Model-Driven Integration for a Service Placement Optimizer in a Sustainable Cloud of Clouds

Junichi Suzuki *, Dung H. Phan*, Masatoshi Higuchi†, Yuji Yamano†, Katsuya Oba†
*Department of Computer Science
University of Massachusetts, Boston, Boston, MA 02125, U.S.A.
Email: {jxs, phdung}@cs.umb.edu
†OGIS International, Inc.
San Mateo, CA 94402, U.S.A.
Email: {higuchi, yyamano, oba}@ogis-international.com

***Abstract*—"Cloud of clouds" (or federated cloud) is an emerging style of software deployment and execution to interoperate application services and data across geographically-distributed clouds. One of key research issues to realize this notion is inter-cloud integration. This paper proposes a model-driven integration (MDI) framework for federated clouds. The proposed MDI framework consists of (1) a metamodel to graphically define inter-cloud integration models using Enterprise Integration Patterns (EIPs) and Cloud Computing Patterns (CCPs) and (2) an MDI tool that accepts an integration model defined with the proposed metamodel and transforms it to integration code (e.g., program code and deployment configurations). This paper presents the design of the proposed MDI framework and describes an integration case study to build a bio-inspired optimization engine for dynamic service placement in a sustainable cloud of clouds.**

***Index Terms*—Cloud of clouds, federated clouds, model-driven system integration and sustainable clouds**

I. Introduction

Cloud computing is a paradigm that realizes the vision of "computing as a utility" by leveraging Internet data centers (IDCs), each of which manages a pool of computing, storage and networking resources (e.g., CPU cores, databases and network bandwidth) [1], [2]. A cloud computing environment (or simply *cloud*) is an application deployment and execution platform that operates on one or more IDCs. (Fig. 1). It virtualizes resources available on IDCs and provides them to applications on an on-demand basis.

As shown in Fig. 1, two major types of cloud users are application providers and application users. Providers develop their applications and upload/deploy them to a cloud(s) so that users can access them via the Internet. Providers lease virtualized resources in a "pay-per-use" manner. Therefore, they pay higher resource utilization fees as their applications process higher demands and consume more resources.

Currently, there exist three primary types of clouds [2]:

- Public cloud: Operated by a cloud provider and used/shared by multiple application providers.
- Private cloud: Dedicated to a particular application provider and not shared with others. May be operated by an application provider or a third-party cloud provider.
- Community cloud: Dedicated to a particular community (e.g., government and m life science communities) and shared by application providers in the community.

As the variety of cloud types, offerings and use cases have been expanding, the notion of "cloud of clouds" (or federated cloud, or InterCloud) is conceived [3], [4]. It is structured and operated as a network (or federation) of geographically-distributed clouds for extended resource utilization, workload distribution, fault tolerance, cost effective (e.g., energy efficient) service/data placement and avoidance of "lock-in" to particular cloud providers. Fig. 2 shows an example federation of clouds that on-premise applications interact with.

In order to make the notion of "cloud of clouds" a reality, there are several research issues to address. A key research issue is *inter-cloud integration* [4]. This issue introduces a new type of cloud users, cloud integrators, who help application providers interoperate application services and data with each other across clouds (Fig. 1). Other issues include quality-of-service (QoS) monitoring, service/data migration, security and trust, billing, and governance across clouds [3], [4].

This paper investigates a model-driven integration (MDI) framework for federated clouds. The MDI framework consists of (1) a metamodel to graphically define integration models for federated clouds and (2) an MDI tool that accepts an integration model defined with the proposed metamodel and transforms it to integration code (e.g., program code and deployment configurations). The proposed metamodel allows cloud integrators to describe and maintain inter-cloud integration solutions as graphical models using Enterprise Integration Patterns (EIPs) [5] and Cloud Computing Patterns (CCPs) [6]. With the proposed EIP-CCP metamodel, inter-cloud integration solutions can be modeled at a high-level of abstraction without depending on any particular low-level implementation, deployment and integration technologies. The proposed MDI tool transforms high-level EIP-CCP models into low-level integration code for integration middleware such as Enterprise Service Bus (ESB).

This paper presents the design of the proposed MDI framework and describes how it is used in model-driven integration for a cloud of clouds. This paper demonstrates a case study that integrates geographically-distributed clouds and builds an optimization engine that leverages evolutionary algorithms to dynamically adjust service placement across clouds with respect to the sustainability and performance applications.

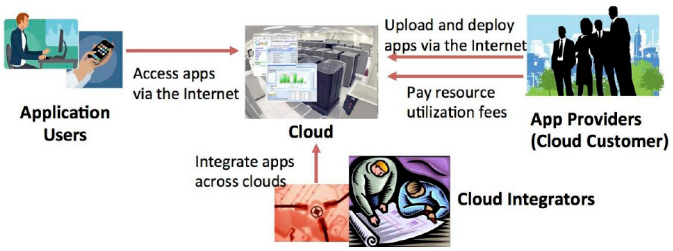


Fig. 1. A Cloud and its Users

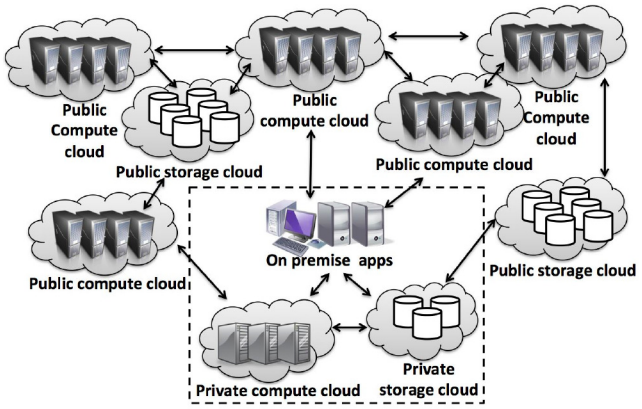


Fig. 2. An Example Cloud of Clouds

II. The Proposed MDI Framework

This section overviews Enterprise Integration Patterns (EIPs) and Cloud Computing Patterns (CCPs) and describes the proposed MDI framework.

A. Enterprise Integration Patterns (EIPs) and Cloud Computing Patterns (CCPs)

Enterprise Integration Patterns (EIPs) are architectural patterns to specify system integration solutions based on messaging architectures [5]. They capture best practices in system integration and provide a common integration terminology and modeling notation. Cloud Computing Patterns (CCPs) are architectural patterns to describe cloud types, cloud service models, cloud offerings and cloud application architectures in an abstract form [6].

Figs. 3 and 4 show some of EIPs and CCPs, respectively. Fig. 5 depicts an example integration model using two EIPs (*Message Channel* and *Message*) and two CCPs (*Public Cloud* and *IaaS*). This example model describes that four services ("Green Monster" and three cloud monitors) run on four different IaaS (Infrastructure-as-a-Service) clouds and interact with each other across the clouds. In this example, Green Monster sends an inquiry message to each cloud monitor through a message channel, and each cloud monitor returns the current health information of the local cloud through another channel. The health information includes the number of services running on the cloud, the number of requests placed on each service, air temperature in the cloud and renewable energy consumption of the cloud. Fig. 6 shows another example model that specifies a similar integration logic

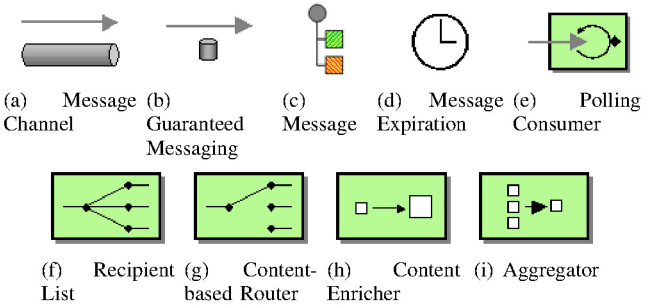


Fig. 3. Enterprise Integration Patterns (EIPs)

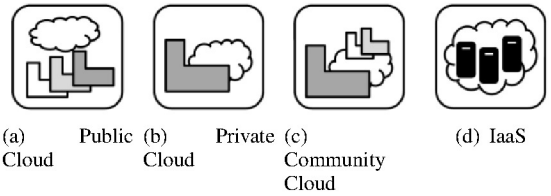


Fig. 4. Cloud Computing Patterns (CCPs)

to the logic in Fig. 5 with two EIPs: *Polling Consumer* and *Message*. In this example, Green Monster periodically polls over cloud monitors to collect the current health information.

B. The Architecture of the Proposed MDI Framework

The proposed model-driven integration (MDI) framework consists of an EIP-CCP metamodel and an MDI tool supporting the metamodel. The proposed metamodel defines EIPs and CCPs with the meta-meta model in the Generic Modeling Environment (GME) [7][1] (Fig. 7). It provides a visual and intuitive abstraction to model the architectures of inter-cloud integration solutions. Given the EIP-CCP metamodel, the proposed MDI tool allows cloud integrators to graphically

[1]http://www.isis.vanderbilt.edu/projects/gme/

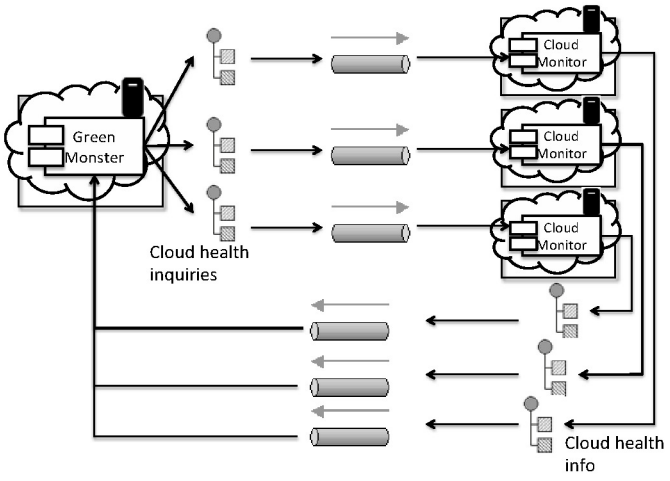


Fig. 5. An Example Integration Model with *Message Channel*

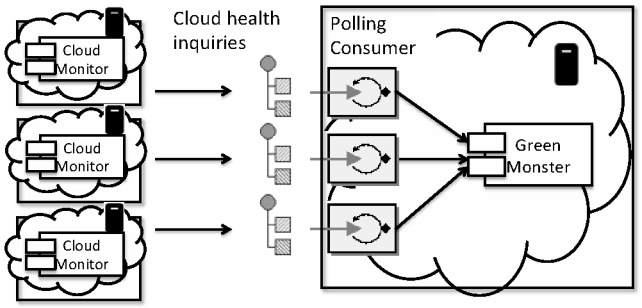


Fig. 6. An Example Integration Model with *Polling Consumer*

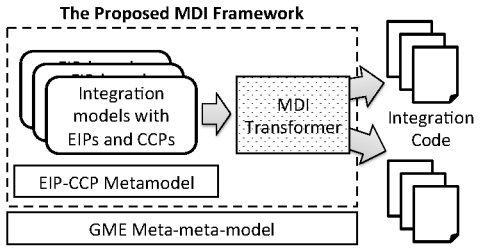


Fig. 7. The Architecture of the Proposed MDI Framework

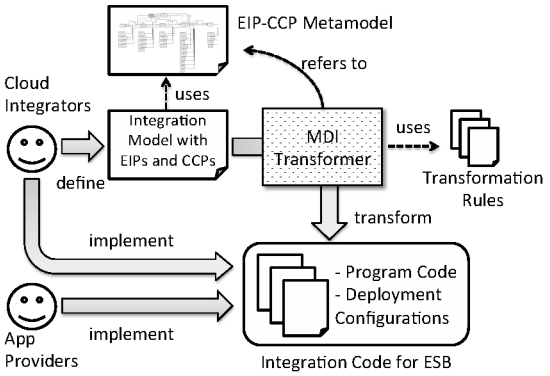


Fig. 8. Model-to-Code Transformation

describe and maintain inter-cloud integration models without relying on low-level implementation, deployment and integration technologies such as programming languages, transport protocols, security configurations and integration middleware (e.g., Enterprise Service Bus (ESB)). The proposed MDI tool also transforms EIP-CCP models to integration code (e.g., program code and deployment configurations) for ESB middleware.

Fig. 8 presents the process of defining and transforming EIP-CCP models with the proposed MDI framework. Cloud integrators define an integration model with EIPs and CCPs. The proposed MDI framework accepts an EIP-CCP model, verifies it against the proposed EIP-CCP metamodel and transforms it to a skeleton of configuration code (program code and deployment configurations) for ESB middleware. Cloud integrators and application providers complete the generated skeleton code to be a compilable and deployable form by adding method bodies (i.e., behavioral code) to each application service and specifying various ESB-related parameters (e.g., transport protocols, messaging timeout and polling interval).

The proposed MDI framework allows EIP-CCP models to be portable and reusable across different implementation, deployment and integration technologies. For example, EIP-CCP models can be independent across Java, C#, HTTP, REST, JMS, Mule ESB[2] and Microsoft BizTalk ESB. This means that single EIP-CCP model can be mapped to different sets of implementation, deployment and integration technologies simultaneously (Fig. 7). The proposed MDI framework allows EIP-CCP models to be intact in switching from a set of implementation, deployment and integration technologies (e.g., Java, JMS and Mule ESB) to another set (e.g., C#, HTTP and BizTalk ESB).

[2]http://www.mulesoft.org/

TABLE I
EIP-MULE MAPPING

EIP	Mule Deployment Configuration
Messaging Endpoints	
Message Endpoint	Transport-specific inbound/outbound endpoint
Polling Consumer	`<poll>`
Competing Consumer	Implemented by a custom outbound router.
Message Dispatcher	Implemented by a custom outbound router.
Selective Consumer	`<selective-consumer-router>`
Idempotent Receiver	`<idempotent-message-filter>`
Message Construction	
Message	Inherently supported by Mule.
Correlation Identifier	Implemented by the correlation ID header.
Message Sequence	Implemented by the sequence number header.
Message Expiration	Implemented by the expiration header, which specifies the expiration time and the maximum count for messaging retries.
Messaging Channels	
Message Channel	Inherently supported by Mule.
Dead Letter Channel	A combination of *Message Expiration* and *Content-Based Router*.
Guaranteed Delivery	Implemented by a custom outbound router.
Message Routing	
Pipes and Filters	Inherently supported by Mule.
Content-Based Router	`<choice>`
Message Filter	`<expression-filter>`
Recipient List	`<recipient-list>`
Splitter	`<collection-splitter>` `<splitter>`
Aggregator	`<collection-aggregator>`
Resequencer	`<resequencer>`
Scatter-Gather	A combination of *Recipient List* and *Aggregator*.
Message Transformation	
Content Enricher	`<enricher>`

C. EIP-to-Mule Transformation

The proposed MDI framework currently supports Java, HTTP and Mule ESB as the target implementation, deployment and integration technologies in its model-to-code transformation, although it does not depend on particular low-level technologies (Fig. 7). Table I summarizes some of the EIP-to-Mule transformation rules that the proposed MDI framework implements.

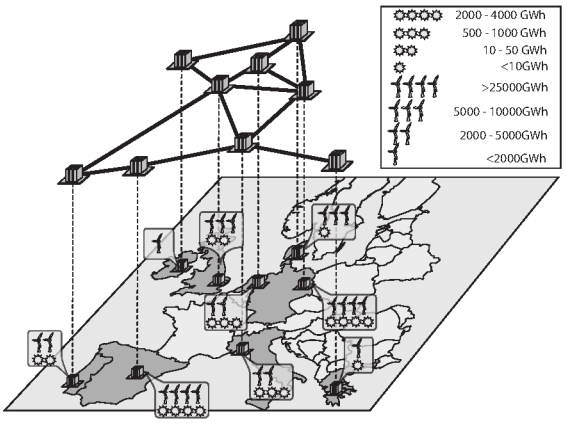


Fig. 9. A Federation of Clouds

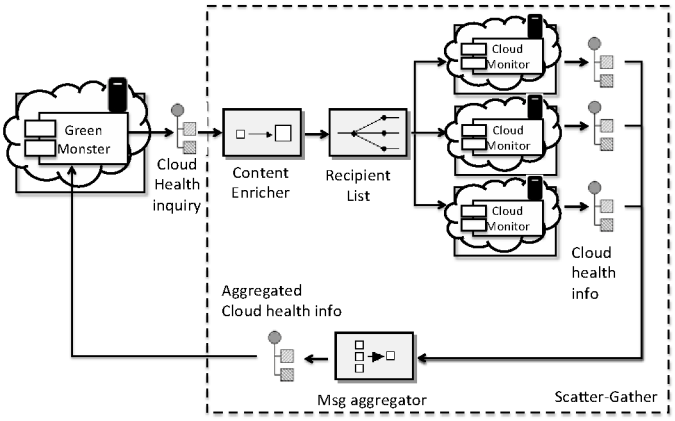


Fig. 10. An Integration Model with *Scatter-Gather*

III. Case Study: Model-Driven Integration for a Sustainable Cloud of Clouds

This section describes a case study where the proposed MDI framework aids to integrate and operate a cloud of clouds in a sustainable manner.

Since clouds have been increasing in scale and complexity, they are a significant source of energy consumption and CO_2 emission. They reportedly consumed 271.8 TWh worldwide in 2010, which accounted for 1.5% of the total electricity usage [8]. It is 3.8 times greater than the energy consumption of clouds in 2000. In 2007, the information and communication technology (ICT) industry produced 2% of global CO_2 emission, which is on par with the aviation industry[3]. Clouds were responsible for 23% of the ICT industry's emission.

In order to replace conventional fuels and reduce CO_2 emission, many countries actively pursue more renewable sources of energy through capital infrastructure projects or grid feed-in tariff incentive schemes. As a result, the capacity of renewable energy has increased exponentially in the past decade [9]. In 2010, renewable energy provided 312 GW worldwide, which accounted for 3% of global electricity generation[4].

Green Monster is an optimization engine to operate a federation of clouds in a sustainable manner [10]. It dynamically moves services (i.e., workload) to clouds with more desirable energy profiles while their maintaining performance (e.g., response time). It makes decisions of service migration and placement with an evolutionary multiobjective optimization algorithm (EMOA) that evolves a set of solution candidates (or *individuals*) under given constraints. Each individual represents a particular placement configuration of individual services. Green Monster considers conflicting optimization objectives: renewable energy consumption (RE), cooling energy consumption (CE) and user-to-service distance (USD). USD represents the response time of services to users. See [10] for full discussion on the EMOA in Green Monster.

This paper focuses on a model-driven integration between Green Monster and clouds. This integration case study assumes a federation of nine geographically-dispersed clouds in nine major European countries: Denmark, Germany, Greece, Ireland, Italy, Netherlands, Spain, UK and Portugal (Fig. 9). In order to meet this inter-cloud integration requirement, this case study uses the EIP-CCP model shown in Fig. 10. This model specifies the integration logic extended from those in Figs. 5 and 6. It uses three EIPs (*Content Enricher*, *Recipient List* and *Message Aggregator*) to form a compositete EIP *Scatter-Gather*.

In Fig. 10, Green Monster periodically sends out a health information inquiry to nine cloud monitors through a *Content Enricher* and a *Recipient List*. A *Content Enricher* obtains a list of endpoints representing individual cloud monitors and inject the list to a header of an incoming inquiry message. A *Recipient List* replicates an incoming message and distributes replicated messages to nine cloud monitors. (Figure 10 shows only three cloud monitors due to space limitation.) Each cloud monitor returns the current health information of the local cloud. A *Message Aggregator* aggregates nine incoming messages into a single message and transmits an aggregated message to Green Monster. A cloud health inquiry requests the following seven information to each cloud monitor:

- Services (IDs of the services) that are currently running on the cloud
- Request rate (# of requests) placed on each service
- Per-request CPU utilization for each service
- Availability of renewable energy for the cloud
- Indoor and outdoor temperature
- Per-request volume of data transmission for each service
- Capacity constraint of the cloud (the maximum number of service requests allowed for the cloud)

Listing 1 shows a part of the Mule deployment configuration that the proposed MDI framework generates from the EIP-CCP model in Fig. 10. As defined in Table. I, the proposed MDI framework transforms *Content Enricher*, *Recipient List* and *Message Aggregator* to `<enricher>`, `<recipient-list>` and `<collection-aggregator>`, respectively.

This case study completes the generated Mule deployment

[3]http://www.gartner.com/it/page.jsp?id=530912

[4]19% if hydroelectricity is included

```
<mule ...>
  <http:endpoint name="CloudMonitor1" host=... port=... method="POST" exchange-pattern="one-way" />
  <http:endpoint name="CloudMonitor2" host=... port=... method="POST" exchange-pattern="one-way" />
  ...
  <flow name="Green Monster">
    <inbound-endpoint address=...  exchange-pattern="request-response"/>
    <component class="edu.umb.cs.gm.GreenMonster"/>

    <enricher target="#[header:recipients]">
        <component class="edu.umb.cs.gm.Enricher"/>
    </enricher>

    <recipient-list enableCorrelation="ALWAYS" evaluator="header" expression= "recipients"/>
  </flow>

  <flow name=" Cloud Monitor 1 - Green Monster">
    <inbound-endpoint address=...  exchange-pattern="request-response"/>
      <object-to-byte-array-transformer />
      <component class="edu.umb.cs.gm.CloudMonitor1"/>
      <message-properties-transformer scope="outbound">
        <add-message-property key="MULE_CORRELATION_ID" value=... />
        <add-message-property key="MULE_CORRELATION_GROUP_SIZE " value="9"/>
      </message-properties-transformer>
    <http:outbound-endpoint host=... port=... path=""  method="POST" exchange-pattern="one-way" />
  </flow>

  <flow name="Aggregator - Green Monster">
    <inbound-endpoint address=...  exchange-pattern="request-response" />
    <object-to-byte-array-transformer />
    <collection-aggregator  timeout=... failOnTimeout="false"/>
    <component class="edu.umb.cs.gm.GreenMonster"/>
  </flow>
  ...
</mule>
```

Listing 1. Mule Deployment Configuration

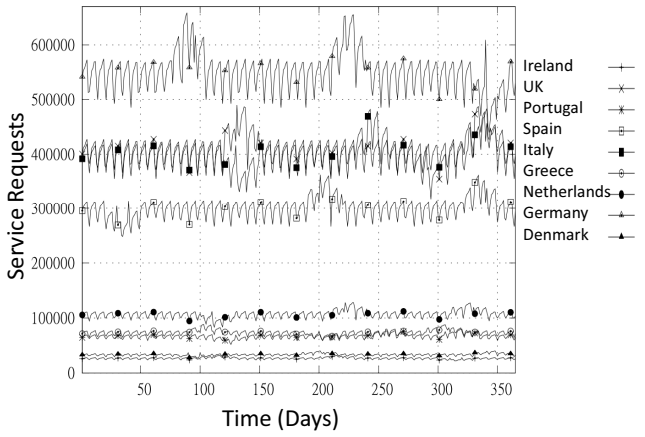


Fig. 11. Daily Service Request Rates

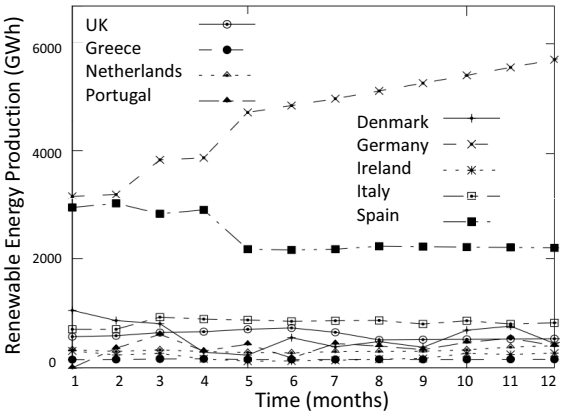


Fig. 12. Renewable Energy Production

configuration, integrates clouds with the completed deployment configuration and evaluates an integrated cloud of clouds through a preliminary experiment. This experiment deploys 10 services (Green Monster and nine cloud monitors) on 10 hosts that are accessible through the HTTP transport. In this experiment, each host emulates a cloud environment that operates a varying number of servers (8–200) and application services (16–400). Hosts are connected with an overlay topology in line with the European Optical Network (Fig. 9). Three types of services are emulated to run on clouds: data, voice and video services. Each host is supplied a set of cloud emulation data. Fig. 11 shows the daily service request rates emulated on different clouds. The average total rate is two million requests per day. The dynamic changes in the request rates are configured by adapting the traffic trace in Akamai's data centers [11][5]. The rates are configured across clouds in proportion to the populations of their host countries. In each cloud, requests are evenly distributed to all emulated services. For the temperature variations in each cloud, this experiment uses the data from the European Climate Assessment & Dataset project, which records real temperature data in Europe[6]. Fig. 12 shows the total renewable energy production in the host country of each cloud from January 2007 to December 2009. This data is produced with the data available from the International Energy

[5]In order to represent long-term fluctuations in request rates, this evaluation study adds a number of randomly distributed surges and falls on Akamai's short-term trace data.

[6]http://eca.knmi.nl

Agency (IEA)[7].

Given the aforementioned experimental configurations, Green Monster transmits a daily health inquiry to each cloud monitor and runs its EMOA every other week throughout 12 emulated months. In order to evaluate the performance of Green Monster, it is compared with the following two benchmark algorithms:

- Static placement: Randomly-selected two services are placed on each server at the beginning of an experiment. They do not dynamically migrate during an experiment.
- Random placement: Services dynamically migrate at random every other week while satisfying a given capacity constraint for each cloud.

Fig. 13 shows how Green Monster and two benchmark algorithms yield objective values (RE, CE and USD values) during emulated months. These results illustrate that Green Monster successfully migrates services according to dynamic changes in the health status of individual clouds and yields superior performance than two benchmark algorithms.

This case study verifies that the proposed MDI framework successfully aids to integrate and operate a cloud of clouds intuitively.

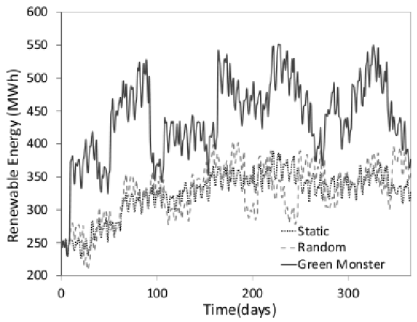
IV. Conclusions

This paper proposes and describes an MDI framework that consists of (1) a metamodel to graphically define integration models for federated clouds using EIPs and CCPs and (2) an MDI tool that transforms EIP-CCP models to integration code for ESB middleware. This paper also reports an integration case study to build a service placement optimizer for a sustainable cloud of clouds.

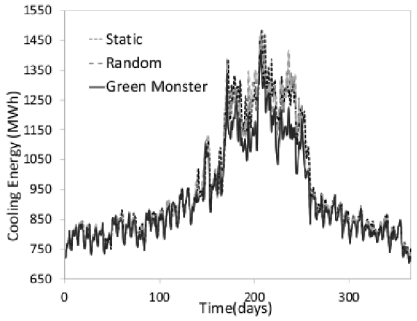
Future work include extending the proposed MDI framework to implement transformations from EIP-CCP models to other ESB middleware than Mule. Extended case study experiments are planned as well.

V. Acknowledgements

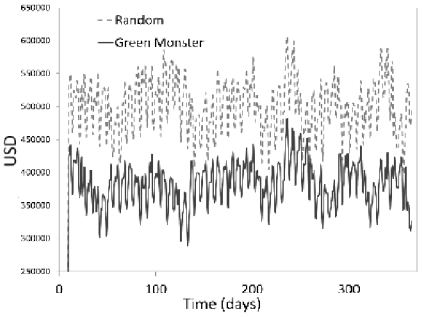
The authors of this paper thank Priyanka Das, Manali Kulkarni and Mayuri Mangireddy for their programming contributions. Special thanks go to Raymond Carroll, Sasitharan Balasubramaniam and William Donnelly for sharing experimental data used in this paper.



(a) Renewable Energy Consumption (RE)



(b) Cooling Energy Consumption (CE)



(c) User-to-Service Distance (USD)

Fig. 13. Objective Values

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, EECS Dept., Tech. Rep., 2009.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Tech. Rep., 2011.

[3] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow, "Blueprint for the intercloud - protocols and formats for cloud computing interoperability," in *Proc. Int'l Conference on Internet and Web Applications and Services*, 2009.

[4] R. Buyya, R. Ranjan, and R. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Proc. Int'l Conference on Algorithms and Architectures for Parallel Processing*, 2010.

[5] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison Wesley, 2003.

[6] C. Fehling, F. Leymann, J. Rutschlin, and D. Schumm, "Pattern-based development and management of cloud applications," *Future Internet*, vol. 4, no. 1, 2012.

[7] M. M. Ledeczi, A. and, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The generic modeling environment," in *IEEE Int'l Workshop on Intelligent Signal Proc.*, 2001.

[8] J. Koomey, "Growth in data center electricity use 2005 to 2010," *Analytics Press*, 2011.

[9] Renewable Energy Policy Network for the 21st Century (REN21), *Renewables 2011: Global Status Report*, 2011.

[10] D. H. Phan, J. Suzuki, R. Carroll, S. Balasubramaniam, W. Donnelly, and D. Botvich, "Evolutionary multiobjective optimization for green clouds," in *Proc. of ACM Genetic and Evol. Computat. Conference*, 2012.

[11] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," in *Proc. ACM SIGCOMM Conference*, 2009.

[7] http://www.iea.org/stats/surveys/elec_archives.asp