Modeling Elasticity Trade-Offs in Adaptive Mixed Systems

Muhammad Z.C. Candra, Hong-Linh Truong, Schahram Dustdar
Distributed Systems Group
Vienna University of Technology
Argentinierstraße 8/184-1, A-1040 Vienna, Austria
Email: m.candra,truong,dustdar@dsg.tuwien.ac.at

***Abstract*—In the past decade, elastic computing has emerged as a popular solution approach for on-demand computing resources' provisioning. Today, such dynamic resource provisioning is used not only in machine-based computing systems but also in mixed systems containing machine- and human-based computing elements. Many efforts have been undertaken to achieve resource elasticity. However, to provide a truly elastic computation, system designers also need to foresee trade-offs between cost, quality, and resources. Unfortunately, the lack of modeling tools leads to difficulty in designing on-demand provisioning strategy in mixed systems. In this paper we present a modeling tool named *Elasticity Profile (EP)* that specifies constructs for modeling the elastic behavior of mixed systems with respect to trade-offs between cost, quality, and resources. We also present a conceptual framework where our EP can be deployed, utilized, and bound to runtime systems. Furthermore, we demonstrate the suitability of EP model using several use cases.**

***Keywords*—cloud computing, human computing, elasticity strategy, elasticity modeling tool**

I. Introduction

In the plethora of cloud computing today, we are seeing elastic computing as a popular mechanism for achieving dynamic resource provisioning. Elasticity defines one key quality of cloud computing: resources must be managed by scaling up and down as needed so that limited resources can be offered for potentially unlimited uses [1]. When designing a system with an elastic behavior, it is important to consider not only the resources but also the trade-off between cost and quality. Let us consider, for example, a typical Software-as-a-Service (SaaS) of a complex application which consists of many application components; each component has its own quality metrics such as performance, reliability, throughput, and so on. These quality metrics may be dynamically specified by the customer and affect the SaaS provider's decision to scale up/down resources. These changes eventually affect the cost needed for the resource provisioning and the cost charged to the customer.

Nowadays, human-based computation also plays an important role in overall service architectures. Many efforts have been done for dynamic human-based resources provisionings on the cloud such as found in *crowdsourcing*. A mixed system is a system that contains a mix of both, machine-based computing elements (MCEs), and human-based computing elements (HCEs). The following simplified cases illustrate elastic behaviors in the context of MCEs and HCEs:

- If the CPUs utilizations of existing machine instances are above 80%, then an additional machine instance must be added.
- When the average utilization of the human workers on a running pool is above 8 hours per day, then additional workers must be assigned to the pool.
- A human-task requester wants to pay a cheaper price if the worker takes more than 1 hour to finish the task.

In such situations, several challenging questions may arise. How can providers and/or requesters model the cost of a service based on the quality adjustment? And furthermore, how the reseources can be scaled based on the quality changes?

To the best of our knowledge, currently there are no tools which can be used to effectively model the aforementioned elastic behavior for *both* MCEs and HCEs. In this paper, we introduce *Elasticity Profile (EP)* as a tool for explicit modeling of systems' elastic behavior to achieve a dynamic resource provisioning. In EP, we introduce the notion of adaptable objects and their corresponding metrics, which are applicable for MCEs and HCEs. Furthermore, EP also provides declarative constructs for defining behaviors to model the dynamic relation between those objects and metrics. Using our proposed tool, a system designer can model system's behaviors as dynamic relationships between resources, cost, and quality. EP is designed to enhance services, applications, and workflows, so that the resources, cost, and quality can be adapted based on the changing runtime environment.

The main contribution of our research is to provide a modeling tool and framework for describing, deploying, and utilizing the elastic behavior of adaptive mixed systems. In particular, our work presented in this paper focus on the following:

- *Elasticty Profile (EP)*, which specifies constructs that can be used to model trade-offs and the dynamic provisioning of resources, and
- *a conceptual runtime framework for EP* as a mechanism and platform for EP deployment and execution.

Furthermore, an evaluation to the proposed model is conducted by modeling several use cases using EP.

This paper is organized as follows. Section II discusses some works related to the dynamic resources provisioning for human-based and software-based systems. Details of the EP

978-0-7695-5002-2/13 $26.00 © 2013 IEEE
DOI 10.1109/WETICE.2013.51

model that consists of objects, metrics, activities, and rules are presented in Section III. In Section IV, we discuss how the model is loosely coupled with the runtime system, and how an implementation can be bound. In Section V we present the conceptual runtime framework for EP. Section VI demonstrates the applicability of our model using some example use cases. Finally, Section VII concludes the paper.

II. Related Work

Recent surveys on elastic computing, e.g., [2], [3], show a significant amount of works that deal with elasticity strategies. Several techniques for achieving autonomic elasticity based on traditional sensor-actuator mechanisms have been proposed by both industry and academy, e.g., [4]–[6]. Other techniques allow execution of elasticity strategies based on user-driven policies. A quite common method for defining such policy is using a rule-based approach, e.g., [7], [8]. Most of the aforementioned works focus on achieving an efficient resource scaling on a certain layer. Our work focus on providing a tool for explicit modeling of multi-dimensional elastic behaviors that considers the trade-offs between resources, cost, and quality. Furthermore, we introduce the concept of elastic objects as an encapsulation of live runtime entities that can be bound to any remote objects. This concept allows easier abstraction on the dynamicity of the runtime environment since an object may represent any entities on any service layers, e.g., it can represent a component on an infrastructure, platform, application, or data layer. Moreover, current elastic computing efforts do not take into account the emerging human-based computing resources provisioning on the cloud such as crowdsourcing.

Another approach to program elasticity is proposed by using SYBL directive language [9]. SYBL can be implemented in particular languages for assisting the compilation and execution of application, so that it supports elasticity. For example, SYBL for Java can be used to inject an object representing a cloud resource into a subsequent variable. Our EP is more focus on the modeling of the elastic runtime environment instead on providing language to program elastic applications. Furthermore, a single EP document can be used for different applications implemented in different languages, as long as they concern about the same objects and metrics as discussed in Section III. EP extends SYBL in a way that a profile can be attached to a SYBL-enriched application to control the runtime adaptation strategy without changing the application logic.

In addition to services, applications, and workflows, our EP model can also be used with configuration frameworks such as the emerging *Topology and Orchestration Specification for Cloud Applications (TOSCA)* framework [10]. TOSCA is an OASIS standard to enable a portable description of cloud applications and services. To enable elastic behavior on a portable cloud service, an EP can be defined and attached to the corresponding TOSCA configuration.

Recently, the concept of dynamic resource provisioning is also applicable for human-based computing resources. The Human-Provided Service (HPS) concept introduces a framework that allows workers to publish their capabilities and skills as services [11]. To extend this concept to a team of worker, the Social Compute Unit (SCU) is introduced as a construct, which consists of loosely coupled virtual and nimble teams of computing resources with specific skills in specific problem domains [12]. Also, some efforts have been done to extend the BPEL [13] to support human capabilities. In particular, BPEL4People [14] is the BPEL extension to enable human interactions in business processes. BPEL4People is based on WS-HumanTask specification [15], which introduces the definition of human tasks. These works allow integration of human-based resources into processes. However, humans are known to have very dynamic quality properties. Our framework offers a methodology to model the relationships between non-functional properties of human-based computing resources, and the process' quality and cost.

III. Elasticity Profile

In our framework, an elastic behavior is modeled by the notion of *Elasticity Profile (EP)*. An EP can be used in an application layer or a platform layer. In the application layer, an EP can be attached to workflows or distibuted applications. And in the platform layer, an EP can be used to complement the system configuration. An EP contains constructs for defining *objects*, *metrics*, *behavior*, and *activities* using a syntax shown in Grammar 1. The following subsections discuss these constructs.

```
⟨ep⟩ ::= profile ⟨identifier⟩ { ⟨statement⟩* }
⟨statement⟩ ::= ⟨objects_statement⟩
   | ⟨metrics_statement⟩
   | ⟨activities_statement⟩
   | ⟨behavior_statement⟩
```

Grammar 1. Overall Grammar of Elasticity Profile (EP)

A. Objects and Metrics

In an adaptive system, we deal with objects and manipulation of the objects. The objects in EP are representations of any components of a system or a process that can behave elastically. Models described using EP focus on objects and the adaptation of objects' properties, which can be measured through metrics. In order to make these objects adaptable, two steps are needed: first, properties must be associated with the objects during the modeling phase; and second, at runtime, the adaptation strategies are decided for these objects based on their properties and runtime information.

When modeling an elastic behavior of a system using EP, we first need to identify which objects/components of the system should be made adaptable. For example,

a) *for MCEs*, the *objects* can be
- instances of real- or virtual-machines (e.g., instances can be terminated or a new instance can be created),
- storage as in Data-as-a-Service (e.g., the capacity of a storage can be increased or decrease), and

b) *for HCEs*, the *objects* can be
- human-based tasks (e.g., non-functional requirements such as deadline and acceptance criteria can be changed),
- human workers (e.g., the fee and quality rating can be dynamic).

TABLE I. Example of Machine and Human Metrics

Metric Dimension	*Machine Metrics*	*Human Metrics*
Resources	Number of resources, utilization, storage capacity, bandwidth capacity	Number of resources, utilization
Quality	Response time, throughput, availability	Response time, rating, availability, throughput, task acceptance rate
Cost	Cost / API calls, virtual instance / hours	Task price, hourly price

To model properties of the objects, we define a set of `metrics`. In the modeling phase, each object are associated with multiple metrics. These metrics represent the quality, resource, and cost properties of the objects, e.g., such as in [16], [17]. Table I lists some examples of metrics for machine-based and human-based elements. Inside an EP, `objects` and `metrics` are declared without implementation using a syntax shown in Grammar 2. In our framework, these `objects` and `metrics` (as well as `activities`) are lately bound during runtime as discussed in Section IV.

⟨*objects_statement*⟩ ::= `objects { `⟨*objects_list*⟩` } ;`

⟨*objects_list*⟩ ::= ⟨*object_identifier*⟩
 | ⟨*object_identifier*⟩ `,` ⟨*objects_list*⟩

⟨*metrics_statement*⟩ ::= `metrics { `⟨*metrics_list*⟩` } ;`

⟨*metrics_list*⟩ ::= ⟨*metric*⟩
 | ⟨*metric*⟩ `;` ⟨*metrics_list*⟩

⟨*metric*⟩ ::= ⟨*object_identifier*⟩ `has` ⟨*metric_method_list*⟩

⟨*metric_method_list*⟩ ::= ⟨*metric_method*⟩`[(`⟨*value*⟩`)]`
 | ⟨*metric_method*⟩`[(`⟨*value*⟩`)] ,` ⟨*metric_method_list*⟩

Grammar 2. Objects and Metrics Grammar

B. Behavior and Activities

A system designer models the behavior of a system to achieve a set of goals. For attaining the goals, the designer can define a set of trade-offs between resources, cost, and quality of the system. In EP, these trade-offs are decribed using first-order logic.

For example, in the context of human-tasks workflow, our goal is to maintain the human workers' performance by ensuring that these resources are not overloaded. A individual worker is said to be overloaded when the utilization is above 8 hours per day. Therefore, we would like to make a trade-offs: for ensuring performance we should add additional human workers when the utilization is reaching 8 hours. We could model this behavior using the following first-order logic:

$$\begin{aligned}
&\forall(worker, pool)\\
&\quad Worker(worker) \wedge ActivePool(pool) \wedge\\
&\quad IsMember(worker, pool) \wedge\\
&\quad HourUtilization(worker) >= 8\\
&\quad \Rightarrow AddWorker(pool)
\end{aligned}$$

Our EP model also has the potential to be used for designing a system that follows a certain compliance requirement. For example, we have a Data-as-a-Service (DaaS) with a certain data location requirement: because of regulatory laws, all data for European customers must be placed in Europe. We could model this compliance requirement in EP using the following rule:

$$\begin{aligned}
&\forall(data, customer)\ \exists(datacenter)\\
&\quad Data(data) \wedge Customer(customer) \wedge\\
&\quad Datacenter(datacenter) \wedge\\
&\quad BelongsTo(data, customer) \wedge\\
&\quad Location(customer) ==' europe' \wedge\\
&\quad Location(datacenter) ==' europe' \wedge\\
&\quad Available(datacenter)\\
&\quad \Rightarrow AssignDatacenter(data, datacenter)
\end{aligned}$$

The `behavior` section of an EP contains several rules in the form of implication statements. During runtime, the objects and their metrics are asserted and serve as the fact for these rules. Each of these rules contain two parts: the conditions and the consequences. Optionally, a *priority* attribute that defines the order of the rules' evaluation may also be specified for each rule. Grammar 3 shows the syntax of the `behavior` section in EP.

When the condition of a rule is evaluated to $true$, its consequence part is applied. Expressions in the condition part use objects and metrics defined previously as operands. Borrowing from Drool language [18], the expression in the condition part may contain an *instance binding* that allows a reference to an instance of object obtained from a collection of constrained objects on which the expression is evaluated. This is particularly useful for evaluating and modifying the metric of a particular object instance. Furthermore, other features for condition expression from Drool language, such as accumulator for measuring aggregated metrics, are also adopted.

Common operators normally found in first-order logic can be used in the condition expression to evaluate objects and metrics. These include logical operators, relational operators, and quantifiers. Furthermore, we also introduce two *loose constraint operators*: *(highest)* and *(lowest)*, which are applicable to any metric that belongs to an object in the collection. These operators can be used to loosen up constraints, so that we could still achieve the goal with the best effort. For example, when we evaluate worker offers, we can assign a task to a worker whose location is the nearest possible (lowest distance) to Vienna.

The consequence part of a rule may contain one or more statements. The following four types of statements can be used:

1) *Assignment* statement is used to change a metric of an object, e.g., changing the fee of a task.
2) *Assertion* statement is used to assert an object to the fact base. The object asserted can be either a new one or a replacement of an existing object.
3) *Invocation* statement is used to invoke an operation provided by the runtime platform. For example, it can be used to invoke remote operations for creating a new instance of a computing resource.
4) *Exception* statement is used to raise an error that should be further handled by the runtime platform. For example, it can be used when the deadline of a task is approaching.

In EP, the remote operations and exception handlers are declared in the `activities` section using syntax defined in Grammar 4.

⟨*behavior_statement*⟩ ::= `behavior { `⟨*implication_list*⟩` } ;`

⟨*implication_list*⟩ ::= ⟨*implication*⟩
  | ⟨*implication*⟩ `;` ⟨*implication_list*⟩

⟨*implication*⟩ ::= `check [:`⟨*priority*⟩`] (`⟨*condition*⟩`) {` ⟨*consequences*⟩ `}`

⟨*consequences*⟩ ::= ⟨*consequence*⟩ `|` ⟨*consequence*⟩`;` ⟨*consequences*⟩

⟨*consequence*⟩ ::= ⟨*metric_identifier*⟩ `=` ⟨*value*⟩
  | `assert` ⟨*instance_identifier*⟩
  | `trigger` ⟨*action_identifier*⟩`(`⟨*value_list*⟩`)`
  | `throw` ⟨*exception_identifier*⟩ `(`⟨*value*⟩`)`

Grammar 3. Behavior Grammar

⟨*activities_statement*⟩ ::= `activities {` ⟨*activities_list*⟩ `} ;`

⟨*activities_list*⟩ ::= ⟨*activity*⟩
  | ⟨*activity*⟩`,` ⟨*activities_list*⟩

⟨*activity*⟩ ::= ⟨*activity_identifier*⟩ `(`⟨*activity_param_list*⟩`)`

Grammar 4. Activities Grammar

Interested readers may get the complete EP syntax provided on the supplemented material online[1].

IV. Runtime Binding

To be meaningful and useful, the objects, metrics, and activities defined in *Elasticity Profile (EP)* must be bound to real entities in the system at runtime. The objects, metrics, and activities constructs in EP are loosely coupled from their implementations. Therefore, designing an elastic behavior requires less assumptions about the runtime system. This approach allows the seperation of concern between the design of the elastic behavior and the implemention of the runtime system. This is particularly useful, for example, when an application or a service should be deployed and executed on different runtime platforms provided by different vendors.

Elastic Reasoning Agent (ERA) (discussed in Section V) is responsible for interpreting an EP and bind the objects, metrics, and activities using certain mechanisms provided by the runtime systems or their proxies. The aforementioned constructs in EP are bound to real entities using the following mechanisms:

a) Objects: are bound using subscriptions to event notifications. ERA sends a subscription request to the runtime system and the runtime system send a notification message, which contains the object, to ERA when the object being monitored is created or modified in the runtime system. For example, ERA may ask a human-based workflow engine to send a notification when a human-based task is instantiated.

b) Metrics: are bound using remote getter and setter methods. A metric may either have both getter and setter methods or only a getter method (i.e., a read-only metric). For example, a metric obtained from a predictive algorithm for measuring the future reliability of a system is read-only. Upon evaluation of a rule during runtime, when a metric is requested or assigned with a new value, ERA invokes its getter or setter methods.

c) Activities: are bound using remote method invocations. ERA simply calls this procedure using the provided parameters when necessary.

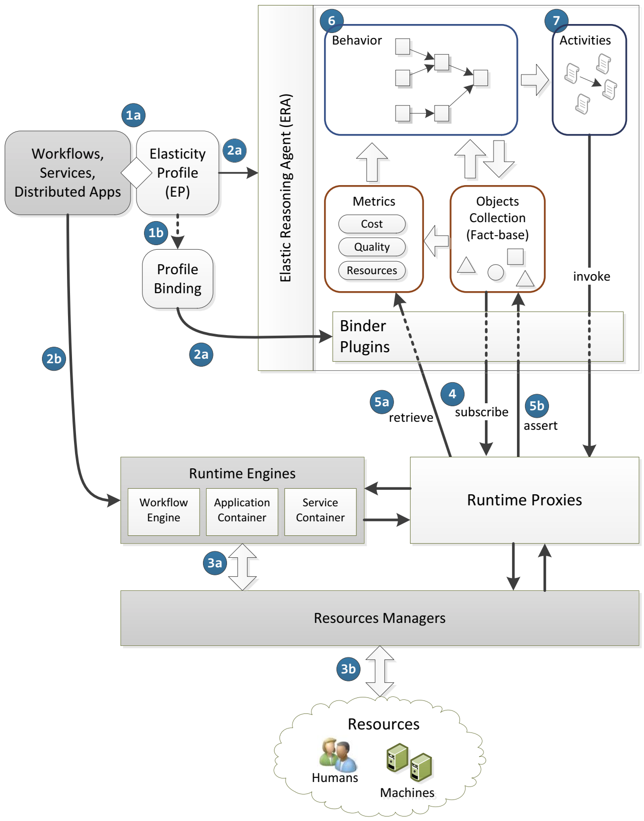

Fig. 1. Conceptual Runtime Framework. (1) An EP is attached to a workflow/application and an associated binding is defined. (2) The EP & workflow/application are deployed. (3) The runtime layer executes the workflow/application and manages runtime resources. (4) Using binder ERA subscribes for object assertions through proxies. (5a) Runtime metrics are retrieved. (5b) Runtime objects are asserted. (6) Reasoning based on behaviorial rules decides what (7) activities to take.

The runtime binding mechanism is defined separately from EP. The binding could utilize standards protocols for remote invocations such as Remote Procedure Call (RPC) or Web service protocols (e.g., Java RMI, SOAP, RESTful, etc.). An example of an EP runtime binding using RESTful Web service can be found on the supplemented material.

V. Runtime Framework

In this section, we present the conceptual runtime framework that enables integration of *Elasticity Profile (EP)* with runtime systems by means of *Elastic Reasoning Agent (ERA)* as depicted in Figure 1. ERA manages the collection of objects defined in EP and their associated metrics. ERA has the capability to reason about strategies to obtain elastic behaviors based on the behaviorial rules provided in an EP using a *production rule system*. ERA is also empowered by *binder plugins* that contains implementations for specific remote method invocation protocols. Several bindings may be active for the same EP to allow integration of different systems.

The *resource managers* manage resources required for executions by *runtime engines*. This underlying runtime layer provides the execution platform and resource management. This platform can be in the form of a service or application

[1] http://www.infosys.tuwien.ac.at/prototypes/ElasticityModeling/

container deployed on a cloud infrastructure, or a scientific or business workflow engine. This underlaying layer can also be a system that utilizes a crowdsourcing platform as a human task execution environment. These underlying runtime layer entities are beyond the scope of this paper. *Runtime proxies* may be required and implemented using specific APIs for a particular runtime system. These proxies enable ERA to interact with runtime systems for obtaining objects and their metrics, as well as invoking remote actions.

An EP can be attached to workflows, services, distributed applications, or system configurations. Before being deployed, a binding for the EP should be defined based on a specific protocol provided by the runtime systems or their proxies. During the deployment stage, an EP and its binding specification are deployed to ERA and the workflow/service/application is deployed to a runtime engine. The EP deployed to ERA contains all definitions required to achieve the desired elastic behavior. The objective of modeling elasticity is essentially to define the behavior of a process or system in response to the changing properties of the system's objects. These objects are asserted by the runtime layer as requested by ERA through a binding mechanism as discussed in Section IV. During execution, objects may be created and asserted to the ERA's fact base; or they can be destroyed and removed from the fact base. Furthermore, the EP also contains metrics definition for these objects and as necessary ERA may call remote methods provided by the runtime layer to get or to set the metrics. Through these metrics we can capture the non-functional properties of the objects (e.g., resources) involved in runtime. Together, these objects and metrics will be utilized by ERA to decide whether it is necessary to invoke adaptability activities during runtime.

VI. Use Cases

This section demonstrates an elastic behavior modeling using *Elasticity Profile (EP)*. Our purpose in evaluating our model is to study its applicability for real use cases. The use case presented here exemplifies some main features of our model. More use cases and complete EP codes can be found online on the supplemented material.

Here, we present a typical workflow for a monitoring and management service of IT infrastructures as shown in Figure 2. This system contains a Social Compute Unit (SCU) construct, which represents virtual and nimble teams of experts [12]. A single SCU instance represents a single execution unit that consists of a group of experts with different skills required for solving a task.

The system monitors and manages IT infrastructures owned by customers. Sensors on the infrastructures generate events, which then captured by a pool of MCEs running analyzer software. When a suspicious event is detected, the analyzers raise a warning to the monitoring agents. These stand-by human agents analyze the warning further. If it is believed that the warning requires an investigation, the agent issues a ticket which indicates the incident. The issuance of the ticket triggers an initialization of an SCU, which contains experts with a set of skills required for investigating the incident and administering the infrastructure to fix the problem. A mechanism for composing such SCU is discussed in [19].

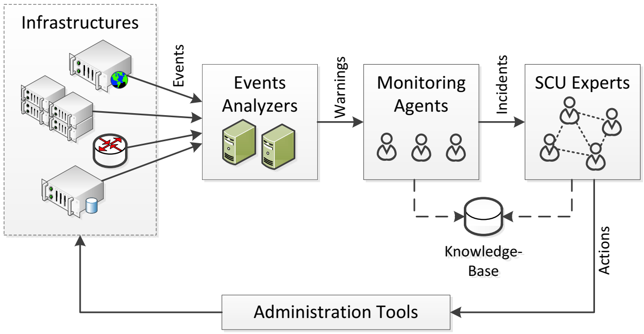

Fig. 2. SCU-based IT Infrastructure Monitoring and Management

Using EP, as shown in Listing 1, we model this system as as a system that has an elastic behavior with respect to three dimensions: cost, quality, and resources. From cost perspective, customers have options to choose two types of services, which are premium and regular services. This categorization implies different guarantees with respect to quality of service (QoS), especially the performance of the service which represents the maximum response time for handling an incident. Therefore, these QoS guarantees lead to different strategies for provisioning resources (both MCEs and HCEs).

```
profile SCU_IT_Management {
  objects {
    Customer, Event, Warning, Incident, Analyzer,
    MonitoringAgent, ExpertSCU
  };
  metrics {
    Customer has ServiceType, ...;
    Incident has Lifetime, ...;
    Analyzer has Utilization, Type, ...;
    ExpertSCU has ExpertiseLevel, ...;
    ...
  };
  actions {
    AddAnalyzer(ANALYZER_TYPE),
    ReduceAnalyzer(ANALYZER_TYPE),
    AddMonitoringAgent(),
    ReduceMonitoringAgent(),
    UpgradeSCU(ExpertSCU, EXPERTISE_TYPE),
    TimeoutException(Incident),
    ...
  };

  behavior {

    /* Dynamically scale analyzer for premium
       service based on the average utilization
       of the premium analyzers */
    check (Number(doubleValue > 0.8)
           from accumulate(
             Analyzer(Type==PREMIUM_MACHINE and
                      u:Utilization),
                      average(u))) {
      /* scale up */
      trigger AddAnalyzer(PREMIUM_MACHINE);
    };
    check (Number(doubleValue < 0.2)
           from accumulate(
             Analyzer(Type==PREMIUM_MACHINE and
                      u:Utilization),
                      average(u))) {
      /* scale down */
      trigger ReduceAnalyzer(PREMIUM_MACHINE);
    };

    /* Scale monitoring agent based on the number of
       queued warnings */
    check (Number(intValue > 20)
           from accumulate(w:Warning(), count(w))) {
       /* scale up */
       trigger AddMonitoringAgent();
    };
    check (Number(intValue < 5)
           from accumulate(w:Warning(), count(w))) {
       /* scale down */
       trigger ReduceMonitoringAgent();
    };

    /* Upgrade SCU when the deadline is
       approaching*/
    check (Incident(Lifetime > 2 * 3600 and
         getCustomer().ServiceType==PREMIUM and
         scu:getAssignedSCU()) and
         (scu.ExpertiseLevel < HIGH_EXPERTISE) ) {
       /* increasing expertise level,
          i.e., it will add more experts with
          higher expertise */
       trigger UpgradeSCU(scu, HIGH_EXPERTISE);
    };

    /* Timeout exception for premium customer */
    check (i:Incident(Lifetime > 4 * 3600 and
             getCustomer().ServiceType==PREMIUM) ) {
       throw TimeoutException(i);
    };
    ...
  };
}
```

Listing 1. EP for SCU-based IT Infrastructure Monitoring and Management

The type of service taken by customers affects how resources are provisioned for handling incidents from the customers' infrastructures. In the EP snippet depicted in Listing 1, we show how to obtain dynamic resources provisioning for premium customers. To assure fast event analysis, we scale the analyzer machines up and down so that the average utilization is between 0.8 and 0.2. Furthermore, when an incident is not resolved within 2 hours, we "upgrade" the assigned SCU by adding more experts with higher expertise level. If after 4 hours the incident is still not resolved, an exception is thrown, e.g., it may notify the supervisor or the manager so that they can take strategic actions. Moreover, our snippet also shows how we can scale the pool of the monitoring agents to assure that the number of raised warnings in the queue is not too much or too few. Note that the metrics' thresholds and actions can be specified differently for other types of customers using different sets of EP rules.

VII. Conclusion and Future Work

In this paper we introduce *Elasticity Profile (EP)* as a tool for modeling the elastic behavior of a system. We present some entities required to model the desired behavior and how this model can be serialized into a profile document and bound to runtime implementations. We also discuss the conceptual runtime framework for EP that can be used to integrate EP platform with runtime systems. Furthermore, we also show the applicability of EP using some use cases.

The framework presented in this paper is part of our ongoing research on human based computation and elastic computing. One of main challenges is to obtain metrics' measurement models for human based computing elements with respect to the human dependability properties. As part of our continuing work, we are also interested in investigating elasticity discovery and negotiation, so that we could automatically match service requests with service offerings.

Acknowledgements

The first author of this paper is financially supported by the Vienna PhD School of Informatics[2].

References

[1] S. Dustdar, Y. Guo, B. Satzger, and H.L. Truong. Principles of elastic processes. *Internet Computing, IEEE*, 15(5):66–71, 2011.

[2] L.M. Vaquero, L. Rodero-Merino, and R. Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[3] G. Galante and L.C.E. Bona. A survey on cloud computing elasticity. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 263–270. IEEE, 2012.

[4] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 559–570. IEEE, 2011.

[5] H.C. Lim, S. Babu, and J.S. Chase. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*, pages 1–10. ACM, 2010.

[6] Amazon Web Services. Auto scaling. http://aws.amazon.com/autoscaling/. [Online; accessed February-2013].

[7] L. Rodero-Merino, L.M. Vaquero, V. Gil, F. Galán, J. Fontán, R.S. Montero, and I.M. Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226–1240, 2010.

[8] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L.M. Vaquero. Service specification in cloud environments based on extensions to open standards. In *Proceedings of the fourth international ICST conference on communication system software and middleware*, page 19. ACM, 2009.

[9] G. Copil, D. Moldovan, H.L. Truong, and S. Dustdar. Sybl: an extensible language for controlling elasticity in cloud applications. In *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE/ACM, 2013.

[10] T. Binz, G. Breiter, F. Leyman, and T. Spatzier. Portable cloud services using tosca. *Internet Computing, IEEE*, 16(3):80–85, 2012.

[11] D. Schall, H.L. Truong, and S. Dustdar. The human-provided services framework. In *10th IEEE Conference on E-Commerce Technology*, pages 149–156. IEEE, 2008.

[12] S. Dustdar and K. Bhattacharya. The social compute unit. *Internet Computing, IEEE*, 15(3):64–69, 2011.

[13] D. Jordan et al. Web Services business Process Execution Language (WS-BPEL) 2.0. *OASIS Standard*, 11, 2007.

[14] M. Kloppmann et al. WS-BPEL extension for people–bpel4people. *Joint white paper, IBM and SAP*, 2005.

[15] A. Agrawal et al. Web Services Human Task (WS-HumanTask), version 1.0. 2007.

[16] GR Gangadharan and V. D'Andrea. Service licensing: conceptualization, formalization, and expression. *Service Oriented Computing and Applications*, 5(1):37–59, 2011.

[17] S. Ran. A model for web services discovery with qos. *ACM Sigecom exchanges*, 4(1):1–10, 2003.

[18] M. Proctor, M. Neale, P. Lin, and M. Frandsen. Drools documentation. http://www.jboss.org/drools/documentation.html. [Online; accessed February-2013].

[19] B. Sengupta, A. Jain, K. Bhattacharya, H.L. Truong, and S. Dustdar. Who do you call? problem resolution through social compute units. In *Proceedings of the 10th International Conference on Service Oriented Computing (ICSOC)*, pages 48–62. Springer, 2012.

[2] http://www.informatik.tuwien.ac.at/teaching/phdschool