CloudMap: A Visual Notation for Representing and Managing Cloud Resources

Denis Weerasiri[1(✉)], Moshe Chai Barukh[1], Boualem Benatallah[1], and Cao Jian[2]

[1] University of New South Wales, Sydney, Australia
{denisw,mosheb,boualem}@cse.unsw.edu.au
[2] Shanghai Jiaotong University, Shanghai, China
cao-jian@cs.sjtu.edu.cn

Abstract. With the vast proliferation of cloud computing technologies, DevOps are inevitably faced with managing large amounts of complex cloud resource configurations. This involves being able to proficiently understand and analyze cloud resource attributes and relationships, and make decisions on demand. However, a majority of cloud tools encode resource descriptions and monitoring and control scripts in tedious textual formats. This presents complex and overwhelming challenges for DevOps to manually read, and iteratively build a mental representation especially when it involves a large number of cloud resources. To alleviate these frustrations we propose a model-driven notation to visually represent, monitor and control cloud resource configurations; managed underneath by existing cloud resource orchestration tools such as *Docker*. We propose a *mindmap*-based interface and set of visualization patterns. We have employed an extensive user-study to base design decisions, and validate our work based on experimentation with real-world scenarios. The results show significant productivity and efficiency improvements.

Keywords: Cloud resource management · DevOps · Visual notations

1 Introduction

Cloud computing is rapidly evolving in public, private and hybrid cloud networks [1]. The many benefits include enabling virtualization capabilities as well as outsourcing strategies – the cloud will be a firm priority for productivity and economic development. It is estimated by 2016, growth in cloud computing will consume the bulk of IT spend[1]. There are however crucial gaps in the cloud-enabled endeavor [1]. Modern resource configuration management systems like *Puppet, Ubuntu Juju, Ansible, Amazon OpsWorks* and *Chef* provide scripting-based languages over cloud services [4]. This implies even sophisticated programmers and administrators are forced to understand different low-level cloud service Application Programming Interfaces (APIs), command-line syntax, and

[1] http://www.gartner.com/newsroom/id/2613015.

S. Nurcan et al. (Eds.): CAiSE 2016, LNCS 9694, pp. 427–443, 2016.
DOI: 10.1007/978-3-319-39696-5_26

programming constructs, to create and maintain complex cloud resource configurations. Moreover, this problem worsens as the variety of cloud services and the variations of application resource requirements and constraints increase.

Inevitably, typical cloud-based organizations are finding it difficult to productively utilize their very large repositories ladened with textual cloud resource description and management artifacts, [1]. For example, simple management tasks commonly involve: analyzing resource descriptions; understanding the inter-relationships between resources; and aggregating monitoring data. However, until now DevOps (i.e., developers and operation personnel who are collectively involved in designing, developing, deploying and managing cloud applications) are required to manually and iteratively read several low-level files and use command-line tools to extract monitoring information. In fact, it has been confirmed that DevOps dedicate the majority of their time to understand existing artifacts instead of creating new ones, updating and/or testing them [6].

To overcome these challenges, we present *CloudMap*. Leveraging the old-age dictum of *"a picture tells a thousand words!"*, we develop visual notations to simplify representing and managing cloud resources. We argue this novel approach will enable DevOps to invest more on creating, configuring and managing cloud resources, instead of the frustrations and time spent to understand them. Since we are at the foundational stage, we have specialized our framework to *Docker* [10]; albeit in future it can easily be extended with other orchestration tools (e.g., *JuJu* or *Ansible*). *Docker* is an open-source and widely-praised industry standard initiative. Its Container-based virtualization technique offers a lightweight and portable resource isolation alternative to Virtual Machines (VMs). This technique emerged to simplify and accelerate the configuration and management of cloud resources. More specifically, for composite service-based cloud resources that depend on multiple service middleware for their operations, container-based virtualizations enable accelerated and efficient deployment of optimally configured, scalable and lightweight middleware instances. However, since to the best of our knowledge current tools merely leverage textual resource representations, it does not do justice to improving the productivity and efficiency of DevOps. Accordingly, this paper makes the following main contributions:

Expert User-Study. Based on an extensive survey of 21 participants (system administrators and software engineers with 3–10 years experience), we discover gaps and challenges in current solutions. We form a strong understanding of the requirements, and derive key design decisions for our novel solution.

Visual Notation for Representing & Managing Cloud Resources. We formulate the necessary notational constructs (i.e., *Entities* and *Links*) and define the semantics for each. We also propose novel auxiliary features called *Probes* and *Control Actions*, that can be "tagged" to entities.

Cloud Visualization Patterns. We identify common visualization patterns for cloud resource configuration. Existing architectural patterns are high-level and mostly suitable for solution architects or IT directors [5]. In contrast, we believe DevOps require more fine-grained visual abstractions for understanding, navigating, monitoring and controlling complex cloud resources. Resultantly, we present three patterns and describe their benefits via practical scenarios.

The rest of this paper is organized as follows: In Sect. 2, we position this work with respect to our previous work; and explain how our visual notation is applied within the cloud resource lifecycle. Furthermore, we present the results of our user-study. In Sect. 3, we detail our visual notation and its semantics, and present three organizational patterns. We employ a mind-map interface, and illustrate how they could be used over various use-cases which span across selection, configuration, deployment, monitoring and controlling of cloud resources. In Sects. 4, we present our implementation and GUI; evaluation in Sect. 5; then related-work, and conclusions in Sect. 6.

2 Background

2.1 Motivating Example

Cloud resources management typically involves: (i) An initially ***sequential*** stage of consisting of: *Select*, *Configure* and *Deploy*; (ii) Followed by an ***iterative*** phase consisting of: *Monitor* and *Control*.

As a running example, consider the 3-tier system illustrated in Fig. 1. We begin by *selecting* the required resources. In this case, business logic is executed using *Business Process Execution Language (BPEL)*, with state data stored on a *MySQL DB*. For scaling purposes, we introduce a *Nginx Load Balancer* that propagates requests to a cluster of *Apache Orchestration Director Engine (ODE) Servers*. To *configure* and *deploy*, DevOps determine the relationships between components and write configuration and deployment scripts, that describes the attributes (e.g., no. of BPEL engines, CPU allocation). Subsequently, DevOps also collect and analyze events to *monitor* and apply *control* actions if necessary.

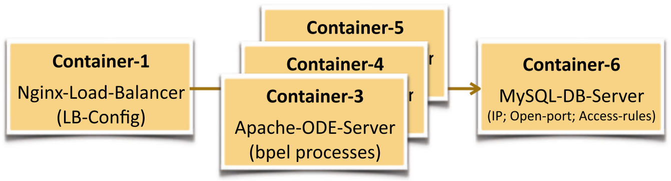
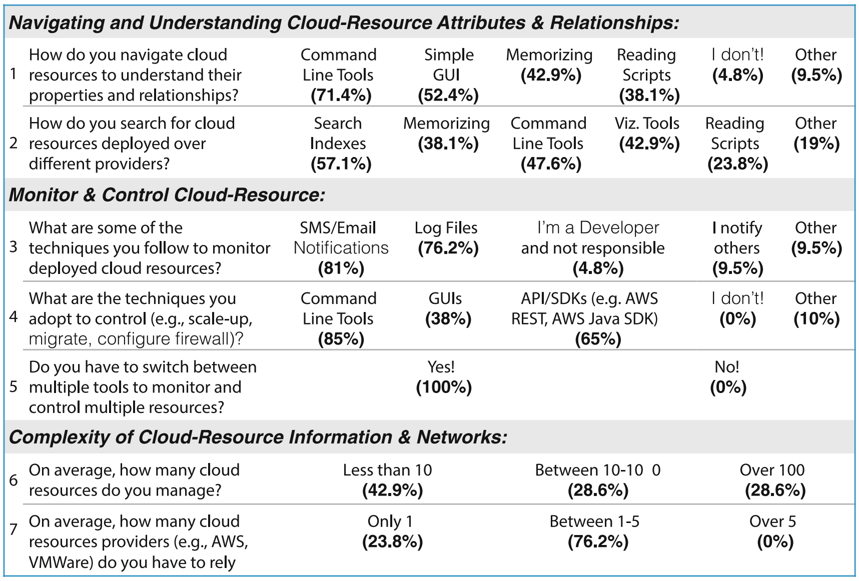


Fig. 1. Resource diagram of the typical 3-tier (BPEL-based) application

2.2 Requirements for Cloud Resources Visual Notation

To articulate the requirements of our novel solution, we sought to gain insight about the gaps and challenges in current strategies. We conducted a user-study over 21 experts with 3–10 years of working experience: 9 server administrators and 12 software engineers (e.g., cloud-based application developers).

Research Questions and Results. Three main areas of investigation were sought. Techniques to: (a) navigate and understand cloud resource attributes and relationships; and (b) monitor and control cloud resources. In addition, (c) we sought to discover how the above increased in complexity when the number of managed resources increased. Responses were automatically recorded in an online spreadsheet, and the raw results summarized in Fig. 2.



Navigating and Understanding Cloud-Resource Attributes & Relationships:							
1	How do you navigate cloud resources to understand their properties and relationships?	Command Line Tools **(71.4%)**	Simple GUI **(52.4%)**	Memorizing **(42.9%)**	Reading Scripts **(38.1%)**	I don't! **(4.8%)**	Other **(9.5%)**
2	How do you search for cloud resources deployed over different providers?	Search Indexes **(57.1%)**	Memorizing **(38.1%)**	Command Line Tools **(47.6%)**	Viz. Tools **(42.9%)**	Reading Scripts **(23.8%)**	Other **(19%)**
Monitor & Control Cloud-Resource:							
3	What are some of the techniques you follow to monitor deployed cloud resources?	SMS/Email Notifications **(81%)**	Log Files **(76.2%)**	I'm a Developer and not responsible **(4.8%)**		I notify others **(9.5%)**	Other **(9.5%)**
4	What are the techniques you adopt to control (e.g., scale-up, migrate, configure firewall)?	Command Line Tools **(85%)**	GUIs **(38%)**	API/SDKs (e.g. AWS REST, AWS Java SDK) **(65%)**		I don't! **(0%)**	Other **(10%)**
5	Do you have to switch between multiple tools to monitor and control multiple resources?		Yes! **(100%)**			No! **(0%)**	
Complexity of Cloud-Resource Information & Networks:							
6	On average, how many cloud resources do you manage?	Less than 10 **(42.9%)**		Between 10-10 0 **(28.6%)**		Over 100 **(28.6%)**	
7	On average, how many cloud resources providers (e.g., AWS, VMWare) do you have to rely	Only 1 **(23.8%)**		Between 1-5 **(76.2%)**		Over 5 **(0%)**	

Fig. 2. Survey questions and results

Survey Analysis. While a more detailed discussion of the analysis is largely outside the scope of this paper, we summarize our findings below and derive the fundamental requirements for our proposed visual notation. This was also done in conjunction with our own investigation and systematic literature review[2] on a range of cloud resource orchestration tools and techniques [2,10,11].

- The majority (71.4 %) of DevOps rely on command line tools to navigate cloud resources and discover their properties and relationships. While nearly half (38.1 %) has resorted to manual and error-prone techniques (e.g., reading and memorizing configuration files). This was primarily due to that fact most of simple GUIs were limited to resource *attributes* only. Accordingly, we have sought to extend visual techniques for displaying not only attributes, but types of resources and important directional links between resources.
- A large majority (76.2–81%) admitted they rely on log files and notifications (e.g., emails, SMS) to *monitor* cloud resources. While regarding *control* and

[2] Internal Report prepared and impending Journal to be submitted: *"D. Weerasiri et al. A taxonomy and survey of cloud resource orchestration techniques"*.

(re-)configuration, most DevOps use command-line tools, APIs and/or SDKs (65–85%). Participants also complained about the inconvenience of switching between multiple tools to monitor and control cloud resources. Accordingly, our work sought to integrate visual *probe* and *control* elements to facilitate not only monitoring, but (re-)actions and control of cloud resources.

– Overall, the need for simplifying all aspects of cloud resource management (i.e., navigation and understanding configured resources; monitoring and control), was widely championed by participants. This was because a wide majority (71.5 %) of DevOps manage large numbers (approx. <100) of cloud resources. Matters also exasperate when many (76.3 %) have to simultaneously manage multiple cloud resource providers (e.g., AWS, VMWare).

3 *CloudMap:* Visual Notation for Cloud Resource Management

CloudMap offers a refreshing "visual" attempt at simplifying the way DevOps can navigate and understand cloud resource configurations, as well as monitor and control such resources. The concepts of *CloudMap* are associated with both

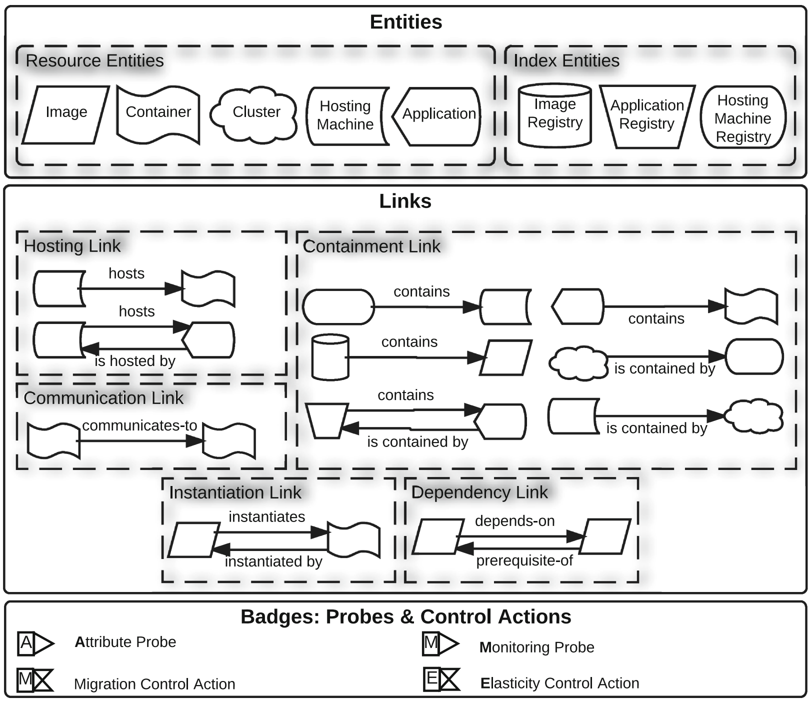
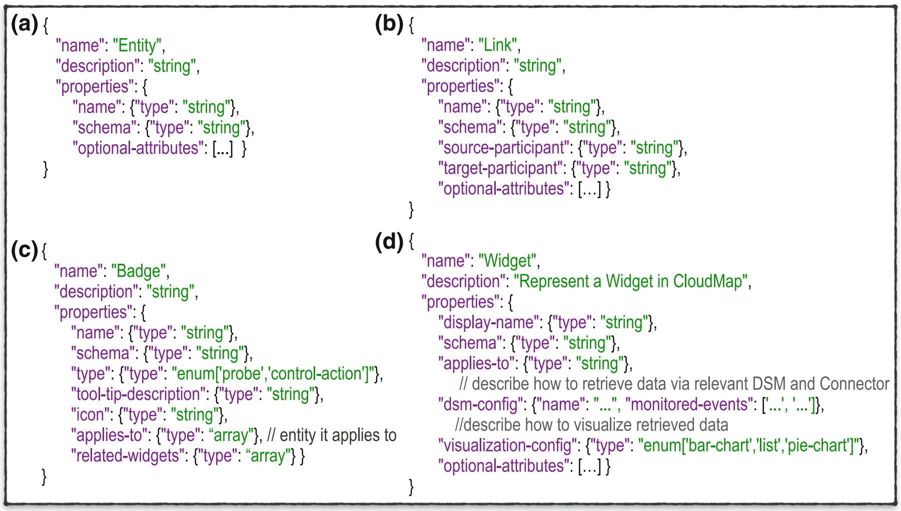


Fig. 3. *CloudMap* visual notations

a visual notation and an underlying textual JSON syntax. The textual syntax provides the context in which visual primates can be specified and executed.

The constructs of the notation are specified as the following: (i) ***Structural model*** represents primitive cloud resource entities and their attributes. Attributes have string name with one or a set of string values. (ii) ***Navigation Model*** represents the topology of links between entities. Links are directional with a single string label. The set of valid links are domain-specific; (we will explain in Sect. 4 how *CloudMap* depends upon our previous work *CloudBase* to determine domain context). (iii) ***Badges*** are an auxiliary feature to the fundamental constructs mentioned above. A badge represents a special entity that may be "tagged" to another entity. Visually, a badge is realized as a single or set of widgets, which may be used to monitor and/or control tagged entities. The data model of a badge specifies which entity-type it applies to.

Figure 3 illustrates the graphical notation, while Fig. 4 the syntactical schema of the underlying JSON model. Below we explain each of these constructs.



```
(a) {
      "name": "Entity",
      "description": "string",
      "properties": {
         "name": {"type": "string"},
         "schema": {"type": "string"},
         "optional-attributes": [...] }
    }

(b) {
      "name": "Link",
      "description": "string",
      "properties": {
         "name": {"type": "string"},
         "schema": {"type": "string"},
         "source-participant": {"type": "string"},
         "target-participant": {"type": "string"},
         "optional-attributes": [...] }
    }

(c) {
      "name": "Badge",
      "description": "string",
      "properties": {
         "name": {"type": "string"},
         "schema": {"type": "string"},
         "type": {"type": "enum['probe','control-action']"},
         "tool-tip-description": {"type": "string"},
         "icon": {"type": "string"},
         "applies-to": {"type": "array"}, // entity it applies to
         "related-widgets": {"type": "array"} }
    }

(d) {
      "name": "Widget",
      "description": "Represent a Widget in CloudMap",
      "properties": {
         "display-name": {"type": "string"},
         "schema": {"type": "string"},
         "applies-to": {"type": "string"},
           // describe how to retrieve data via relevant DSM and Connector
         "dsm-config": {"name": "...", "monitored-events": ['...', '...']},
           //describe how to visualize retrieved data
         "visualization-config": {"type": "enum['bar-chart','list','pie-chart']"},
         "optional-attributes": [...] }
    }
```

Fig. 4. *CloudMap* syntactical schema of constructs

3.1 Structural Model: Entities

An *entity* is a single cloud resource, referred to as a *Resource Entity*; or collection thereof, referred to as a *Index Entity*. Syntactically, an entity as shown in Fig. 4(a) contains a string `name` and `description` and set of `properties`. The overall structure for each entity type is in accordance with the Domain-Specific `schema`[3].

[3] See our previous work: ftp://ftp.cse.unsw.edu.au/pub/doc/papers/UNSW/201514.pdf.

Resource Entities. We have identified 5 resource entities types: *(1) Containers* represents a virtualized software container (e.g., Linux and OpenVZ[4] containers) where an application or a component of an application (e.g., an Apache ODE Server installed on Ubuntu OS) is deployed. *(2) Hosting Machines* represents a computer system where *Containers* are hosted (e.g., Virtual Machine (VM) or a physical machine). *(3) Clusters* represents a set of *Hosting Machines.* This reduces the overhead of dynamically managing multiple machines. For example, the *Cluster* may automatically decide which *Hosting Machine* will be chosen to deploy the given container based on an optimization algorithm [7]. *(4) Applications* represents a logical entity that includes a collection of related *Containers.* Each *Container* constitutes a component of the *Application. (5) Images* represents the deployment description of a *Container*, that is fed to the runtime of the orchestration tool in order to instantiate the *Container.*

Index Entities. Similarly, we have identified 3 index entity types: *(1) Hosting Machine Registry* is a logical entity that contains a set of *Hosting Machines* and *Clusters. (2) Application Registry* represents a repository of *Applications.* DevOps organize and discover all deployed cloud applications within the Registry. *(3) Image Registry* represents a repository of *Images* where DevOps may organize, curate and share resource deployment knowledge.

3.2 Navigation Model: Links

The relationship between entities are represented as links and enable navigation. Syntactically, links have a string `name` and `description`. They also define the `source` and `target` participants. Additional `attributes` may also be defined.

We have identified 5 different groups of links, as categorized below: *(1) Communication Links* are defined between two *Containers* that interact or exchange data. For example, an Apache ODE (BPEL) server communicates with a MySQL database server, about a BPEL instances. *(2) Containment Link*s defines the hierarchical organization of entities. In practice, they may also be also used to simultaneously control a set of related resources (e.g., control actions on a parent automatically triggers actions on all children). The valid set of containment links are defined in the links' schema (also as illustrated in Fig. 3). *(3) Hosting Links* defines a relationship between a *Hosting Machine* and *Container* or an *Application.* For example, between a PHP application engine and a VM where the PHP application engine is deployed. *(4) Dependency Links* defines a relationship between two *Images*, where the attributes of a particular resource depend upon the other resource. *(5) Instantiation Links* defines a relationship between an *Image* and *Container*, when the latter may be produced from the former. For example, a *Conteiner* may instantiate an *Image.*

3.3 Badges: Probes and Control-Actions

A *badge* represents an auxiliary feature to the fundamental constructs of *entities* and *links.* Syntactically, they define a string `name` and `description`, and may

[4] http://openvz.org/.

only be applied to certain entity types, defined in the `applies-to` property. When *tagged* to some entity, they apply some behavioral function, depending on the `type` of badge: either a *probe* (used for monitoring) or *control action* (used for performing some action). Visually, when a badge is tagged to an entity, it renders a widget; a badge thus also contains a pointer to one or a set of `widgets`.

Probes. At present, we have developed 2 *probes*: *(1) Attribute Probe* displays the name and values of attributes for a given entity. For example, a VM in AWS-EC2 contains attributes about the number of CPU cores, storage and memory capacity, OS and access rules. *(2) Monitoring Probe* continuously monitor runtime data of deployed cloud resources. For example, if an *Application*, a *Container* or a *Hosting Machine* is tagged by the "monitoring probe", an appropriate widget becomes available to graphically analyze the underlying resource consumption statistics (e.g., memory usage, network I/O, CPU usage). Monitoring probes may also be attached on several resources at the same time, and the associated widget aggregates, summarizes and provides a visual technique to compare the performance of multiple cloud resources.

Control Actions. Actions can be both manual or automated. Manual actions are self-prompted by dragging specific badges onto entities. For example, *(1) Elasticity Control* applies to a *Container* or *Hosting Machine* to scale up or down (e.g., no. of CPUs or memory); and (2) *Migration Control* to migrate a container across VMs. Automated actions are inputed as ECA-rules (see Sect. 4), via a dedicated universal badge that renders a rule-input widget.

3.4 Visualization Patterns for Cloud Resource Configurations

We consolidate the above by identifying 3 organizational patterns commonly found within cloud resource management. We describe the benefits of these via practical scenarios and accompanying illustrations.

Image Map. An *Image Map* visualizes the recursive dependency of *Images* within a *Registry*. In general, cloud resource descriptions, deployment and/or

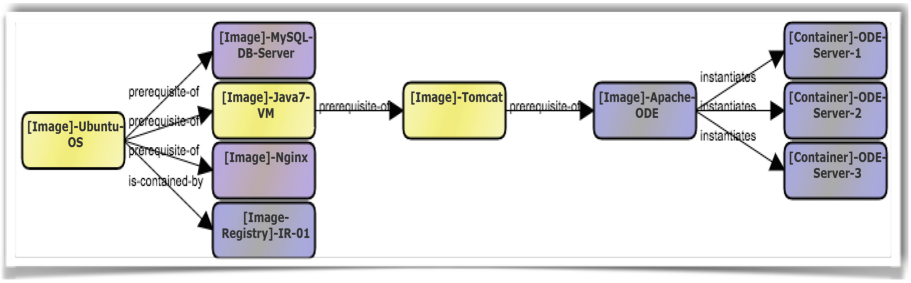


Fig. 5. Image Map

control scripts typically have inter-relationships among them. Understanding this is essential during deployment; it also assists to avoid errors in updating and creating new deployment artifacts. Ordinarily, DevOps would have to manually examine and comparing deployment artifacts to extract such inter-relationships.

Example. Figure 5 depicts an *Image Map* focused on "Ubuntu-OS". Consider we want to extend the "Apache-ODE" *Image* with an additional feature (e.g., BPEL4People). We are required to know all existing dependencies (e.g., Java-VM version) recursively up until the root Image. Thus the *Image Map* provides an indispensable visual technique to easily discover whether the "Apache-ODE" *Image* is dependent upon a particular version of "Java-VM" (e.g., Java7-VM). It is also useful as DevOps can identify, customize and reuse existing resources.

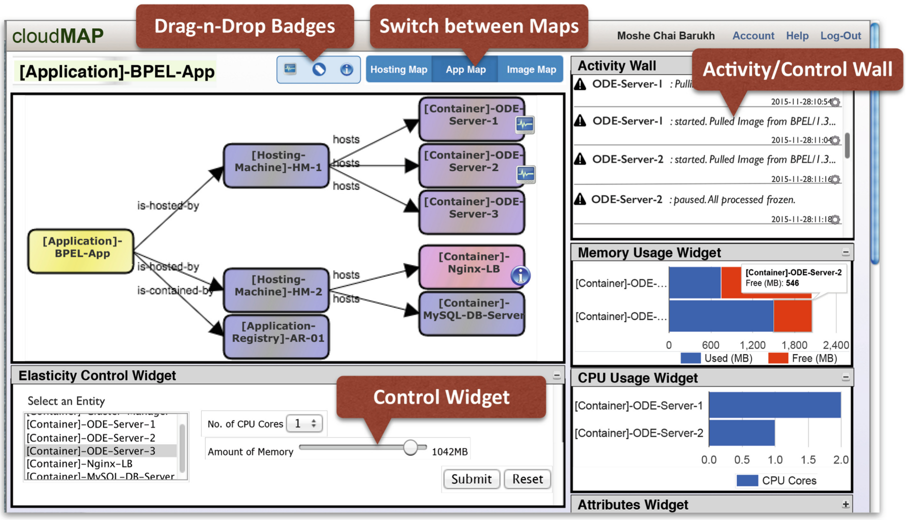
Application Map. An *Application Map* visualizes the organization and interaction of *Hosting Machines* and/or *Containers* of an *Application.* Deployed resources usually depend on other cloud resources to provide and consume services. Thus understanding their runtime interactions proves extremely important, particularly when applying modifications (e.g., reconfiguring, scaling, shutting down). This could otherwise lead to Service-Level-Agreement (SLA) violations, or catastrophic disruptions to the complete resource infrastructure[5].

Example. Figure 6 depicts an Application Map focused on the "BPEL-App". Consider a DevOp may wish to scale-up the "Apache-ODE-Server". This requires creating a new "Apache-ODE-Server" and the communication links with any related *Containers* (e.g., "MySQL-DB-Server" and "Nginx-LB"). DevOps hence need to understand: (a) what are the existing *Containers* of an *Application*; (b) how each are related to one another; and (c) what *Image* is to be used to instantiate the new *Container.* Using the *Application Map* in conjunction with the *Image Map*, DevOps can easily determine which image would be needed, and the related containers to setup the communication links.

Another use-case is understanding the communication links between *Containers.* Traditionally, this is achieved via command-line tools, which only returns details about a single container. DevOps would thus iteratively discover information about each *Container* to derive a global view. Command-line tools are also only suitable for sophisticated administrators. Monitoring details and control actions are represented in textual forms which are hard to memorize and understand compared to visual forms.

Example. To optimize the overall performance of an *Application* we may minimize data communication across different *Hosting Machines.* One such technique is to detect inter-communicating *Containers* and migrate into one *Hosting Machine* to reduce network latencies.

[5] http://aws.amazon.com/message/65648/.



ODE-Server-1 : Pulli
2015-11-28:10:54

ODE-Server-1 : started. Pulled Image from BPEL/1.3...
2015-11-28:11:04

ODE-Server-2 : started. Pulled Image from BPEL/1.3...
2015-11-28:11:16

ODE-Server-2 : paused. All processed frozen.
2015-11-28:11:18

Fig. 6. Application Map

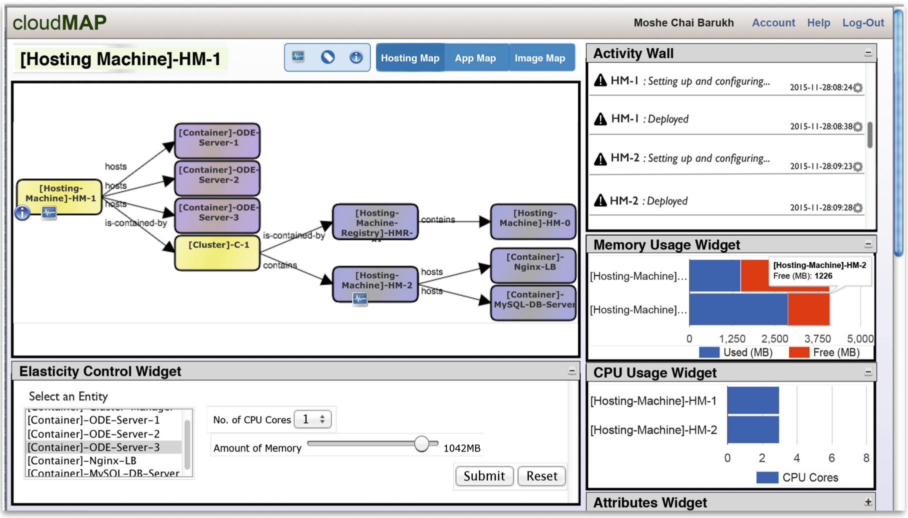
Hosting-Machine Map. A *Hosting-Machine Map* visualizes the organization of *Containers* and *Applications* within a specific *Hosting Machine*. This is useful for DevOps who manage a complete cloud environment; as opposed to the Application Map which only shows the Containers for a specific Application.

Example. Figure 7 shows the set of *Containers* deployed on a *Hosting Machine* "HM-1". DevOps are constantly responsible to check for optimization strategies: identifying under- or over- used machines. Suppose "HM1" can host a maximum of five *Containers*. We can use the map to determine there are only three running *Containers* (i.e., ODE-Server 1, 2 and 3). Thus it is possible to deploy two new *Containers* or migrate two existing ones. On the other hand, DevOps may delete *Hosting Machines* which are not currently hosting any *Containers*.

Furthermore, in conjunction with the *Monitoring Probe*, DevOps may observe data such as memory and storage utilization, as well as the existence of any exhausted machines. Actions may then be taken to avoid potential memory overflows and crashes of *Containers*. For instance, scaling-up the *Hosting Machines* to increase their underlying resources; or notify the owners to take any necessary actions (e.g., migrate *Containers* to a non-exhausted machines).

4 Implementation

We leverage our previous work *CloudBase* [14] to simplify the interactions between underlying orchestration tools. With our proposed *Domain Specific Model (DSM)*, high-level resource configurations can be supplied which are then automatically translated into their native language using *Connectors*. Layered above this as shown in Fig. 8, *CloudMap* implements: (i) An interactive mind-map visualization for navigating cloud resources; (ii) Detecting and displays



HM-1 : *Setting up and configuring...* 2015-11-28:08:24
HM-1 : *Deployed* 2015-11-28:08:38
HM-2 : *Setting up and configuring...* 2015-11-28:09:23
HM-2 : *Deployed* 2015-11-28:09:28

[Container]-ODE-Server-1
[Container]-ODE-Server-2
[Container]-ODE-Server-3
[Container]-Nginx-LB
[Container]-MySQL-DB-Server

No. of CPU Cores 1
Amount of Memory 1042MB

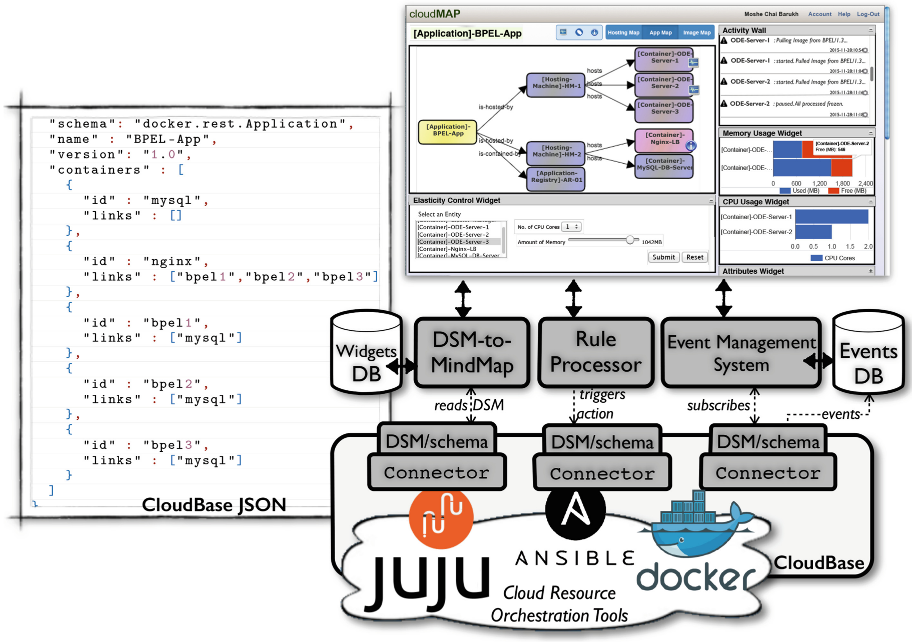
Fig. 7. Hosting Machine Map

events for monitoring; and (iii) allowing to perform both manual and automated actions.

Mind-Map Generation. The knowledge needed to generated the mind-maps are serialized from a supplied `CloudMap JSON` file. This is typically written by DevOps for a desired cloud configuration using the *CloudMap Notation* we presented in Sect. 3. As mentioned, constructs that pertain to some orchestration tool (e.g., *Docker*), must abide by the DSM's schema. For example, the *BPEL-App* entity abides by the `docker.rest.Application` schema, as shown in Fig. 8. When this is the case, behind the scenes the *CloudBase* engine is able to interpret complex and heterogeneous configurations and seamlessly connect to the underlying orchestration tool. This means, when a configuration file is written it is automatically translated into the low-level tool-specific language/API and deployed. The graphical mind-maps is rendered via the JS InfoVis Toolkit[6].

Event Management System. Upon loading a `CloudMap JSON` file, we also use *CloudBase* to determine the type of events that can be detected. For example, the *BPEL-App* includes events such as: `@Created`, `@Stopped`, `@Pasued`, `@Running`, etc. Thereby, the *Event Processor* component sets up the necessary, polling, processing and aggregating of event data. Once again, we leverage the `Connectors` implemented in *CloudBase* to help extract monitoring data from the low-level API (e.g., Docker Remote API[7]); and we assume access credentials are supplied in advance by the user. `Connectors` leverage Apache Camel[8] for event subscription. Events are then archived and indexed in a single MySQL

[6] http://philogb.github.io/jit/.
[7] http://docs.docker.com/reference/api/docker_remote_api/.
[8] http://camel.apache.org/.



```
"schema": "docker.rest.Application",
"name" : "BPEL-App",
"version": "1.0",
"containers" : [
  {
    "id" : "mysql",
    "links" : []
  },
  {
    "id" : "nginx",
    "links" : ["bpel1","bpel2","bpel3"]
  },
  {
    "id" : "bpel1",
    "links" : ["mysql"]
  },
  {
    "id" : "bpel2",
    "links" : ["mysql"]
  },
  {
    "id" : "bpel3",
    "links" : ["mysql"]
  }
]
```

Fig. 8. CloudMap system architecture

database table, the *Monitoring Events DB*. Each table entry includes the ID of the Resource, timestamp, data-type (e.g., CPU or memory usage) and data-value.

Rule Processor. To enable automation, DevOps may also supply simple reactive rules. For example, *if* `@Stopped` *then* `#notify`, which implies if the BPEL-App stopped perform some notification action. To greatly simplify the way rules can be defined, we reuse a simple rule-definition language adapted from our previous work [9]. In that previous work, we assumed a *"Knowledge-driven"* approach, which means APIs and their constituents (i.e., operations, input/output types) of the orchestration tools are loaded in a knowledge-base. This makes it possible to write high-level rule definitions and translate into concrete actions.

Activity/Control Wall. To enable interactivity, we have implemented a contextualized dashboard and control wall. For example, when either a *Probe* or *Control* badge is drag-n-dropped onto a mind map entity, an appropriate widget is displayed. Activity events are also posted. Badges may also be attached to multiple nodes to formulate an aggregated visualization. For example, Figs. 6 and 7 compares absolute memory consumption statistics of each *Container* and *Hosting Machine*. Similarly, control actions widgets allow DevOps to "manually" perform actions to modify the resource configurations. Widgets are implemented in HTML/JS and leverages Google Chart Library[9]; we assume the requisite widgets are pre-built and curated in the *Widgets Base*. Realtime updates to the widget is also achieved by triggers on the *Monitoring Events DB* that notify the affected widget when new event entries are received in the DB.

5 Evaluation

5.1 Experimental Setup

We conducted a user-study to evaluate the following hypotheses: ***H1,*** *CloudMap* increases the efficiency to accurately understand and navigate attributes and relationships of deployed cloud resources; ***H2,*** increases the efficiency to accurately perform monitor and control actions; and ***H3,*** the key features offered are useful and comprehensible. We measured *efficiency* as the time taken to complete the tasks; and *accuracy* was determined using a set of questions (see below). The total time to complete the tasks was until all questions were answered accurately.

Evaluation Task & Questionnaire. The task consisted of both a *practical* and *written* component. Written feedback was provided via a questionnaire divided into four main parts: (a) Background; (b) Functionality; (c) Usability; and (d) Insights and Improvements. The *Background* questions sought to discover the participants' familiarity with existing cloud resource orchestration techniques (i.e., Docker). The *Functionality* questions provided the necessary instructions and to determine the accuracy in completing the tasks. Questions[10] targeted different features and were related to the given task, such as understanding attributes; navigating relationships; and performing control actions.

Participant Selection & Grouping. Participants were sourced with diverse levels of technical expertise. For the sake of analysis, we classified a total of 12 participants into 2 main groups: (I) *Experts (7 participants)* with sophisticated understanding of cloud orchestration tools with 2–8 years of experience. And (II) *Generalists (5 participants)* who have average knowledge of cloud orchestration tool for day-to-day requirements, with around 1–5 years of experience.

5.2 Experiment Results and Analysis

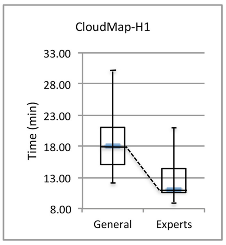
Evaluation of $\boldsymbol{H1}$ and $\boldsymbol{H2}$. The hypotheses $H1$ and $H2$ were evaluated based on the *time* taken to perform the tasks and provide *accurate* responses to the questionnaire. Alternatively, we sought to disprove the null hypotheses $H1_0$ and $H2_0$. Both hypotheses were examined by conducting a t-test with a probability threshold of 5 %, and assuming unequal variance.

As shown in Fig. 9, it was pleasantly surprising that even generalists demonstrated a significant increase in efficiency (and reduction in time). The comparative experiment focused on one third-party tool only, *Shipyard*[11]. Due to the

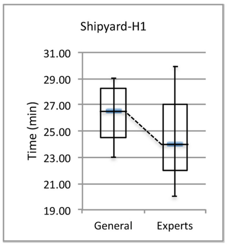
[9] http://developers.google.com/chart/.

[10] See mosheb.web.cse.unsw.edu.au/CloudMapQns.html for a complete list of questions.

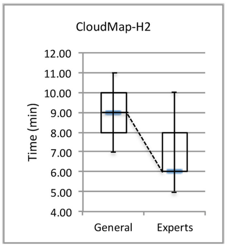
[11] https://shipyard-project.com/.



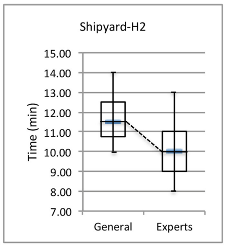
(a) **H1:**CloudMap



(b) **H1:**Shipyard



(c) **H2:**CloudMap



(d) **H2:**Shipyard

	H1		H2	
	CloudMap	Shipyard	CloudMap	Shipyard
Mean	15.58	25.33	7.83	10.89
Variance	37.17	11.5	3.606	3.611
Observations	12	9	12	9

	H1	H2
df	19	19
$p(T \neq t)$	0.0004	0.0017
t Critical two-tail	4.3063	3.6479
Reject null hypothesis	Yes	Yes

Fig. 9. Time results (grouped by expertise) to complete the tasks; and below t-test Results for H1 and H2

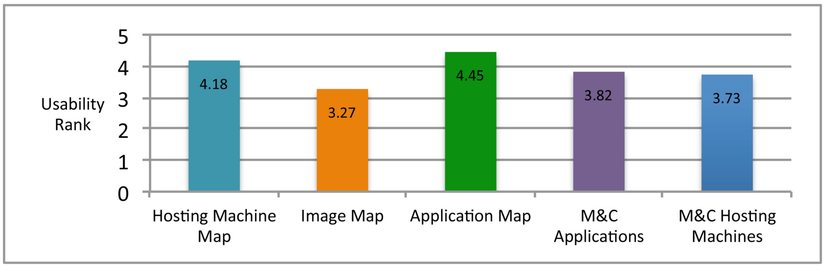


Fig. 10. Rate of usability of the main features of CloudMap

high number of existing cloud management tools, as well as project-based constraints, a more exhaustive comparative experiment was outside the scope. However, given the stark differences in times (means of 15.58 mins against 25.33 mins for $H1$; and means of 7.83 mins against 10.89 mins for $H2$), we postulate that it is unlikely to observe fundamental differences when comparing with any other tools similar to Shipyard. Accordingly, given our observations the likelihood of both $H1_0$ and $H2_0$ (equal mean modeling time) was around 5 %. Therefore, we could safely reject these null hypotheses, and imply the truth of $H1$ and $H2$.

Evaluation of *H*3. We evaluated this hypothesis through the *Usability* section of the questionnaire, and by asking participants to rate the usability for each feature (scale 0–5). We examined basic features such as the *Application*, *Image* and *Hosting Map*. As well as advanced features such as using badges and widgets to monitor and control (M&C) applications and hosting machines. We observed that the mean score for all features in Fig. 10 is above the neutral value of 3.

5.3 Discussion

Overall participants found that Mind-Map visualization a new but familiar concept. It was also impressive that *CloudMap* had a considerably fast learning-curve rate. Participants also championed the explicit visualization of cloud resource relationships as it is very useful for navigating through complex cloud resources. Similarly for the widgets that enabled seamless monitoring, analysis and control. Participants also suggested potential extensions such as widgets for: (a) cost visualization; (b) sorting and filtering based on the geographical region and role; (c) cost comparison; (d) scheduling orchestration tasks; and (e) generating recommendations to recover from error conditions.

6 Related Work and Concluding Remarks

Orchestrations tools (e.g., *AWS OpsWorks*, *Juju* or *Docker*) provide languages to represent and manage resources over cloud environments [10,11]. These languages can either be textual, visual or hybrid (i.e., a mixture of both textual and visual notations). The visual paradigm often simplify the manner of understanding compared to textual notations. While such visual techniques can be applied over most of the cloud resource lifecycle, we scope this paper on: navigation, understanding, discovery, monitoring and control concerns of cloud resources.

Discovery, Navigation, Understanding and Selection. Tools and research initiatives such as *AWS Management Console*, *OpenTOSCA*, and *CA AppLogic* provide visual features to facilitate discovery, navigation, understanding and selection of cloud resources [2]. However, these tools provide a flat view (e.g., catalogs) of cloud resources with sorting and filtering features. DevOps may select a particular resource to analyze their attributes, albeit they do not explicitly visualize relationships, dependencies and memberships between cloud resources. This implies DevOps would need to manually mine relationship details via textual descriptions. In contrast to the above, we contribute an extensible framework (i.e., in future additional Entities, Links, Badges and/or Widgets can be curated) for visualizing cloud resources via the familiar notion of mind-maps.

Deployment, Monitoring and Controlling. Tools such as *Juju GUI*, *OpenTOSCA* and *VisualOps* provide visual abstractions to describe deployment workflows and resource topologies [2,11,12]. Cloud resource monitoring tools such as *Nagios* and *CloudFielder* allow DevOps to define Service Level Agreement (SLA), detect anomalies and notify about SLA violations. *AWS Management Console*, *VisualOps*, *CA AppLogic* and other cloud resource management tools provide control features such as restarting, scaling and migration [12]. Ordinarily, DevOps would have to switch between multiple tools for different aspects of the cloud resource management lifecycle, this is time-consuming and cumbersome. In contrast, our tool greatly compliments this work, as we can integrate these features as pluggable widgets to seamlessly and centrally manage cloud resources.

Visual Notations in Other Domains. Visual notations are adopted in other closely related domains, such as Object-oriented programming, and Business Process modeling. Existing visual notations for service orchestration such as Business Process Modeling Notation (BPMN), focus primarily on the application layer. However, orchestrating cloud resources requires rich abstractions to reason about application resource requirements and constraints; support troubleshooting; and flexible and efficient scheduling of resources. *Architexa* [8] visualizes Java-based source codes and execution aspects in terms of hierarchical trees and UML sequence diagrams. WebML introduces a visual notation to model Web sites [3]. All these visual notations or languages adopt Entity-Relationship (ER) models (e.g., graphs, trees, UML class diagrams), which served as our motivation. *Eden* [15] is a visual notation for network management that proposed the concept of *Badges* to associate security and access policies with network devices. Similarly, we were inspired to propose the concept of *Badges* which can be attached to cloud resources to enable *Probes* and *Control Actions* functions.

Summary. Visual techniques provide a refreshing approach in contrast with existing largely text-based solutions. With the vast proliferation of cloud computing and large amount of complex configurations DevOps are faced with, this work provides a timely contribution. Our design was based on a detailed survey comprising 21 experts, where we aggregated, analyzed and applied our findings to propose a visual notation for cloud resource management. We further proposed the notion of *Badges* via drag-n-drop to enable monitoring and control features. To support the effectiveness of our approach, we also identified 3 common visualization patterns. We evaluated our work with a user-study of 12 participants, and our approach yielded significantly promising results with 33.29 % improved efficiency. We are therefore confident our work provides an innovative approach to a new way of cloud management. As future work, we plan to integrate visual notations to specify cloud resource deployment and reconfiguration workflows, also based on our previous work [13]. Moreover, we endeavor to provide high-level monitoring features such as cost estimation and comparison of cloud-based solutions across multiple providers (e.g., *AWS EC2* and *Google Cloud*).

References

1. Armbrust, M., et al.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
2. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA – a runtime for TOSCA-based cloud applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 692–695. Springer, Heidelberg (2013)
3. Ceri, S., Fraternali, P., Bongio, A.: Web modeling language (WebMI): a modeling language for designing web sites. Comput. Netw. **33**(1), 137–157 (2000)
4. Delaet, T., Joosen, W., Vanbrabant, B.: A survey of system configuration tools. In: 24th International Conference on LISA, pp. 1–8. USENIX Association (2010)
5. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns. Springer, Wien (2014)

6. Pigoski, T.M.: Practical Software Maintenance: Best Practices for Managing Your Software Investment. Wiley, New York (1996)
7. Schulte, S., Janiesch, C., Venugopal, S., Weber, I., Hoenisch, P.: Elastic business process management: state of the art and open challenges for bpm in the cloud. Future Gener. Comput. Syst. **46**, 36–50 (2015)
8. Sinha, V., et al.: Understanding code architectures via interactive exploration and layout of layered diagrams. In: Companion to the 23rd ACM SIGPLAN Conference on OOPSLA, OOPSLA Companion 2008, pp. 745–746. ACM (2008)
9. Sun, Y.-J.J., Barukh, M.C., Benatallah, B., Beheshti, S.-M.-R.: Scalable SaaS-based process customization with casewalls. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 218–233. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48616-0_14
10. Turnbull, J.: The Docker Book: Containerization is the new virtualization (2014)
11. Ubuntu: Juju (2013). http://www.ubuntu.com/cloud/tools/juju
12. VisualOps: Visualops - wysiwyg for your cloud (2015). http://docs.visualops.io/
13. Weerasiri, D., Benatallah, B., Barukh, M.C.: Process-driven configuration of federated cloud resources. In: Renz, M., Shahabi, C., Zhou, X., Cheema, M.A. (eds.) DASFAA 2015. LNCS, vol. 9049, pp. 334–350. Springer, Heidelberg (2015)
14. Weerasiri, D., et al.: A model-driven framework for interoperable cloud resources management. Technical report UNSW-CSE-TR-201514, UNSW (2015)
15. Yang, J., Edwards, W.K., Haslem, D.: Eden: supporting home network management through interactive visual tools. In: Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology, pp. 109–118. ACM (2010)