Semantic Techniques for Multi-Cloud Applications Portability and Interoperability

CloudForward 2016 – Madrid – October 18th 2016

Project Toreador: http://www.toreador-project.eu

Beniamino Di Martino
Second University of Naples
CINI
beniamino.dimartino@unina.it
Market competition dynamics often lead to confusing service descriptions, making Service discovery and composition a difficult and cumbersome task.
Portability and Interoperability definitions

- **Data Portability (NIST/OpenGroup-DP):** to transfer or copy data objects to or from different Cloud platforms.

- **System Portability (NIST-SP):** to migrate virtual machine instances, machine images, applications or even services from one Cloud provider to another.

- **Application Portability (OpenGroup-AP):** re-use and migration of applications, across cloud PaaS services.
  - Development and Operational environments.
  - Software Modernization

- **Service Interoperability (NIST-SI):** use services across multiple Cloud platforms.

- **Application Interoperability (OpenGroup-AI):** ability of Cloud applications to collaborate.

- **Platform Interoperability (OpenGroup-PI):** ability of platform components to interoperate.
A positioning 4D «space»

- **Software entities**: Data, Service, Application and System
- **Service level**: Non cloud (in-house solution), IaaS, PaaS and SaaS
- **Deployment model**: Public, Private and Hybrid
- **Cloud features**: Portability and Interoperability
Positioning of Cloud Definitions

- Some definitions overlap
- Not all aspects are covered
  - Are definitions for all combinations needed?
Use Cases

- **CSCC S1**: Customer Switches Providers for a Cloud Service
- **CSCC S2**: Customer Uses Cloud Services from Multiple Providers
- **CSCC S3**: Customer Links One Cloud Service to another Cloud Service
- **CSCC S4**: Customer Links In-house Capabilities with Cloud Services
- **CSCC S5**: Migration of Customer Capabilities into Cloud Services
- **CCUC 1**: Changing SaaS vendors
- **CCUC 2**: Changing middleware vendors
- **CCUC 3**: Changing VM hosts
- **CSC-CB**: Cloud Bursting


**Cloud Computing Use Case Discussion Group.** Cloud computing use cases white paper. [http://cloudusecases.org/Cloud_Computing_Use_Cases_Whitepaper_4_0.odt](http://cloudusecases.org/Cloud_Computing_Use_Cases_Whitepaper_4_0.odt)
Positioning of Use Cases

**CSCC**: Cloud Standards Customer Council

**CCUC**: Cloud Computing Use Case Discussion Group
Standardization Efforts

- Topology and Orchestration Specification for Cloud Applications (TOSCA)
- Cloud Infrastructure Management Interface (CIMI)
- Cloud Data Management Interface (CDMI)
- Open Cloud Computing Interface (OCCI)
- Cloud Application Management for Platforms (CAMP)
- Cloud Standards Coordination (CSC) initiative
- IEEE Standard for Intercloud Interoperability and Federation (IEEE P2302)
- De-Facto Standards: AWS, OpenShift, Cloud Foundry
- **Winery**: graphic environment, accessible through Web browsers, which supports modelling of both TOSCA topologies and plans.
- **OpenTosca**: open source browser-based runtime environment for running applications described using TOSCA specifications.
- **Vinothek**: repository for TOSCA configurations.
CIMI

- Distributed Management Task Force (DMTF) standard
  - Specifies an interface, represented by a set of RESTful APIs, to manage Cloud platforms
  - Focused on the Infrastructure as a Service (IaaS) layer

Focus on the infrastructure management

The CIMI model defines a set of resources, associated templates and configurations, which can be accessed, operated and managed through basic HTTP methods in a RESTful fashion.
  - These include cloud entry points to access lists of all available assets, virtual machines, storage, network and monitoring resources.
  - Security issues are also addressed by the interface, with a focus on client’s identification.
CDMI

- Storage Networking Industry Association (SNIA) standard
- CDMI defines a functional interface that users and applications can use to create, retrieve, update and delete data elements from Cloud storages.
- Discover the capabilities offered by the cloud platform and manage the containers and the data that is placed in them, together with meta-data associated to both containers and data.
RESTful Protocol and API published by the Open Grid Forum (OGF)

A shareable and homogeneous interface to support all kinds of management tasks

Covers IaaS, PaaS and SaaS
IEEE-SIIF (P2302 WG)

Definition of a topology, a set of functionalities and a governance model to support cloud interoperability and federation among different platforms.

- **Intercloud Root** enables the management of the Cloud Resources Directory Services.
- **Intercloud Exchanges** support the resources’ negotiation and collaboration
- **Intercloud Gateways** represent the access points to the Intercloud environment

- Semantic description of resources enables comparison and composition of Cloud services
- **OCCI** standard applied
- Based on FP7 **mOSAIC** project results
Patterns

Design patterns

- Introduced in 1995 by the “Gang of Four”
- Definition of 23 basic patterns
- Description of a solution for a recurrent common problem in a known context
- Based on previous experiences

Cloud patterns

- Recent evolution of Design patterns
- Early stage of development
- Lack of supporting tools and formalization languages
Cloud Patterns

- Defined by Cloud Vendors
  - More specific
  - Strongly tied to the reference Cloud Platform
  - Immediately implementable
  - Poor flexibility

- Defined by Academia
  - High abstraction level
  - No implementation proposed
  - High flexibility
Agnostic VS Vendor Dependent Cloud Patterns

- **Agnostic Patterns** provide generic solutions, which are not bound to a specific platform and are therefore more flexible and seamlessly applicable to different targets. They are not related to a specific Cloud Platform and can virtually be applied to any target environment.

- **Vendor Dependent Patterns** are tailored for a target environment and provide optimized solutions for it. They provide many useful details regarding the actual Cloud components and services to use to deploy an application on the target platform, thus actively supporting developers in their work.
Agnostic VS Vendor Dependent Cloud Patterns

Agnostic Pattern

Amazon Pattern

CloudWatch

EC2

Auto Scale

WEB/AP

WEB/AP

System

AMI

4) Launch Server

3) Notify Auto Scaling

2) Rules

Auto Scaling
Agnostic Patterns

- **Cloud Computing Patterns**
  - Available online at [http://www.cloudcomputingpatterns.org/](http://www.cloudcomputingpatterns.org/)
  - Patterns organized in categories:
    - Cloud computing fundamentals
    - Cloud offerings
    - Cloud application architectures and management
    - Composite cloud applications
  - The description of the Pattern contains:
    - A description of the addressed problem
    - The context in which the solution can be applied
    - The solution itself (supported by images)
    - Related Patterns
Cloud Application Management

- Provider Adapter
- Managed Configuration
- Elasticity Manager
- Elastic Load Balancer
- Elastic Queue
- Watchdog

- Elasticity Management Process
- Feature Flag Management Process
- Update Transition Process
- Standby Pooling Process
- Resiliency Management Process
## Composite Cloud Applications

<table>
<thead>
<tr>
<th>Two-Tier Cloud Application</th>
<th>Three-Tier Cloud Application</th>
<th>Content Distribution Network</th>
<th>Hybrid User Interface</th>
<th>Hybrid Processing</th>
<th>Hybrid Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid Backup</td>
<td>Hybrid Backend</td>
<td>Hybrid Application Functions</td>
<td>Hybrid Multimedia Web Application</td>
<td>Hybrid Development Environment</td>
<td></td>
</tr>
</tbody>
</table>
Composite Cloud application example

Integration Provider

Integration functionality such as messaging and shared data is hosted by a separate provider to enable integrate of otherwise separated hosting environments.

Context

How can application components that reside in different environments, possibly belonging to different companies, be integrated through a third-party provider?

When companies collaborate or one company has to integrate applications of different regional offices, different applications or the components of a Distributed Application are distributed among different hosting environments. Communication between these environments may be restricted and enabling communication may be hindered by corporate regulations.

Solution

The Distributed Applications or their components communicate using integration components offered by a third party provider.

Related Patterns

Hybrid Processing, Hybrid Backend, Hybrid Data, Restricted Data Access Component, Application Component Proxy, Message Mover, Compliant Data Replication
Example: AWS Cloud Design Patterns

The AWS Cloud Design Patterns (CDP) are a collection of solutions and design ideas for using AWS cloud technology to solve common systems design problems.

- Basic Patterns
- Patterns for Improving Availability
- Patterns for Processing Dynamic Content
- Patterns for Processing Static Content
- Patterns for Uploading Data
- Patterns for Relational Database
- Patterns for Batch Processing
- Pattern for Operation and Maintenance
- Patterns for Network
List of AWS Cloud Design Patterns

**Basic Patterns**
- Snapshot Pattern (Data Backups)
- Stamp Pattern (Server Replication)
- Scale Up Pattern (Dynamic Server Spec Up/Down)
- Scale Out Pattern (Dynamically Increasing the Number of Servers)
- On-demand Disk Pattern (Dynamically Increasing/Decreasing Disk Capacity)

**Patterns for High Availability**
- Multi-Server Pattern (Server Redundancy)
- Multi-Datacenter Pattern (Redundancy on the Data Center Level)
- Floating IP Pattern (Floating IP Address)
- Deep Health Check Pattern (System Health Check)

**Patterns for Processing Dynamic Content**
- Clone Server Pattern (Cloning a Server)
- NFS Sharing Pattern (Using Shared Content)
- NFS Replica Pattern (Replicating Shared Content)
- State Sharing Pattern (Sharing State Information)
- URL Rewriting Pattern (Saving Static Content)
- Rewrite Proxy Pattern (Proxy Setup for URL Overwriting)
- Cache Proxy Pattern (Cache Provisioning)
- Scheduled Scale Out Pattern (Increasing or Decreasing the Number of Servers Following a Schedule)

**Patterns for Processing Static Content**
- Web Storage Pattern (Use of High-Availability Internet Storage)
- Direct Hosting Pattern (Direct Hosting Using Internet Storage)
- Private Distribution Pattern (Data Delivery to Specified Users)
- Cache Distribution Pattern (Locating Data in a Location That Is Physically Near to the User)
- Rename Distribution Pattern (Delivery Without Update Delay)
Patterns for Uploading Data
- Write Proxy Pattern (High-Speed Uploading to Internet Storage)
- Storage Index Pattern (Increasing the Efficiency of Internet Storage)
- Direct Object Upload Pattern (Simplifying the Upload Procedure)

Patterns for Relational Database
- DB Replication Pattern (Replicating Online Databases)
- Read Replica Pattern (Load Distribution through Read Replicas)
- Inmemory DB Cache Pattern (Caching High-Frequency Data)
- Sharding Write Pattern (Improving Efficiency in Writing)

Patterns for Batch Processing
- Queuing Chain Pattern (Loose-Coupling of Systems)
- Priority Queue pattern (Changing Priorities)
- Job Observer Pattern (Job Monitoring and Adding/Deleting Servers)
- Scheduled Autoscaling Pattern (Turning Batch Servers On and Off Automatically)

Pattern for Operation and Maintenance
- Bootstrap Pattern (Automatic Acquisition of Startup Settings)
- Cloud DI Pattern (External Placement of Parts That Are Frequently Updated)
- Stack Deployment Pattern (Creating a Template for Setting up Groups of Servers)
- Server Swapping Pattern (Transferring Servers)
- Monitoring Integration Pattern (Centralization of Monitoring Tools)
- Web Storage Archive Pattern (Archiving Large Volumes of Data)
- Weighted Transition Pattern (Transitioning Using a Weighted Round Robin DNS)
- Hybrid Backup Pattern (Using the Cloud for Backups)

Patterns for Network
- OnDemand NAT Pattern (Changing Internet Settings at the Time of Maintenance)
- Backnet Pattern (Establishment of a Management Network)
- Functional Firewall Pattern (Multi-Tier Access Control)
- Operational Firewall Pattern (Controlling Access by Individual Function)
- Multi Load Balancer Pattern (Setting Up Multiple Load Balancers)
- WAF Proxy Pattern (Effective Use of a Costly Web Application Firewalls)
- CloudHub Pattern (Setting Up VPN Sites)
Example: AWS Scale-Out Pattern
Example: Windows Azure Cloud Patterns - Auto-Scaling Pattern
Example: IBM Cloud Patterns

A collection of elements describing a complete software solution
Configuration included in the pattern

A virtual application pattern represents a collection of application components, behavioral policies, and their links. Using this application-centric definition of the workload the IBM PureApplication System will automatically construct the necessary infrastructure and middleware resources to provision and continually manage this virtual application.
Towards a Framework for semantic and cloud-pattern based support to Cloud Portability and Interoperability / Federation

Semantic and Patterns based model, ontology and inferential engine for porting Applications to the Cloud and among Multiple Clouds, and to achieve Cloud Interoperability

Components:

• A Semantic, Machine Readable and Uniform Representation of Cloud Resources, Services and Patterns

• A Semantic and Rule-based System that works over the semantic representation to Support Multi Cloud Portability and Interoperability

• A Semantic and Matchmaking System for Discovery, Mapping and Aligning Cloud Providers’s Services
A Semantic and Rule-based Methodology to Support Multi Cloud Portability and Interoperability

i. identification of the application components starting from an application (an existing legacy application or the design of a new one);

ii. mapping of these components on agnostic cloud patterns and cloud services;

iii. discovery of vendor dependent cloud service and patterns to implement the application components through automated reasoning;

iv. composition among cloud services in order to implement cloud patterns;

v. replacement of cloud services of a particular cloud vendor with other cloud services offering the same functionalities exploiting the composition described in the cloud pattern, thus enabling interoperability.
Based on results of EC project mOSAIC - Open-Source API and Platform for Multiple Clouds http://www.mosaic-cloud.eu

An API

Cloud-based language- and platform-independent API
Extends the existing language- or platform-dependent API capabilities with composite features based on patterns

A framework

Semantic engine
Cloud ontology & Semantic retrieval of agnostic Cloud services and resources driven by Design, Functional and Application Patterns
Automatic inferencing of resources configurations

Cloud agency to perform multiCloud brokering and negotiation
Dynamic Semantic Discovery Service for Services discovery and integration
Application Tools supporting Cloud Apps development

An open-source platform
ready to be tested, exploited or extended by its users; includes instances of the APIs for three programming languages and application tools
A uniform, integrated, machine-readable, semantic representation of cloud services, patterns and appliances.
Patterns Layers
The Entity level
The Entity Level Ontology: Semantic Description of Cloud Providers’ Resources and Services, and Virtual Appliances
The Entity Level Ontology: Semantic Description of Cloud Providers’ Resources and Services, and Virtual Appliances
IEEE P2302 – “Intercloud” Standard for Intercloud Interoperability and Federation (SIIF)

mOSAIC Cloud ontology is being included in the Standard
Cloud Services’ Semantic Description with OWL-s

```
xml:base = "http://127.0.0.1/OSservices/Nova/NovaBoot.owl">
<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Grounding.owl"/>
<owl:imports rdf:resource="http://127.0.0.1/ontologies/CSOntology.owl"/>
<owl:imports rdf:resource="http://127.0.0.1/ontologies/OpenStack.owl"/>
</owl:Ontology>

<process:AtomicProcess rdf:ID="NOVA BOOT PROCESS">
  <service:describes rdf:resource="#NOVA BOOT SERVICE"/>
  <process:hasInput rdf:resource="# SERVER REQ"/>
  <process:hasOutput rdf:resource="# SERVER RES"/>
</process:AtomicProcess>

<process:Input rdf:ID="# SERVER_REQ">
  <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
  http://127.0.0.1/ontologies/OpenStack.owl#Server</process:parameterType>
  <rdfs:label></rdfs:label>
</process:Input>

<process:Output rdf:ID="# SERVER_RES">
  <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
  http://127.0.0.1/ontologies/OpenStack.owl#Server</process:parameterType>
  <rdfs:label></rdfs:label>
</process:Output>
```
Cloud pattern semantic based formal representation

Definition of a common formalism, based on a unique and shared model, which could be used to completely and uniformly describe Design and Cloud patterns, and map them to Cloud Providers’ offers.
### Pattern description languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Base Technology</th>
<th>Structural Description</th>
<th>Behavioural Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lepus</td>
<td>First Order Logic (FOL)</td>
<td>Codechart/Graphical Notation</td>
<td>None</td>
</tr>
<tr>
<td>DisCO</td>
<td>Temporal Logic of Actions (TLA)</td>
<td>None</td>
<td>Pure Text Graphical Description dependent on tools</td>
</tr>
<tr>
<td>BPSL</td>
<td>FOL+TLA</td>
<td>Pure Text</td>
<td>Pure Text</td>
</tr>
<tr>
<td>DPML</td>
<td>UML</td>
<td>Specification and Initialization Diagram</td>
<td>None</td>
</tr>
<tr>
<td>SPINE</td>
<td>Prolog</td>
<td>Prolog facts</td>
<td>None</td>
</tr>
<tr>
<td>DPDL</td>
<td>XML</td>
<td>XML Documents</td>
<td>XML Documents</td>
</tr>
<tr>
<td>ODOL</td>
<td>OWL</td>
<td>OWL Ontology</td>
<td>Limited Support</td>
</tr>
<tr>
<td>DOP</td>
<td>OWL</td>
<td>OWL Ontology</td>
<td>None</td>
</tr>
</tbody>
</table>

- **Missing concepts**
- **Poor flexibility**

**OWL-S**
An example: AWS Scale-Out Pattern
AWS Scale-Out Pattern: ODOL
AWS Scale-Out Pattern: OWL-S
Agnostic vs Vendor dependent concepts
Two Instances of the same Agnostic Pattern and their relationships
Different kinds of equivalences

- Equivalence among services, methods and parameters
- Patterns equivalence
- Equivalence «in role»
Different kinds of equivalences
Enrichment of the Semantic Representation with Automated Reasoning

Different definitions of equivalence (first order logic)

(1) Exact equivalence

\[ \forall x, y (functionallyEquivalent(x, y) \land \\
    \forall z, w (hasMethod(x, z) \land hasMethod(y, w) \\
    \land compatibleMethods(z, w) \rightarrow Exact(y, x))) \quad (1) \]

(2) PlugIn equivalence

\[ \forall x, y (functionallyEquivalent(x, y) \land \\
    \neg Exact(x, y) \land \forall z (hasMethod(x, z) \land \\
    \exists w (hasMethod(y, w) \land compatibleMethods(z, w) \\
    \rightarrow plugIn(y, x))) \quad (2) \]

(3) Subsume equivalence

\[ \forall x, y (functionallyEquivalent(x, y) \land \\
    \neg Exact(x, y) \land \neg plugIn(x, y) \\
    \forall z (hasMethod(x, z) \land \exists w (hasMethod(y, w) \land \\
    adaptableParameters(z, w) \rightarrow Subsume(x, y))) \quad (3) \]
Enrichment of the Semantic Representation with Automated Reasoning

- Assessing equivalence between Services and Appliances

```sparql
SELECT ?service ?appliance
WHERE {
  ?service rdf:type cloudOntology:CloudService.
  ?appliance rdf:type cloudOntology:VirtualAppliance.
  ?service cloudOntology:functionallyEquivalent ?appliance
}
```

**SWRL rule**
- Automatic reasoning
- KB enrichment
- Query simplification

CloudService(?a), VirtualAppliance(?b), CloudCategory(?c), aKindOf(?a, ?c), aKindOf(?b, ?c) => functionallyEquivalent (?a, ?b)

**Sparql queries**
- Information retrieval
- Do not create new information
Enrichment of the Semantic Representation with Automated Reasoning

- Building Vendor specific Patterns from Agnostic ones

Infer equivalence among patterns' components and cloud services

Use equivalence between patterns' participants and services to create new (vendor specific) patterns.

Instantiate patterns with heterogeneous services (with slight modifications)

```
CloudComponentTemplate(?a), CloudService(?b), DifferentFrom(?a, ?b), CloudCategory(?c), aKindOf(?a, ?c), aKindOf (?b, ?c) --> equivalentComponent (?a, ?b)
```

```
SELECT ?agnosticComponent ?vendorComponent ?vendor
WHERE {patternOntology:
    ?agnosticComponent cloudOntology:
        hasVendor ?vendor
}
```
Porting applications to Interoperable Multi-Clouds
Example: Document Workflow application
Application pattern definition:
identification of the application components starting from an application
Application pattern definition:
identification of the application components starting from an application
Discovery of Multi-Cloud Services and Pattern to use
# Discovery of Multi-Cloud Services and Pattern to use

```
   component ?vendor
WHERE { patternOntology:DocumentWorkflow
   patternOntology:include ?
   includedPattern.
    ?includedPattern patternOntology:
    implemented ?providerPattern.
    {{?participant patternOntology:
     participant patternOntology:
     DocumentWorkflow}
   MINUS
   {{?participant patternOntology:
     participant ?providerPattern
   }}
   UNION {{?participant
     patternOntology:participant ?
     providerPattern}
   ?participant patternOntology:
   equivalent ?component
}
```

<table>
<thead>
<tr>
<th>Participant</th>
<th>Pattern</th>
<th>Vendor</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Azure_PipesFilters</td>
<td>Azure</td>
<td>Service_Bus</td>
</tr>
<tr>
<td>Indexing</td>
<td>Azure_PipesFilters</td>
<td>Azure</td>
<td>Worker_Role</td>
</tr>
<tr>
<td>Sorting</td>
<td>Azure_PipesFilters</td>
<td>Azure</td>
<td>Worker_Role</td>
</tr>
<tr>
<td>Classification</td>
<td>Azure_PipesFilters</td>
<td>Azure</td>
<td>Worker_Role</td>
</tr>
<tr>
<td>Analysis</td>
<td>Azure_PipesFilters</td>
<td>Azure</td>
<td>Worker_Role</td>
</tr>
<tr>
<td>Storage</td>
<td>MySQL</td>
<td>MySQL</td>
<td>MySQL</td>
</tr>
<tr>
<td>Notification</td>
<td>Azure</td>
<td>Azure</td>
<td>Service_Bus</td>
</tr>
<tr>
<td>Notification</td>
<td>OpenShift</td>
<td>OpenShift</td>
<td>JBoss</td>
</tr>
</tbody>
</table>
### Discovery of Multi-Cloud Services and Pattern to use

<table>
<thead>
<tr>
<th>Participant</th>
<th>Vendor</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Azure</td>
<td>Service_Bus</td>
</tr>
<tr>
<td>Communication</td>
<td>OpenShift</td>
<td>RabbitMQ</td>
</tr>
<tr>
<td>Indexing</td>
<td>Azure</td>
<td>Worker_Role</td>
</tr>
<tr>
<td>Indexing</td>
<td>IBM</td>
<td>WebSphere</td>
</tr>
<tr>
<td>Indexing</td>
<td>OpenShift</td>
<td>JBoss</td>
</tr>
<tr>
<td>Sorting</td>
<td>Azure</td>
<td>Worker_Role</td>
</tr>
<tr>
<td>Classification</td>
<td>Azure</td>
<td>Worker_Role</td>
</tr>
<tr>
<td>Analysis</td>
<td>Azure</td>
<td>Worker_Role</td>
</tr>
<tr>
<td>Storage</td>
<td>MySQL</td>
<td>MySQL</td>
</tr>
<tr>
<td>Notification</td>
<td>Azure</td>
<td>ServiceBus</td>
</tr>
<tr>
<td>Notification</td>
<td>OpenShift</td>
<td>RabbitMQ</td>
</tr>
</tbody>
</table>

**SELECT** ?participant ?vendor ?component
**WHERE** { patternOntology:DocumentWorkflow
  patternOntology:participant ?participant.
  ?component cloudServiceOntology:hasVendor ?vendor }
## Comparison with TOSCA

<table>
<thead>
<tr>
<th>TOSCA</th>
<th>Multidimensional Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description of Cloud services</td>
<td>Semantic-based description of Cloud services, also agnostic</td>
</tr>
<tr>
<td>Representation of services interactions and behaviour</td>
<td>Description of services orchestration via OWL-S</td>
</tr>
<tr>
<td>Services compositions represented via templates</td>
<td>Services composition obtained via Patterns (Application\Cloud\Design)</td>
</tr>
<tr>
<td>Support to application deployment</td>
<td>Production of code snippets via skeletons</td>
</tr>
<tr>
<td>Matching based on capabilities and requirements</td>
<td>Semantic matchmaking via inference rules</td>
</tr>
<tr>
<td></td>
<td>Definition of new patterns based on existing ones</td>
</tr>
</tbody>
</table>
A prototype tool: Cloud Composer

- Browsing of the Layered ontology
- Visual Creation of Patterns’ description
- Inferencing
- Matchmaking
Job Observer Agnostic Pattern
Job Observer Pattern for Openstack
Cloud Composer GUI

- Visualization Patterns workflow
- Reasoning
- Cloud Patterns creation
- Patterns Mapping
Patterns Workflow Visualization

1. Open ...
   - New ...
   - Edit ...

2. Agnostic JobObserverPattern
   - AgnosticThreeTierPattern
Patterns Workflow Visualization

Agnostic Pattern: AgnosticJobObserverPattern

Highlight Equivalent Services And Methods

Patterns WorkFlow | Pattern's Fragments | Patterns WorkFlow Creation | Pattern's Fragments Creation

Client → VirtualMachines → QueuesPipelines → ResourceScaling → ResourceMonitoring

Open

Visualize A Workflow for pattern

Pattern properties

Client access to pattern properties

Delete/modify pattern properties

Visualize workflow
Patterns Workflow Visualization

SPARQL Query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX and: <http://127.0.0.1/ontologies/ExtendedWop.owl#>
PREFIX p: <http://127.0.0.1/CloudPatterns/OpenStackPatterns.owl#>

SELECT ?proprietary
WHERE { ?proprietary and:agnosticEquivalent p: AgnosticJobObserverPattern}
```
Patterns Workflow Visualization
Reasoning 1/5
Reasoning 2/5
Reasoning 3/5

PREFIX cso: <http://127.0.0.1/ontologies/CSOntology.owl#>
SELECT ?categoria
WHERE { cso: serviceName cso:aKindOf ?categoria }

PREFIX amz: <http://127.0.0.1/ontologies/Provider.owl#>
PREFIX po: <http://127.0.0.1/ontologies/Parameters.owl#>
SELECT ?othermethod
WHERE { amz: methodName po:isEquivalentTo ?othermethod }
Reasoning 5/5
Browsing of Knowledge Base 1/4
Browsing 2/4
Browsing 3/4
Patterns Creation 1/5
Patterns Creation 2/5
Patterns creation 3/5
Patterns creation 4/5
Patterns creation 5/5
Patterns Equivalence 2/4

- **Simple Fragment**
- **Complex Fragment**

All Pattern's Fragment

- AllPatternsFragment
  - AgnosticJobObserverPattern vs. JobObserverPattern
  - AgnosticPipesAndFiltersPattern vs. PipesAndFiltersPattern
  - AgnosticThreeTierPattern vs. ThreeTierPattern
  - AgnosticJobObserverPattern vs. ScaleOut

CoupledFragment0
- Type: node Label: OpenStack_Ceilometer
- Type: nodes Label: ResourceMonitoring UNION ResourceScaling

CoupledFragment1
- Type: node Label: OpenStack_Nova
- Type: node Label: VirtualMachines

CoupledFragment2
- Type: edge Label: NovaBoot
- Type: edge Label: CreateVM

CoupledFragment3
- Type: edge Label: NovaStart
- Type: edge Label: StartVM

CoupledFragment4
- Type: edge Label: AlarmThresholdCreate
- Type: edges Label: CreateASConfiguration UNION CreateMetric

CoupledFragment5
- Type: edge Label: SampleCreate
- Type: edge Label: CreateStatistic

AgnosticPipesAndFiltersPattern vs. PriorityQueuePattern
Pattern Equivalence 3/4
Patterns Equivalence 4/4
Mapping Application Patterns to Agnostic Cloud Patterns

Distributed Batch k-means Application Pattern and its mapping to Map-Reduce
Mapping Agnostic Cloud Patterns to Agnostic Service
From Distributed Batch k-means to Map-Reduce
Mapping Agnostic to Vendor Specific Cloud Patterns
From Map-Reduce to Azure specific solution (and vice-versa)

Solution not using Azure's native support to Map-Reduce
Mapping Agnostic to Vendor Specific Cloud Patterns
From Map-Reduce to Azure specific solution (and vice-versa)

Solution exploiting Azure's native support to Map-Reduce
Describing the DKMS Application Pattern with OWL-S
Mapping DKMS Application pattern to MapReduce

DKMS

Client

AssignRecalculate

Shared File System

MapReduce

Client

Map

Reduce

Shared Storage

0. Start

1. GetData
2. Assign Prototype
3. PutData

7a. ReturnResults

7b. NewIteration

0. Map

1. GetDataLocation
2. DoMapping
3. Reduce

5. Calculate

8. PutData

9. ReturnResults

0. Map

4. GetData
5. Assign Prototype
6. PutData

6a. ReturnPartial

7. Recombine
Mapping DKMS Participants and Methods
Towards automated Applications Portability to Multi-Clouds
Parallel skeletons

- There are patterns in parallel applications
- Those patterns can be generalized in Skeletons
- Applications are assembled as combination of such patterns

- Functional point of view
- Skeletons are Higher-Order Functions
- Skeletons support a compositional semantic
- Applications become composition of state-less functions

- Orchestration and synchronization of the parallel activities are implicitly defined and hidden to the programmer
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;

public class yourProgram {
    public static class yourMapper extends Mapper<Object, keyClass, keyClass, WritableClass> {
        private final static WritableClass one = new WritableClass();
        private keyClass keyC = new keyClass();
        public void map(Object key, keyClass value, Context context) {
            [Write here the mapper code]
            context.write();
        }
    }

    public static class yourReducer extends Reducer<keyClass, WritableClass, keyClass, WritableClass> {
        private WritableClass result = new WritableClass();
        public void reduce(keyClass key, Iterable<WritableClass> values, Context context) {
            [Write here the reducer code]
            context.write();
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "comment");
        job.setJarByClass(yourProgram.class);
        job.setMapperClass(yourMapper.class);
        job.setCombinerClass(yourReducer.class);
        job.setReducerClass(yourReducer.class);
        job.setOutputKeyClass(keyClass.class);
        job.setOutputValueClass(WritableClass.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
Conclusions

• Lackness of standards for enabling
  • Applications’ portability among multiple Clouds
  • Service-level Interoperability and composition
• Semantic technology can overcome such standards’ lackness
• A common, vendor agnostic, semantic model
• Agnostic Cloud patterns to represent and design Cloud applications
• Semantic technologies (inference, matchmaking) to map applications to Cloud providers’ offers and to enable services’ interoperability and composition, based on Cloud Patterns
Thanks for your attention!

beniamino.dimartino@unina.it
http://www.diii.unina2.it
http://www.diii.unina2.it/it/dipartimento/persone/docenti-elenco/ordinari-elenco/dimartino