QDSL - QUALITY DOMAIN SPECIFIC LANGUAGE FOR CLOUD COMPOSITE APPLICATIONS

Short Research Paper

Ethan Hadar[1], Irit Hadar[2] and Donald F. Ferguson[3]
[1]*CA Technologies, Inc., Herzelia, Israel*
[2]*Department of Management Information System, University of Haifa, Haifa, Israel*
[3]*CA Technologies, Inc., New York, NY, U.S.A.*

Keywords: Domain Specific Language, Quality Assessment, Service Level Agreements, Cloud Service Quality, Cloud Modeling.

Abstract: Quality Domain Specific Language (QDSL) is a model-driven approach providing a taxonomy, model, and visual editing tool for evaluating and benchmarking the quality of composite applications in cloud environments. Our language and associated modeling tool provide visual and textual means for constructing mathematical algorithms needed for computing aggregated quality assessment of cloud services. QDSL enables the illustration and definition of metrics, measurements and indicators, relationships for computation, and transformation functions that normalize the measurements into relative quality scoring. As a result, QDSL provides a structure that guides overall quality assessments. The computation algorithm is structured in a visual manner and associates the quality assessments graph with the structure of the cloud composite application in a hybrid environment. QDSL supports transformation from physical measurements into scoring comparative assessments of benchmarked provided IT solutions. This paper presents a basic model for QDSL and examples of usage. A prototypical eclipse EMF modeling tool of QDSL is used for communication, whereas commercial monitoring tools implement the instantiated models for evaluating service qualities.

1INTRODUCTION

In the cloud domain, composite IT services and composite applications may be implemented across hybrid cloud environments (Ferguson and Hadar, 2010; 2011) with interchangeable alternatives, differing in their quality levels (Adam and Doerr, 2007). The increasing number of offerings from different vendors for the same conceptual cloud service generates an economic attraction. Specialization in these services presumably increases the quality of the services (Donzelli and Bresciani, 2004). Alternative services should be evaluated systematically and concisely. Such semantic definitions (Frank et al., 2009; Gruber, 1995; Franch and Carvallo, 2003) and computation methods that can be understood by all stakeholders (Kupfer and Hadar, 2008) are captured in this paper as QDSL: Quality Domain Specific Language.

The need to provide a service as quickly as possible drives service agility. Risk is associated with evaluating the ability to perform a task, and accordingly, all the associated Costs. Agility, Risk and Cost are considerations for selecting a cloud provider and service instead of constructing solutions on-premise. The IT team evaluates service alternatives, and selects the best cloud service according to quality requirements. There are several challenges in evaluating cloud services involving different aspects (Franch and Carvallo, 2003). The technical aspects require the ability to: (1) collect metrics from the vendors' cloud services; (2) convert these metrics into a set of scores that enable rating; (3) define methods of aggregation and the relative contributions of each measurement, compound, or derived metric; and, (4) compare these scores with other similar services according to an agreed standard, such as the Service Measurement Index (SMI) (Zachos, 2011). Social aspects include the ability to: (1) add social indicators of the perceived quality of a cloud service or vendor as surveyed by consumers; (2) increase the trust level of people

Hadar E., Hadar I. and F. Ferguson D..
QDSL - QUALITY DOMAIN SPECIFIC LANGUAGE FOR CLOUD COMPOSITE APPLICATIONS - Short Research Paper.
DOI: 10.5220/0003913102280233
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 228-233
ISBN: 978-989-8565-05-1

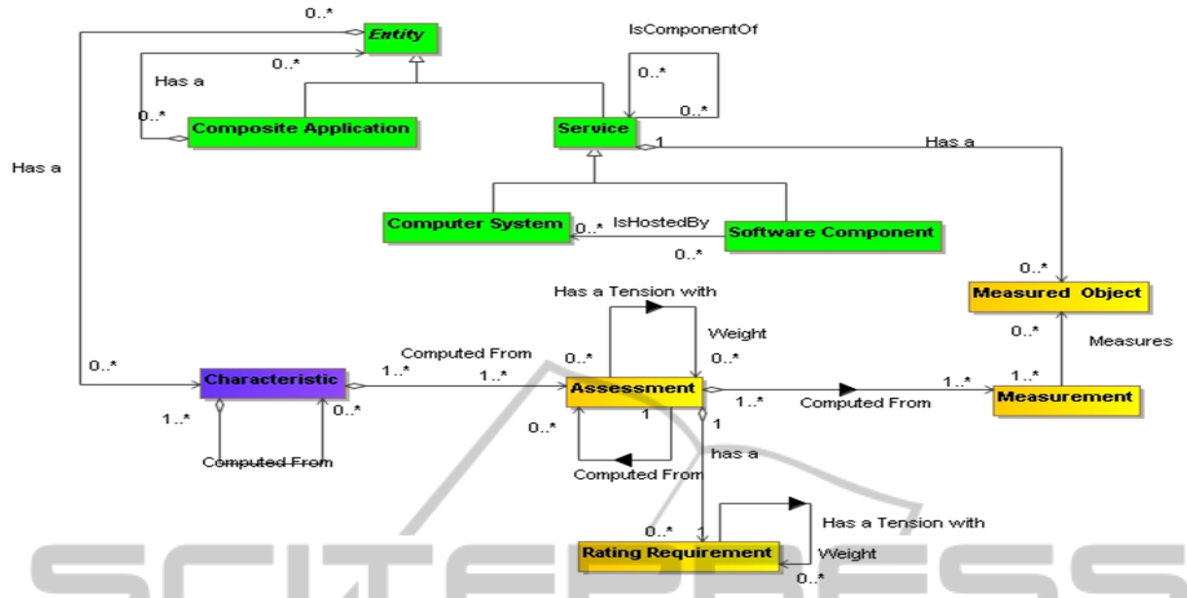

Figure 1: Simplified conceptual model of QDSL.

evaluating the services according to the integrity and fidelity of the data presented; (3) control the level of importance of each metric and the overall contribution according to the individual considerations of the evaluator.

Benchmarking cloud services is based on comparing the scores of the functional and non-functional characteristics of a service. Such characteristics can be Quality, Agility, Risk, Capability, Cost, and Security.

QDSL is a modeling language for describing the computation structures for evaluating the characteristics of cloud services and composite applications. These evaluation ratings are computed based on certain objective (metrics) and subjective (indicators) measurements. The measurements are extracted from monitored cloud services or their composite IT system and application structures. The measurements' assessments define the score of the service or its ingredient characteristics, and aggregate them according to the preferences of the evaluator (consumer). Finally, the aggregated characteristic of a service is evaluated against that of other similar services. QDSL characteristics prioritization is applied to measurements as part of the mathematical score computation process, as well as the importance of different characteristics. QDSL enables one to define an algorithm for computing characteristics. It is used when combining measurements (monitored or derived metrics and indicators) and weighting (prioritization) factors according to the evaluator's specific aggregation considerations.

Using QDSL's graphical editor enables the modeler to capture, share, and reuse characteristics structures as presented in Figure 3. These models can be rapidly instantiated into run-time models and machine-readable formats, executed on computation engines that aggregate the metrics and produce characteristics dashboards.

In the next sections a partial view of a conceptual model is described to illustrate the QDSL. Several examples are presented, and a brief description of a prototypical tool is provided.

2 THE QDSL CONCEPTUAL MODEL

The QDSL domain encompasses conceptual binding cloud composite applications, measurements and quality characteristics. A simplified view of QDSL specifications is provided in Figure 1, while subsequent sections present selected entities and relationships. In QDSL, measurements are monitored via a brokering service, using connectors defined as *Measured Objects*. The monitored *Composite Application* entities have associated functional and/or non-functional characteristics.

The QDSL universe encompasses three main domains:

1. Composite Application Domain, which comprises the entities under evaluation: Service, Software, Computer System, and Composite Application.

2. Characteristics Domain, which defines the criteria for evaluation, such as functional and non-functional properties.
3. Measures Domain, which provides the supporting data: Assessments, Rating Requirements, Measurements, and Measured Objects.

2.1 The Composite Application Domain

The Composite Application Domain (Figure 2) supports the constructing of logical software components (including libraries, operation systems, and applications) and services, and assigns them to computing applications. Each characteristic should be associated to each of these entities (Service, Software, Composite Application, or Computer System). A *Measured Object* (from the Measures Domain) retrieves data from a measured entity associated with a certain capability, such as the CPU utilization, or fixed cost of a service per month.

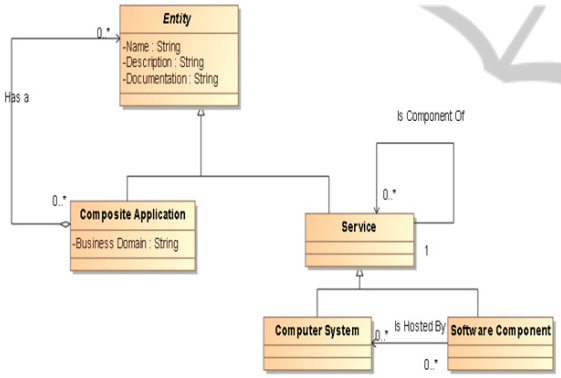

Figure 2: The Composite Application Domain.

Figure 3 depicts a QDSL model of a shopping cart e-commerce composite application. The Composite application sub-services (catalog, ordering, and billing) and their associated costs are modeled. In this example, the billing service is provided by the billing engine software that runs on Amazon EC2 infrastructure. The EC2 has a relative hardware cost score that is derived from the Amazon service. This Amazon service has a price statement Measured Object, populating information into the pay-per-server Measurement. This measurement is transformed into the Assessment Score using the pay-per-server Range Requirement. Accordingly, the hardware cost of the Amazon infrastructure is calculated, and contributes its portion to the overall composite application cost.

Figure 3: Composite Application associated with the Characteristics Domain.

2.1.1 Definitions

Entity is defined as a Base Configuration Item entity, implemented and well known in the standard IT configuration management database (CMDB) systems. It is an abstract element, which is not presented as a selectable modeling entity in the QDSL visual editor.

Software Component is a logical software entity that implements a functional behavior, captured as a collection of installed files, and runs on a Hardware Component. A Software component exhibits the behavior of an IT Service.

Computer System entity represents a computer system. The Type element classifies the computer as a "Server," "Mainframe," "Personal Device," etc. Its sub-elements define its hardware, IP networking, proxy/management protocol, and computer-related information.

A Composite Application entity is a common term for the end-to-end IT solutions that implement business services. IT implementations of business services, or any cloud services for that matter, are composites of hardware, software, applications, data, network, etc., where the function needed for the service is driven from several different sources, regardless of ownership of these sub-modules. More specifically, when examining the end-goal of an IT service to cater for business (or other) cloud services, the delivered application is the one that is perceived as the service in question. Thus, the resources comprising such a service (e.g., network elements and servers) are a means to delivering the service solution.

2.2 The Characteristics Domain

An IT service or composite application is assessed by monitored *Measurement*, representing a certain *Characteristic* of a Service (Figure 4). A characteristic can be computed either from other sub-characteristics, or directly from an associated *Assessment*.

Figure 5 exemplifies a computation structure of a cost *Characteristic*. In this example, the cost *Characteristic* scoring is derived from the labor cost *Characteristic* score, which is computed from the salary measurement. The contribution considers only 10% of an employee's monthly payment, according to the financial system. The overall cost is equally averaged with a computed cost extracted from a relative cost assessment of an Amazon server, according to an hourly rate.

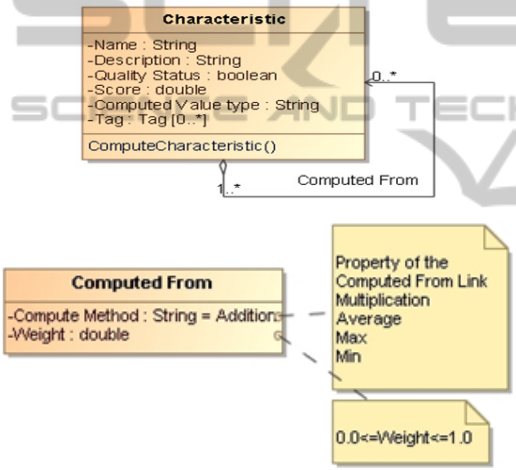

Figure 4: The Characteristic Domain.

2.2.1 Definitions

Characteristics entity describes the quality attributes of a service or a component or any *Entity* type. *Characteristic* examples are security, usability, testability, maintainability, extensibility, scalability, portability, interoperability, and availability. As an example, availability is defined as the ability of a component (software, hardware, or any *Configuration Item*) or IT Service to perform its agreed function when required. Availability is determined by Reliability, Maintainability, Serviceability, Performance, and Security, and is usually calculated as a percentage. This calculation is often based on agreed service uptime and downtime.

Computed From relationship is linking (1) another *Characteristic* entity that provides weighted

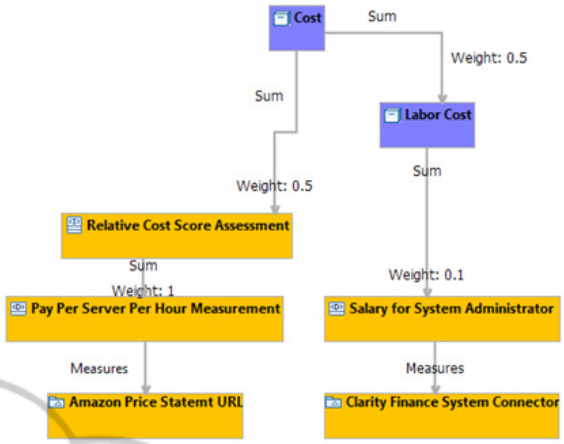

Figure 5: Complex structure of characteristic and assessments dependency.

aggregation of an existing *Score*, or (2) an *Assessment* that is linked to *Measurement* (metric or indicator), which provides the *Score* value by calculating the *Scores* equation presented below. The **Compute Method** of the *Computed From* relationship enables aggregation according to several possible mathematical functions. These functions can be: a summation by adding or removing values, Max, Min, Average, and weighted average. All functions are multiplied by a **Weight** attribute that reflects the level of importance of the targeted element. When normalized, the sum of all targeted *Characteristic Weights* should be 1.

The **Score** attribute is a percentage representation. The calculation method of the *Score* is defined in Equation 1 in the Measures Domain section. It is important to note that in the case that a *Characteristic* is computed from other sub-*Characteristic*s, it must be supported by an *Assessment* entity to provide the initial *Score* value.

Computed Value type attribute describes the type of the numerical value: percentage (default), count, ratio, or Boolean. *Scores* that are not of the same type cannot be jointly aggregated.

Quality Status attribute indicates whether or not the requirements for all the underlying *Assessments* and sub-*Characteristics* are met. *True* represents a healthy system, and *False* represents that a Service Level Agreement (SLA) or requirement is not achieved. Another condition for a *True* value is that all underlying scores are greater than 0.0.

Compute Characteristic () - enables construction of a dedicated algorithm to calculate the *Score* or computed values of the entity.

2.3 The Measures Domain

The Measures Domain (Figure 6) contains elements that retrieve data based on metrics (machine or automation driven) and indicators (people, statistics driven) using *Measurements*. In turn, *Measurements* access specific connectivity to the real world via *Measured Objects*. Each *Measured Object* is associated with an entity from the *Composite Application* Domain (see Figure 1), and is transformed into *Characteristic's Scores* by the *Assessment* entity according to an accepted range levels. A single *Assessment* may have many *Rating Requirements*, consequently catering for multiple business goals and different stakeholders.

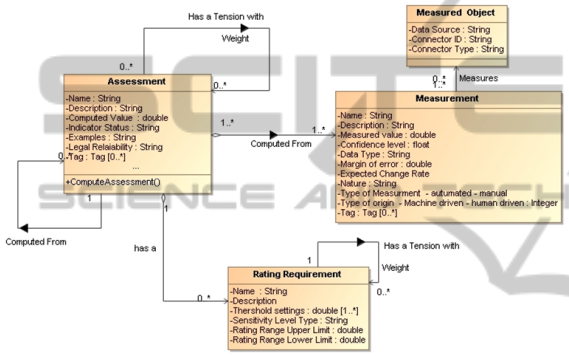

Figure 6: The Measures Domain.

2.3.1 Definitions

Measurement entity data contain the monitored data of an *Entity* or configuration item, by producing a metric. The resultant **Measured Value** attribute must be nonnegative and additive, meaning that the value of two non-overlapping *Measurements* equals the sum of their individual *Measure Values*.

Assessment entity captures a method for aggregating or converting *Measurements* into *Scores* that are evaluated by the *Characteristics* entity utilizing associated *Rating Requirement's Threshold* values. The assessment captures the level sets of a Service/Operation Level Agreement (SLA/OLA). The calculation of SLAs or *Scores* is triggered on-demand by an external entity, using the **ComputeAssessment** () method.

Rating Requirement entity defines the accepted range of values that are used when the *Measured Value* is assessed, computing the associated *Characteristics* entity's *Score* (percentage, distance, or ratio). Many *Characteristics* exist for a single *Measurement*, depending on the structure of the *Composite Application*. Fundamental to cloud environments, the evaluation structure reduces the quantity of *Measured Objects*, improving monitoring capacity. This many-to-many relationship between *Characteristics Assessment* and *Measurement* supports the scalability and multi-tenancy of monitored cloud services, where a single monitoring tool can support many observers, and vice versa. **Scores** calculation is based on many quality requirements of different stakeholders, expressed as an acceptable *Rating Range* for a quality attribute of a service. Accordingly, multi-tenants can use the same QDSL system, and exhibit different levels of satisfaction with the same evaluated service, due to different *Rating Requirements*.

Scores are thus computed in equation 1 as:

$$Score_i = \begin{cases} 100.0 & \text{if } ACV \geq RRU_i \\ \frac{(ACV - RRL_i)}{(RRU_i - RRL_i)} & \text{if } RRU_i > ACV > RRL_i \\ 0.0 & \text{if } ACV \leq RRL_i \end{cases} \quad (1)$$

where *i* indicates the relevant *Rating Requirement*, *ACV* is the *Assessment* Computed Value, RRL_i is *Rating Range Lower Limit* of the *i Rating Requirement*, and RRU_i is *Rating Range Upper Limit* of the *i Rating Requirement*. The *Score* is provided to the relevant *i Characteristic.*

Has a Tension with relationship highlights that deeper evaluation is needed when examining an *Assessment*. The different values (tensions) are:

- *Constructing*: the contribution is sufficient to construct another *Assessment*.
- *Supporting*: the contribution is needed but is not sufficient to support another *Assessment*.
- *Destructing*: the contribution conflicts with another *Assessment* and cannot co-exist.
- *Degrading*: the contribution diminishes the affected *Assessment* outcomes, however, not entirely.

Legal Reliability reflects the fidelity of the data sources and their information reliability. *True* represents reliable. For example, automated, machine driven metrics have a higher reliability level than human, manually generated information.

Threshold Settings is an array of acceptable range. defines a Threshold as the value of a metric (key performance indicator) that should generate an alert or take a management action. For example: "Priority1 incident not solved within 4 hours"; "more than 5 soft-disk errors in an hour"; or, "more than 10 failed changes in a month".

Measured Object entity enables connection to and monitoring of a Configuration Item (CI) and provides the measured data. Agents, connectors, adapters and other installed or remotely accessed

API are examples. In the cloud domain, due to encapsulation and abstraction of the physical assets by means of virtualizations, most of these measured objects are connectors that abstract remote monitoring calls.

3 QDSL IMPLEMENTATION

The QDSL prototypical visual modeling tool was developed based on the Eclipse Modeling Framework and ECORE models, including DSL constraints language, such as Object Constraints Language (OCL). The QDSL tool enables people to interact with the design environment of IT quality assessments using visual modeling editors.

The QDSL tool enables modelers to model visually the algorithm for computing the quality assessment and express how to capture basic measurements. QDSL graphically associates the needed transformation that captures measurements into a comparable, normalized scoring system, including methods for aggregation, average, or accumulation of scores across composite application characteristics. The tool exports the visual represented models to machine-readable files (XMLs) that can be interpreted by a run-time computation engine, according to the defined visual algorithm. The approach restricts the relations between the computation elements by adhering to a constrained environment (using OCL), prohibiting human error.

The codified models can be transportable to real-time monitoring and computation tools. One of these tools, termed CA Business Insight, commercially implements structured measurements, known as the Service Measurement Index (SMI) (Zachos, 2011).

4 DISCUSSION AND CONCLUSIONS

The Quality Domain Specific Language (QDSL) and its associated modeling tool enable modelers to capture quality assessment algorithms for evaluating cloud services. Using QDSL graphical editors and its underlying Domain Specific Model enables modelers to capture cloud services' qualities accurately and concisely, share algorithms, and reuse quality structures. By rapidly instantiating the model into machine-readable format, QDSL models can be executed on computation engines that aggregate the metrics and produce benchmarking and quality dashboards. The QDSL-based conceptual model supports a multi-tenancy approach for both reduction of monitored information and tailored derived dashboards (personalization, role based), functioning as a cloud service for evaluating composite applications quality.

REFERENCES

Adam, S., Doerr, J., 2007. On the Notion of Determining System Adequacy by Analyzing the Traceability of Quality. *Advanced Information System* Donzelli, P., Bresciani, P., 2004. Improving Requirements Engineering by Quality Modeling A Quality-based Requirements Engineering Framework. *Journal of Research and Practice in Information Technology*, Vol. 36 Issue 4, pp. 277-294.

Ferguson, D.F., Hadar, E., 2010. Constructing and evaluating supply-chain systems in cloud-connected enterprise. In *5th international conference on software and data technologies, ICSOFT 2010*, Athens, Greece, July 2010

Ferguson, D.F., Hadar, E., 2011. Optimizing the IT business supply chain utilizing cloud computing. *The 8th International Conference on Emerging Technologies for a Smarter World (CEWIT2011),* Long Island, Hauppauge, New York, November 2-3, 2011.

Frank, U., Heise, D., Kattenstroth, H., Ferguson, D.F., Hadar, E., Waschke, M.G., 2009. ITML : A Domain-Specific Modeling Language for Supporting Business Driven IT Management. *Proceedings of the 9th OOPSLA workshop on domain-specific modeling (DSM), OOPSLA 2009*, Orlando, Florida, US,.

Gruber, T., 1995. Towards principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies,* 43(5/6), pp. 907-928.

Kupfer, M., Hadar, I., 2008. Understanding and Representing Deployment Requirements for Achieving Non-Functional System Properties. In *International Workshop on Non-functional System Properties in Domain Specific Modeling Languages, affiliated with MoDELS*, Toulouse France, September 2008.

Zachos, K., Lockerbie, J., Hughes, B., Matthews, P., 2011. Towards a framework for describing cloud service characteristics for use by chief information officers. In *Requirements Engineering for Systems, Services and Systems-of-Systems (RESS),* pp: 16 – 23, Trento, Italy, Aug 2011

Franch, X., Carvallo, J.P. 2003. Using quality models in software package selection. *IEEE Software*, vol. 20(1), pp.34–41