantic Web developments exist. Web Service Modeling Ontology (WSMO) [42] "*provides the conceptual underpinning and a formal language for semantically describing all relevant aspects of Web services in order to facilitate the automatization of discovering, combining and invoking electronic services over the Web*" [33]. WSML was offered as a companion language to WSMO, for representing modelled ontologies by a common terminology for Web Services interactions [10, 33]. The Semantic Web Services Framework (SWSF) offers a similar approach, with its two major components, the Semantic Web Services Language (SWSL) and the Semantic Web Services Ontology (SWSO) [3].

Semantically-enabled services offer the means for intelligent selection of services, with automation of different tasks, including service discovery, mediation, invocation, or composition. Current research efforts are enhancing typical web services technologies in order to provide a semantically-enhanced behaviour in developments like OWL-S [22], WSDL-S and METEOR-S [32, 29], WSML [10], WSMO [33], or SWSF [3].

The top level of the mOSAIC Ontology is shown in Fig.4.1 which reports the main concepts of the mOSAIC ontology. Concepts have been identified analysing standards and proposals from literature. In the following its main concepts will be listed and described. A deeper description of the mOSAIC ontology is in [27], this works describes only the elements which are used for the creation of the modelling profile in MetaMORP(h)OSY.

The **Language** class contains instances of languages used for APIs implementation (for example, Java and

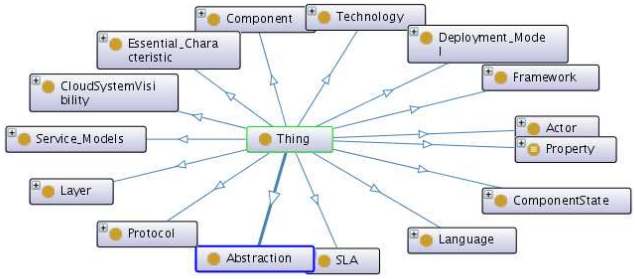


Fig. 4.1: Top Level Concepts in mOSAIC Ontology

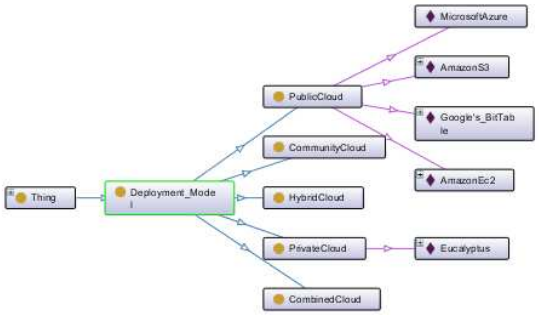


Fig. 4.2: Deployment Model

Python). **Abstraction** class contains the abstraction level at which services are provided as described in [45]. Here, Cloud services belong to the same layer if they have equivalent level of abstraction. **Deployment Model** class includes concepts required by Cloud NIST [45] standard for what deployment model of Cloud services concerns. **Essential Characteristics** class includes individuals which are defined by NIST. **Framework** class contains individuals that identify programming framework supporting API programming Languages. **Actor** contains subclasses where actors interacting with Cloud systems are divided. **Property** subclasses contain all elements needed for describing characteristics of Cloud resources. These are also used to specify SLA requirements. **ComponentState** includes all concepts for defining the states which Cloud components and resources may assume. **SLA** class defines concepts for SLA definitions. **Protocol** class contains individuals for protocols used in communication among Cloud components. **Layers** class distinguishes firmware, hardware and software infrastructures for Cloud platforms. **Service Models** class includes all kinds of services provided by Cloud Systems. **Predicate** contains classes used for description of the behaviours of statefull Cloud components. **CloudSystemVisibility** class allows for specification of Cloud systems visibility, like private and public clouds. **Component** is the main class of mOSAIC ontology. All cloud elements (resources, services, infrastructures etc.) are its subclasses. **Technology** class contains all concepts related to technology involved in Cloud services provisioning, like virtualization.

Fig.4.2 shows the *Deployment_Model* subclasses.

They include several types of deployment models for Cloud Systems: PublicCloud contains all individuals providing public or world wide access to their resources, like MicrosoftAzure, Amazon and Google. PrivateCloud instead is related to Deployment Models of framework that can provide access to private Cloud resources, like Eucaliptus.

The Actor class identifies cloud actors, that can be divided as in Fig.4.3.

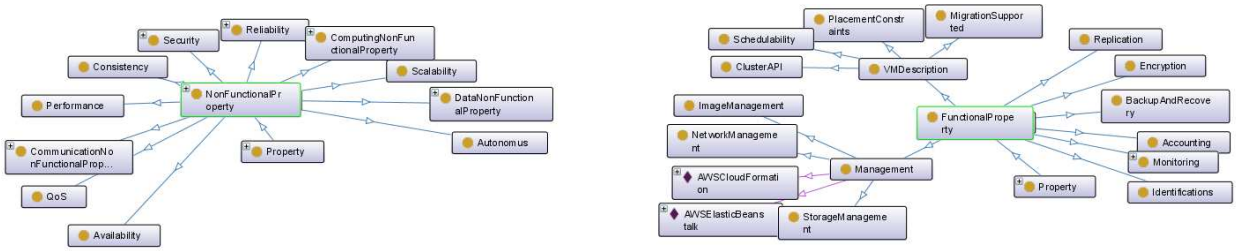
Provider, Consumer and Creator subclasses follow the IBM cloud computing reference Architecture [23]. Administrator manages cloud infrastructure; Orchestrator composes Cloud Services in order to provide value

(a) Actor (b) Consumer

Fig. 4.3: Actors

added services; Developer implements new Cloud Services. Notice that the difference between Developer and Creator, is in the way they interact with cloud Providers. Developers use offline resources (tools and frameworks) in order to implement new Cloud Service. A Creator instead builds cloud services by using functionalities exposed by a Cloud Service Provider. Consumer Actors can be further divided as shown in Fig. 4.3(b).

Some Property's subclasses are shown in Fig.4.4. They are divided into NonFunctionalProperties and FunctionalProperties that respectively define the sets of non functional and functional properties of a Cloud Component. Properties can be used to characterize Cloud Components (services, infrastructure etc.) and to request given characteristics for components when dealing with SLA.



(a) Non Functional Properties (b) Functional Properties

Fig. 4.4: Properties

The main non-functional properties for cloud components are: Scalability; Autonomy; Availability; QoS; Performance; Consistency; Security; Reliability.

Computing Non Functional properties can be divided into CPU and Memory related properties. A deeper division identifies: CPUSpeedProperty; CPUNumberOfCores; CPUArchitecture; CPUTypeProperty and CPUFlopsProperty. These properties are used to specify the clock frequency, the number of cores, the architecture, the model and the FLOPS of CPUs respectively. The properties follow the OCCI [40] standard and API. A Data Property is defined for each of them in order to specify the value of the property for the related individuals. Properties for memory are divided into: MemoryAllocationProperty and MemorySize. The first property is used to specify memory allocation policies while the second one is used to declare (or require) the amount of memory in a Cloud infrastructure.

Subclasses of this Network Non Functional element are: NetworkLatencyProperty, NetworkDelayProperty, NetworkBandwidthProperty. The first class is used to define the mean latency of a network, the second one the mean, the maximum and the minimum delay for packets and the last one is used to define the mean and the maximum bandwidth of a network. The values for individuals are defined by specifying proper data properties

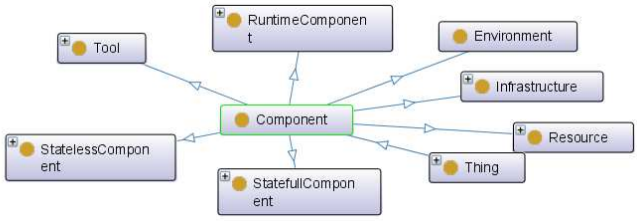


Fig. 4.5: Component

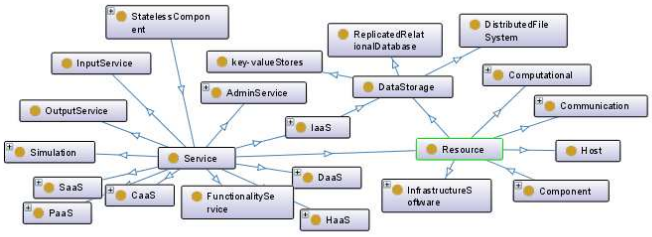


Fig. 4.6: Resource and Services

defined on these classes. Data non functional properties are related to disk size (DiskSpaceProperty), transfer rate (DiskTransferRateProperty) and bandwidth (DiskBandwidthProperty).

The main functional properties are: Replication (for the definition of the type of replication policies of resources); Encryption (it specifies the encryption policies of resources); BackupAndRecovery (it is used to describe the back up and recovery strategies used for a Cloud Component); Accounting (its individuals define the accounting policies for resources); Monitoring (this class allows for the specification of monitoring policies for resources); Identification (it contains individuals that can specify the algorithms and policies for users identification); VMDescription (used to describe virtual machines technologies and configuration eventually used in cloud infrastructure); Management (it defines the management policies for cloud resources).

Management contains the following subclasses: ImageManagement, NetworkManagement and StorageManagement. The first one is used to define the management policies of a VM image, the second one to define network management policies in a cloud infrastructure, while the third one is used to define storage management policies for cloud resources.

Service_Models subclasses includes all models for services in Cloud. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Service as a Service (SaaS) are the classical models defined in the NIST standard, while the last, BPaaS (Business Process as a Service) is defined in the IBM Cloud Computing Reference Architecture.

Component Subclasses are reported in Fig.4.5.

They are divided into: Tool (this contains all tools used for Cloud services development or cloud resources management); RunTimeComponent (this class contains the elements for defining mOSAIC run-time components); Environment (used to define individuals concerning the cloud environment used by cloud services); Infrastructure (describes the component in the cloud infrastructure); StatefullComponent and StatelessComponent; Resource (it collects all resource classes in the cloud ontology).

The Resource class is the most complex in mOSAIC, since, following the OCCI documentation, in Cloud Systems everything is a cloud Resource. Hence this is a super class for other main cloud components as shown in Fig.4.6.

Service is a Resource. Platform as a Service (PaaS), Computing as a Service (CaaS), Data as a Service (DaaS), infrastructure as a Service (IaaS), Hardware as a Service (HaaS) and other service models (Simulation,

services which offers some functionalities, admin, data input and output services) are subclasses of Services. For example, in Figure, DataStorage is provided as an IaaS. Key-valueStores, ReplicatedRelationalDatabeses and DistributedFileSystems are examples of DataStorage services. Cloud Component are also considered as Cloud Resources, like Hosts, Computational and Communication resources or InfrastructureSoftware.

5. MetaMORP(h)OSY modelling methodology. The *MetaMORP(h)OSY* modelling methodology (MMM) extends and improves the one used in the REMM [28] framework. In REMM, systems are modelled as Multi Agents, by using Beliefs, Desires, Intentions (BDI) logics [44]. REMM models are based on a UML modelling profile that implements a modelling language (RT-AML) for real-time BDI MAS. Requirements are verified at design phase by using formal methods following a MDE approach. REMM models are translated into timed automata in order to check properties expressed in timed temporal logics. At run-time, REMM provides a mean for checking properties verified at design-time. This is achieved by collecting run-time measures that are used to tune again design models. They are then used to verify previously checked properties in the case temporal behaviour of real systems differs from the designed one. REMM framework also provides translation from design models to Jadex [31] run-time agents with real-time scheduling features.

The REMM methodology has some limitation. First of all, requirements can be expressed only in timed temporal logics and the only way to analyse them is to translate models into timed automata. At design phase, no way is provided for choosing the type of analysis required on the model. In addition recently OMG has defined a UML profile for modelling real-time systems (MARTE [12]). Since the methodology on which REMM (and *MetaMORP(h)OSY*) is based on UML models, the use of a profile which also supports MARTE standard is appealing. Finally, REMM does not provide any translations features for producing simulation models. Simulation gives the ability of performing fault injection and other fault analyses on the system to implement, and this is useful especially for critical systems.

The MMM overcomes these problems, extending and improving the modelling language, the profile used in REMM, and the techniques used to translate modelling and simulation models.

The main phases of the MMM are depicted in Fig.5.1.

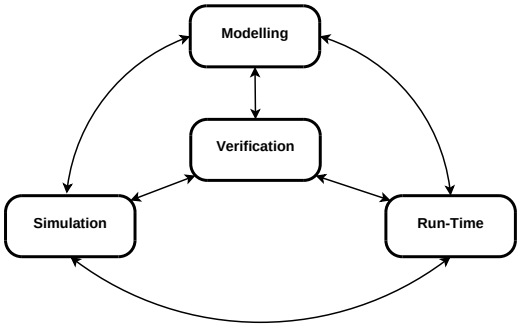


Fig. 5.1: *MetaMORP(h)OSY* methodology

They consist in: *Modelling*, *Simulation*, *Run-time* and *Verification*. They will be described briefly in the following.

5.1. Modelling phase. In this phase design models for the system to study and realize, and requirements to verify on the system are defined. Models are described by using the Real-Time Agent Modelling Language (RT-AML) which extends the one presented in [28]. RT-AML is based on UML diagrams, and extensions of Class diagrams, Activity diagrams and Sequence diagrams are used to describe the BDI model of agents. Models developed in this phase can be used to define run-time and simulation components. According to MDE philosophy, requirements have to be verified on models even at this stage. Proper translation techniques are defined to build models used in verification phase. Models generated from verification depend on requirements to verify and on the analysis to perform.

5.2. Simulation phase. In this phase Simulation models are defined or generated automatically from models defined in modelling phase. In the last case, stubs are generated from agent diagrams and real-time schedulers and monitors are provided in order to verify requirements. The simulation models used in MMM are

based on MASON [21] toolkit and can be augmented with fault-injection by using proper libraries. Requirements verification is instead performed at verification stage, by using formal models which have been eventually generated during the modelling phase.

Agents behaviours during simulation may differ from the ones defined in the modelling phase. This happens when faults are injected in the simulation, or if conditions that may change temporal behaviours of the agents are considered in simulation. In these cases, requirements defined in simulation phase can be verified on the same models used for verification in modelling phase. The difference is in the parameters used to tune the verification models, which are collected from simulated behaviour.

5.3. Run-Time phase. In this phase the real system is developed and executed. Stubs for run-time components can be generated from modelling phase as for simulation. Also verification models can be used in real-time scheduling in order to forecast if real-time constraints and other requirements are verified at run-time, in the same way of simulation. In addition, code from simulation can be used to implement the system, and also simulation models can be used at run-time for verification purposes.

5.4. Verification. Verification phase is driven by formal models. Models from modelling phase are translated into models which are useful for verifying system requirements. The models generated for verification depends on the properties and on the analyses to perform.

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

In the *Agent Diagram* the MAS system structure is described. Here classes with proper stereotype represent Agents with their Plans and Beliefs.

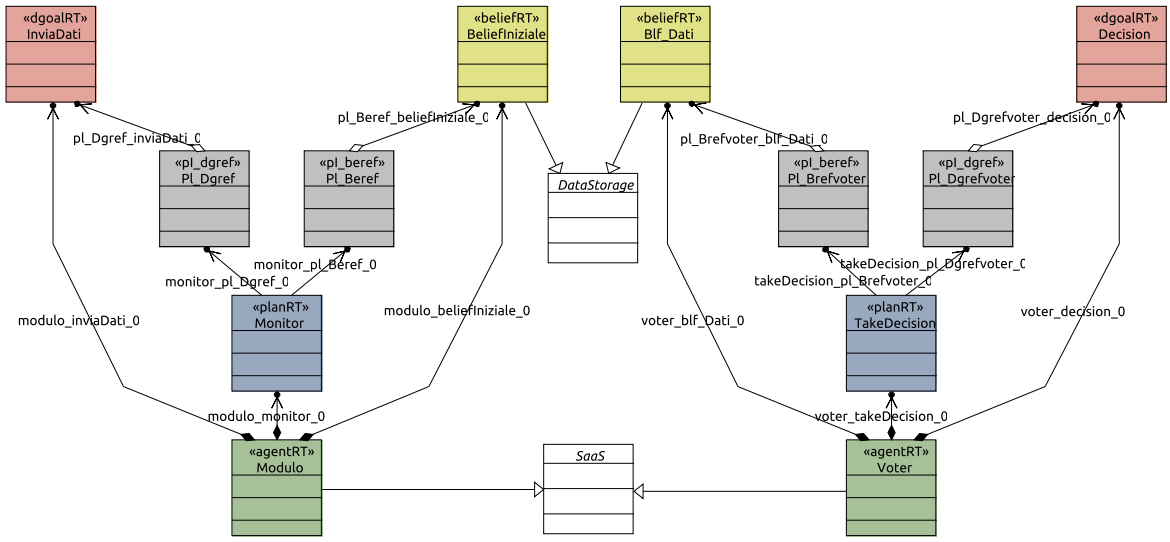


Fig. 5.2: Agent Diagram

For example, diagram in Fig.5.2 depicts two agents. The stereotype *AgentRT* is applied to these classes. It is defined in the MetaMORP(h)OSY modelling 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. *PlanRT*, *BeliefRT* and *DGoalRT* stereotypes are defined in the MetaMORP(h)OSY modelling profile which in turn are used to define plans, beliefs and goals of agents. Plans are related to some beliefs and pursue some goals. Relationships between plans, beliefs and goals are implemented by mean of other stereotypes which description is omitted for brevity.

The Agent Diagram in Fig.5.2 describes agents structures, listing their plans, beliefs and goals related to each plan. In order to complete the model a dynamic description of agents behaviours has to be provided. In MetaMORP(h)OSY this is done by mean of particular activity and sequence diagrams.

MetaMORP(h)OSY profile defines several stereotypes for messages and timed activities in the activity diagrams. This allows for the analysis of timed behaviour and interaction of agents. At the state, each PlanRT in the Agent Diagram 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.

For example, in Fig.5.3 the Activity diagram for the *TakeDecision* plan is shown.

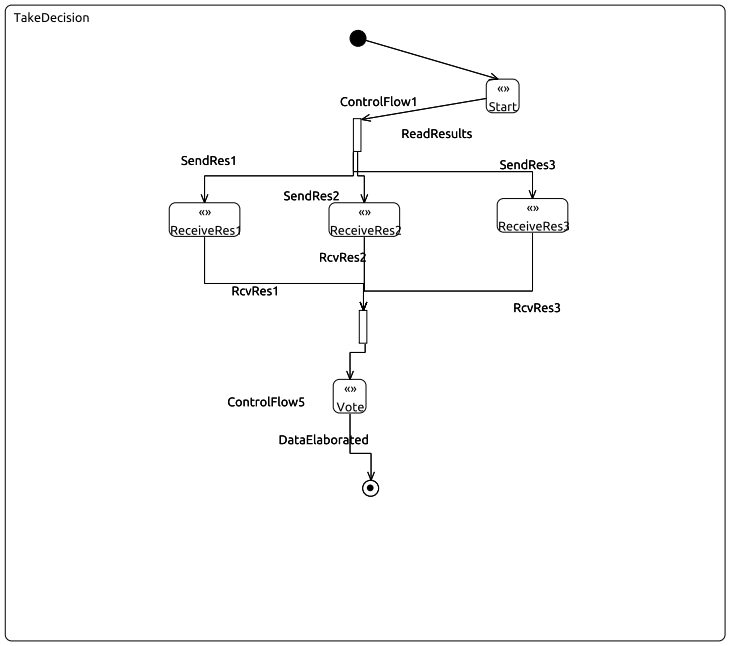


Fig. 5.3: TakeDecision Plan

Activity diagrams describe agents behaviours 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 used in MetaMORP(h)OSY. The main purpose of this diagram is the definition of agents interactions in a use case.

All messages in the sequence are *StimulusRT*: a stereotype able to represent the timed behaviour of messages exchanged during the execution of a use case. *StimulusRT* messages are related to messages sent and received in activity diagrams.

Fig.5.4 depicts a Sequence Diagram.

6. MetaMORP(h)OSY modelling profile. MetaMORP(h)OSY modelling methodology is based on the creation of a UML modelling profile, that can be considered as a meta-language for definition of components in UML models.

The basic MMM Modelling profile extends the MARTE [12] profile. It defines the stereotypes and the properties needed to define a MAS system. Fig.6.1 shows the profile used for Agent Diagrams definition.

The Profile extends the basic UML and MARTE profiles. This means that classical UML elements and stereotypes from MARTE meta-language can be used in a model compliant with the profile. In addition, it introduces the elements in Tab.6.1

Agent Plans are modelled by using an extension of the UML Activity diagram. Fig.6.2 shows the extended profile. It extends the basic UML activity diagrams with the elements described in Tab.6.2

Activity diagrams defined with the previous elements take into account of interaction among agents. In order to cope with complexity, it is possible to define the sequence of messages that agents exchange during the execution of a particular use case. Fig.6.3 depicts the elements in the MetaMORP(h)OSY profile used for sequence diagrams definition.

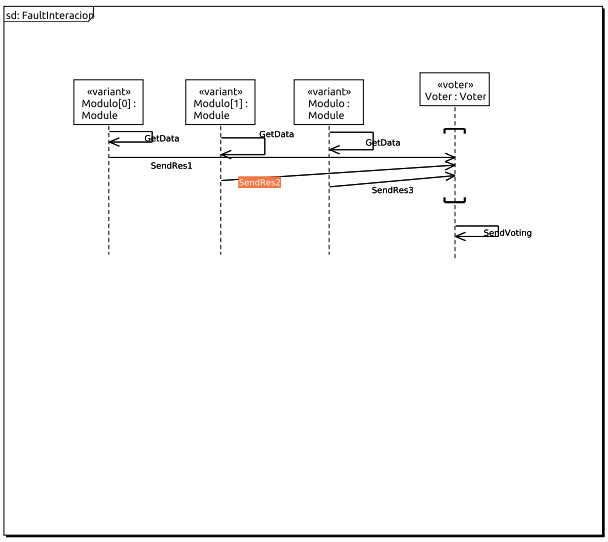


Fig. 5.4: Sequence Diagram

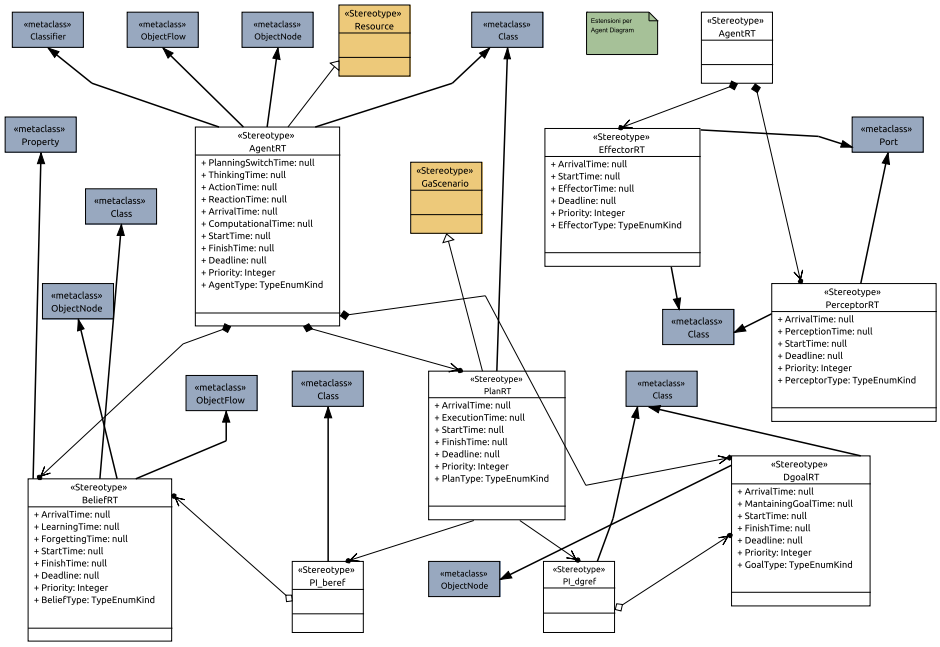


Fig. 6.1: Agent Diagram Profile

Again the profile extends the basic UML sequence diagrams and allows for the definition of the elements listed in Tab.6.3.

The basic modelling profile of the MMM is used in order to describe MAS structure and behaviours, but it lacks of detailed information about modelling domain. This information can be retrieved from the mOSAIC ontology. The ontology is translated into a set of classes that are inherited during Agent Diagram modelling phase. The translation first identifies equivalent classes in the ontology. The ontology taxonomy is then translated into a hierarchy of UML classes and then relationships among classes are translated into associations.

Part of the hierarchy produced from the piece of the ontology in Fig.4.6 is depicted in Fig.6.4.

Table 6.1: Agent Diagram Elements

Component	Description
AgentRT	Used for definition of agents with temporal description
PlanRT	Used for the definition of agent plans. Each plan is related to an Activity diagram that reports the behaviour of the agent while following the plan in terms of Action States
DGoalRT	Used for the definition of agents goals. Agents pursue goals while executing plans
PerceptorRT and EffectorRT	These are commonly used to define inputs and outputs for agents
BeliefRT	Beliefs are used to store status for agents containing their beliefs about the external environment and other agents

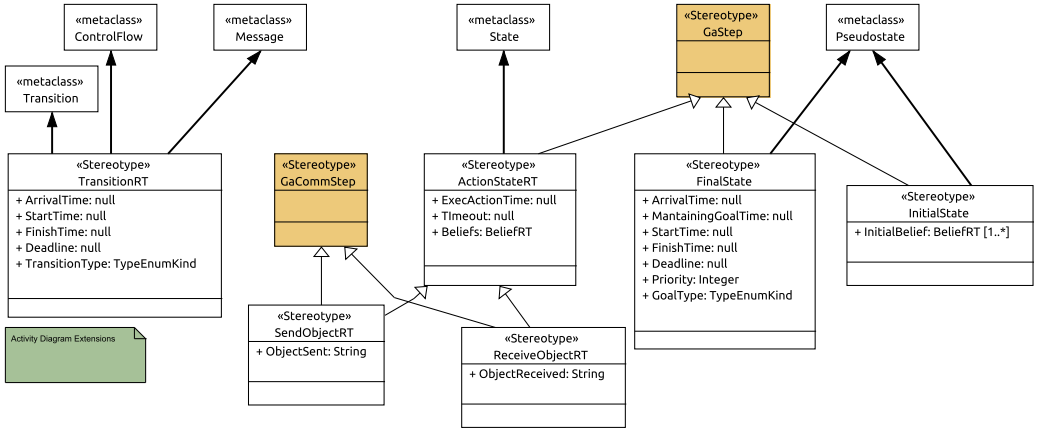


Fig. 6.2: Activity Diagram Profile

Following the MARTE specification, properties (and requirements) on a model can be analysed or monitored by elements called *Observer* [12].

Verification in MetaMORP(h)OSY is enacted by means of model transformation and model analysis. For example, state reachability of goals under real-time constraints is performed by translating the RT-AML model into Timed Automata and then executing a model checker on the translated model([28]).

In order to specify on which elements Observers work, it is necessary to identify the elements on which properties can be evaluated. For example, (timed) state reachability analysis is available on DGoalRT elements, that are associated to final states of activities.

When dealing with complex SLA, the generation of the profile containing properties definition is appealing. The mOSAIC ontology is used in order to generate properties modelling profile.

Thanks to the hierarchical organization of elements in the ontology, subclasses of *Property* concept (see Fig.4.4) are identified and inserted in the modelling profile as shown in Fig. 6.5

The ontology also contains relationships containing information about the classes on which properties can be defined. For example, Availability can be requested on storage resources etc. This information is used in order to complete the properties profile and to define proper Observers.

7. Example. In this section an example of the use of MetaMORP(h)OSY for definition of a cloud service with a requested SLA. It will be shown how MMM is used in order to validate the SLA on the model *before* user and provider agree on the offered service.

In the example the user requests a service with high availability. The service vendor is able to provide a service with triple redundancy and voting and it want to assure that its composed service has the level of reliability required by the user. The service (called here simply *Module*) is replicated thrice and each service works stand-alone producing its own results. Results are than collected by a service called *Voter* that forward results only if two on three are equal.

The Agent Diagram of the system is reported in Fig.5.2 where the two agents representing the *Module* and *Voter* services are depicted. The Module agent has the goal of sending computed results to the Voter (*SendData*)

Table 6.2: Activity Diagram Elements

Component	Description
ActionStateRT	Actions in agents plan are modelled with this component that allows for specification of temporal behaviour of actions
TransitionRT	Transitions among ActionStateRT. For these elements it is possible to specify events synchronizations and deadlines
InitialState FinalState	Initial and Final states of the plan usually final state can be associated to a DGoalRT

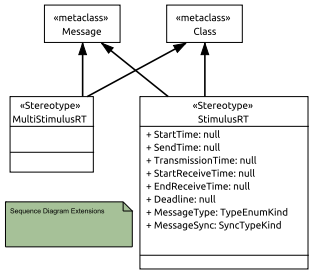


Fig. 6.3: Sequence Diagram Profile

and the Voter agent has the goal of taking a decision (*Decision*).

The *SendData* goal is achieved by executing the *Monitor* plan, while *Decision* by executing the *TakeDecision* plan. Both Module and Voter are provided as *SaaS*. And the beliefs of the agents are stored in *DataStorage*. For each Agent properties defined in the MetaMORP(h)OSY profile are used. In particular, in this example, each Agent has its own Fault probability and its own Mean Time to Failure that will be used in order to evaluate global reliability on the system.

The *Monitor* plan is simple and it is not described for brevity.

The *TakeDecision* plan is shown in Fig.5.3, where the voter requests results from the three modules ($Receive\_Res(i)$) and *vote*s when messages arrive.

The Sequence Diagram in Fig.5.4 is used to define that three different *Module* agents send their messages to enact voting. Voting agent collects messages requiring that at least 2 messages correctly arrive. StimulusRT stereotype is used for messages and information about messages reliability is reported in their modelling properties.

Finally, an Observer for the evaluation of the global reliability of the modelled system is defined. This is depicted in Fig.7.1

The Observer translates the RT-AML model into a Fault Tree Model and generates the input for the Sharpe [35] framework in order to evaluate availability, but the description of the analysis model is out of the scope of this work. Service providers can change model parameters in order to establish the correct configuration of services for assuring the requested QoS.

8. Conclusions. In this paper the mOSAIC Ontology, the MetaMORP(h)OSY methodology and framework have been introduced. It has been shown how domain-related information contained in the ontology can be used in order to enhance a modelling profile for formal verification of QoS of 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 analysis of high available and fault tolerant scenarios in the mOSAIC project.

Acknowledgments. This work has been supported by the mOSAIC project (EU FP7-ICT programme, project under grant #256910).

REFERENCES

Table 6.3: Sequence Diagram Elements

Component	Description
StimulusRT	This elements allows for the specification of asynchronous and synchronous messages, where deadline, arrival time and other temporal properties can be specified
MultiStimulusRT	It is a StimulusRT for redundant messages

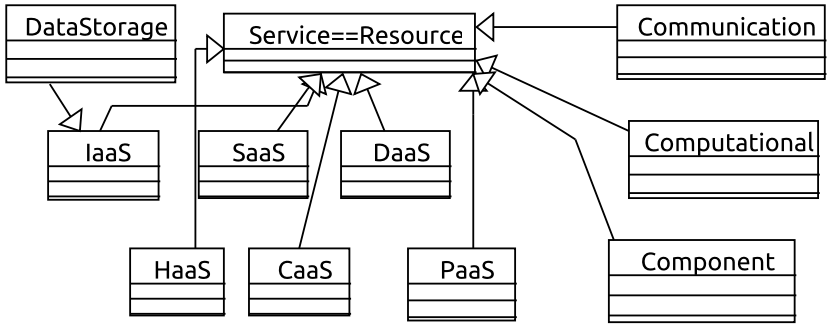


Fig. 6.4: Classes imported from mOSAIC ontology

[1] Appistry, *Cloud Taxonomy: Applications, Platform, Infrastructure : http://www.appistry.com/blogs/sam/cloud-taxonomy-applications-platform-infrastructure*, 2008.

[2] R. Aversa, B. Di Martino, N. Mazzocca, and S. Venticinque, *A skeleton based programming paradigm for mobile multi-agents on distributed systems and its realization within the magda mobile agents platform*, Mob. Inf. Syst., 4 (2008), pp. 131–146.

[3] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, J. Su, and S. Tabet, *Semantic web services framework*, September 2005.

[4] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. Mcguinness, P. F. Patel-Schneider, and L. A. Stein, *Owl web ontology language reference*, tech. rep., W3C, 2004.

[5] R. Buyya, C. S. Yeo, and S. Venugopal, *Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities*, in HPCC '08: Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications, IEEE Computer Society, September 2008, pp. 5–13.

[6] B. Chen, H. H. Cheng, and J. Palen, *Integrating mobile agent technology with multi-agent systems for distributed traffic detection and management systems*, Transportation Research Part C: Emerging Technologies, 17 (2009), pp. 1 – 10.

[7] C.Hoff, *Cloud Taxonomy and Ontology : http://rationalsecurity.typepad.com/ blog/2009/01/cloud-computing-taxonomy-ontology.html*, 2009.

[8] M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci, *Introducing pattern reuse in the design of multi-agent systems*, in Agent Technologies, Infrastructures, Tools, and Applications for E-Services, vol. 2592 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2003, pp. 107–120.

[9] V. T. Da Silva and 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), pp. 145–189.

[10] J. De Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel, *The web service modeling language wsml*, tech. rep., DERI, October 2005.

[11] U. V. E.P. Mancini, M. Rak, *PerfCloud: GRID Services for Performance-oriented Development of Cloud Computing Applications*, in Proceedings of WETICE, IEEE Computer Society, July 2009.

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

[13] Galen Gruman and Eric Knorr, *What cloud computing really means. InfoWorld : http://www.infoworld.com/article/08/04/07/15FE-cloud-computing-reality 1.html*, 2008.

[14] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler, *Owl 2: The next step for owl*, Web Semant., 6 (2008), pp. 309–322.

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

[16] K. Hwang, *Massively distributed systems: From grids and p2p to clouds*, in Proceedings of The 3rd International Conference on Grid and Pervasive Computing - gpc-workshops, 2008, p. xxii.

[17] Jeremy Geelan, *Twenty one experts define cloud computing. Virtualization : http://virtualization.sys-con.com/node/612375*, 2008.

[18] K. M. Kavi, M. Aborizka, D. Kung, and 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.

[19] P. Lairds, *Cloud Computing Taxonomy*, in Procs. Interop09, IEEE Computer Society, May 2009, pp. 201–206.

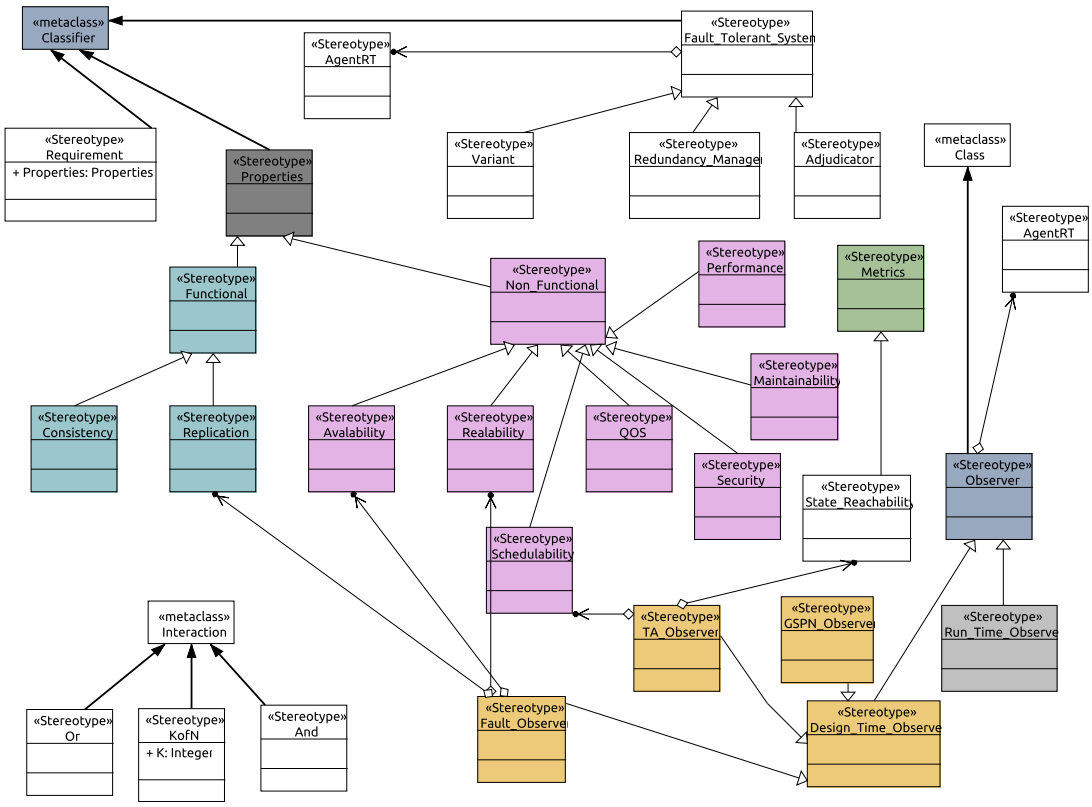


Fig. 6.5: Properties Diagram Profile

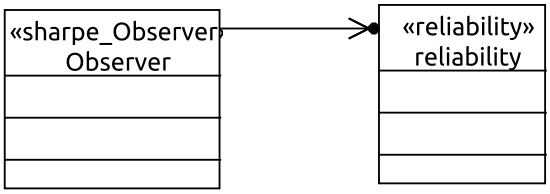


Fig. 7.1: Reliability Observer

[20] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, *What's inside the cloud? an architectural map of the cloud landscape*, in Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, CLOUD '09, Washington, DC, USA, 2009, IEEE Computer Society, pp. 23–31.

[21] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. C. Balan, *Mason: A multi-agent simulation environment*, Simulation, 81 (2005), pp. 517–527.

[22] D. Martin, M. Paolucci, S. Mcilraith, M. Burstein, D. Mcdermott, D. Mcguinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara, *Bringing Semantics to Web Services: The OWL-S Approach*, in SWSWPC 2004, J. Cardoso and A. Sheth, eds., vol. 3387 of LNCS, Springer, 2004, pp. 26–42.

[23] M.Behrendt, B.Glasner, P.Kopp, R.Dieckmann, G.Breiter, S.Pappe, H.Kreger, and A. Arsanjani, *Introduction and Architecture Overview, IBM Cloud Computing Reference Architecture 2.0: https://www.opengroup.org/cloudcomputing/uploads/40/23840/CCRA.IBMSubmission.02282011.doc*, 2011.

[24] McGuinness, D.L., van Harmelen, F., *OWL Web Ontology Language Overview. W3C Recommendation: http://www.w3.org/TR/2004/REC-owl-features-20040210/*, 2004.

[25] Members of EGEE-II, *An egee comparative study: Grids and clouds - evolution or revolution. Technical report, Enabling Grids for E-sciencE Project : https://edms.cern.ch/document/925013/*, 2008.

[26] D. Milojicic, *Cloud computing: Interview with russ daniels and franco travostino*, IEEE Internet Computing, (2008), pp. 7–9.

[27] F. Moscato, R. Aversa, B. Di Martino, T. Fortis, and V. Munteanu, *An analysis of mosaic ontology for cloud resources annotation*, in Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on, IEEE,

2011, pp. 973–980.
[28] F. Moscato, S. Venticinque, R. Aversa, and B. Di Martino, *Formal modeling and verification of real-time multi-agent systems: The remm framework*, in Intelligent Distributed Computing, Systems and Applications, C. Badica, G. Mangioni, V. Carchiolo, and D. Burdescu, eds., vol. 162 of Studies in Computational Intelligence, Springer Berlin / Heidelberg, 2008, pp. 187–196.
[29] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma, *Meteor-s web service annotation framework*, in Proceedings of the 13th international conference on World Wide Web, WWW '04, New York, NY, USA, 2004, ACM, pp. 553–562.
[30] Paul McFedries, *The cloud is the computer. IEEE Spectrum Online, : http://www.spectrum.ieee.org/aug08/6490*, 2008.
[31] A. Pokahr, L. Braubach, and W. Lamersdorf, *Jadex: A bdi reasoning engine*, in Multi-Agent Programming, J. D. R. Bordini, M. Dastani and A. E. F. Seghrouchni, eds., Springer Science+Business Media Inc., USA, 9 2005, pp. 149–174. Book chapter.
[32] R. Akkiraju, J. Farrell, J.Miller, M. Nagarajan, M. Schmidt, A. Sheth, K. Verma, *Web Service Semantics WSDL-S. A joint UGA-IBM Technical Note, version 1.0: http://lsdis.cs.uga.edu/projects/METEOR-S/WSDL*, 2005.
[33] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, *Wsmo - web service modeling ontology*, in DERI Working Draft 14, vol. 1, BG Amsterdam, 2005, Digital Enterprise Research Institute (DERI), IOS Press, pp. 77–106.
[34] Roy Bragg, *Cloud computing: When computers really rule: http://www.technewsworld.com/story/63954.html*, 2008.
[35] K. S. Trivedi and R. Sahner, *Sharpe at the age of twenty two*, SIGMETRICS Perform. Eval. Rev., 36 (2009), pp. 52–57.
[36] S. Venticinque, R. Aversa, B. D. Martino, and D. Petcu, *Agent based cloud provisioning and management: design and protoypal implementation*, in Proc. of Cloud Computing and Services Science (CLOSER), SciTePress, 2011, pp. 184–191.
[37] VV.AA., *Cloud Computing Interoperability Forum, Unified Cloud Computing: http://code.google.com/p/unifiedcloud/.*
[38] ———, *Cloud Computing Interoperability Forum,Cloud taxonomy : http://groups.google.com/group/cloudforum/web/ccif-cloud-taxonomy.*
[39] ———, *Open Cloud Manifesto, Spring 2009 : http://www.opencloudmanifesto.org.*
[40] ———, *Open Grid Forum: Open Cloud Computing Interface (OCCI): http://forge.ogf.org/sf/projects/occi-wg.*
[41] ———, *Papyrus uml: http://www.papyrusuml.org.*
[42] ———, *Web Service Modelling Ontology (WSMO): http://www.wsmo.org.*
[43] A. Weiss, *Computing in the clouds*, netWorker, 11 (2007), pp. 16–25.
[44] M. Wooldridge, *Agent-based software engineering*, in IEE Proceedings on Software Engineering, 1997, pp. 26–37.
[45] L. Youseff, M. Butrico, and D. D. Silva, *Towards a unified ontology of cloud computing*, in Grid Computing Environments Workshop, 2008. GCE '08, Nov 2008, pp. 1–10.
[46] Q. Zhang, L. Cheng, and R. Boutaba, *Cloud computing: state-of-the-art and research challenges*, J. Internet Serv Appl, (2010), pp. 7–18.
[47] Y. Zhang, E. Manisterski, S. Kraus, V. Subrahmanian, and D. Peleg, *Computing the fault tolerance of multi-agent deployment*, Artificial Intelligence, 173 (2009), pp. 437 – 465.

Edited by: Dana Petcu and Daniela Zaharie
Received: March 1, 2012
Accepted: April 15, 2012