An architecture for decentralized service deployment

Daniel Lázaro Iglesias
Joan Manuel Marquès
Josep Jorba Esteve
Universitat Oberta de Catalunya

Summary

- Introduction
- Requirements
- Architecture
- Conclusions

Introduction

- Virtual community of users who share common interests
- They want to offer services
  - To the members of the community: activity coordination, file sharing, etc.
  - To the rest of the world: information about the community.
  - Examples: file sharing, shared calendar, web publishing, Content Management Systems (CMS), wikis, forums, blogs, audio and video streaming, etc.

Introduction

- The community doesn't have a single provider which satisfies its needs of resources. It needs to:
  - Aggregate the resources of different sources
  - Use resources provided by the members of the. community.
- Resources must be aggregated easily.

Characteristics

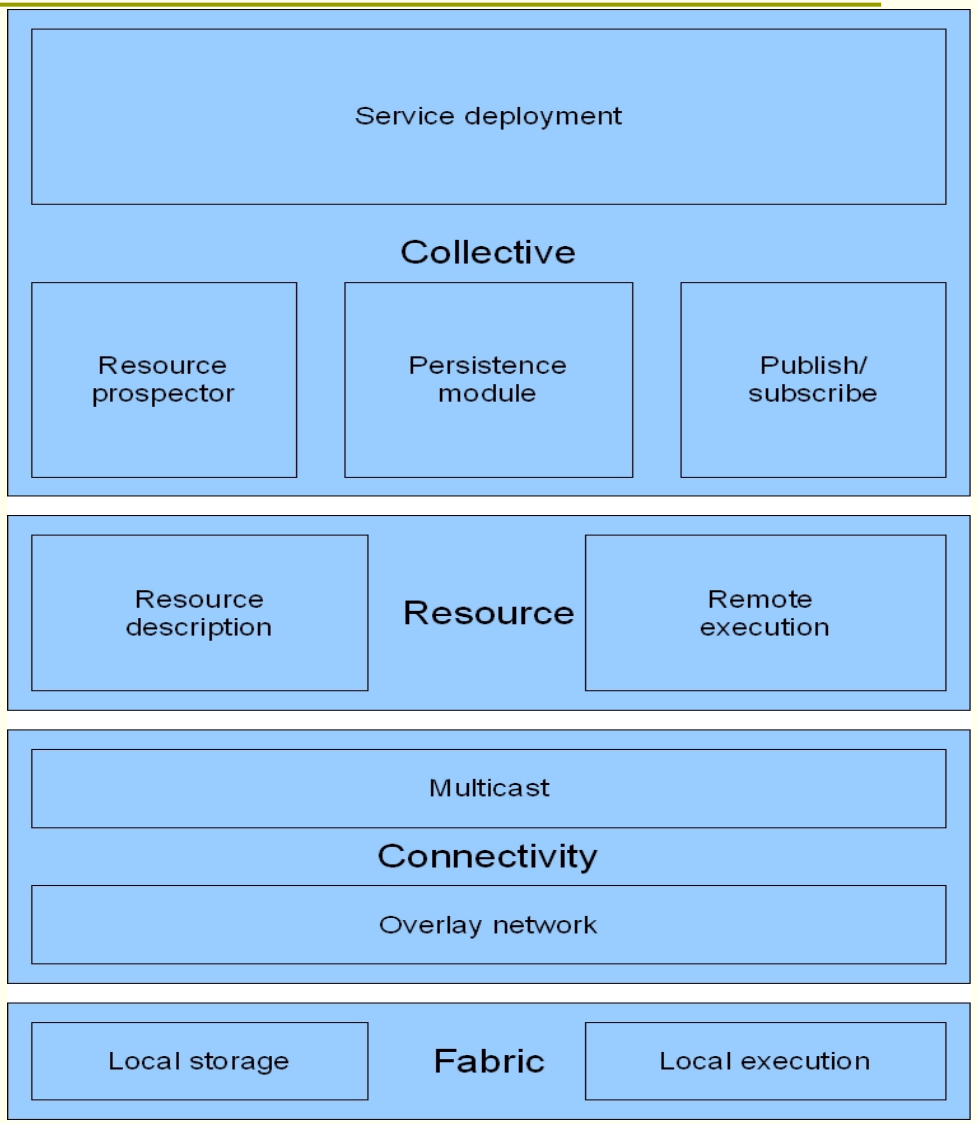
- Scale:
  - Potentially big groups.
  - Members scattered around the world.
- Users may not have technical knowledge
  - Self-* properties will be needed
- Members provides their own resources. This implies:
  - Dynamism
  - Heterogenious resources

Requirements

- Community self-sufficiency
- Individual autonomy
- Decentralization
- Scalability
- Heterogeneity
- Fault tolerance
- Location transparency
- Self-management

Architecture

- Layered architecture (grid-like).
  - Fabric
  - Connectivity
  - Resource
  - Collective



Fabric

- Nodes must offer methods for:
  - Local storage
  - Local execution
- Common interfaces to deal with heterogeneity.
- Each member is free to offer resources at will.
- At this level, methods can even control access to a pool of resources (cluster, sub-group, etc). To the upper layers, it is seen as a single node with a standard interface.

Connectivity

- This layer forms an overlay network:
  - DHTs satisfy our requirements (scalable, decentralized, etc.)
  - We will use common DHT operations so our system can be ported to many DHTs.
- This level will also provide multicast messaging
  - Many mechanism for multicasting in DHTs exist.
  - Our system should work with any multicast system implemented over a DHT.

Resource

- Language to describe resources.
  - Many languages exist:
    - Resource Specification Language (RSL)
    - Job Description Document (JDD)
    - Job Submission Description Language (JSDL)
- Mechanism to control remote execution.
  - Globus' GRAM
  - OGSA's execution management
  - We can adapt these mechanisms for our system

Collective

- Components which allow the collective use of the resources.
  - Publish/subscribe
  - Resource prospector
  - Persistence module
  - Service deployment module

Publish/subscribe

- Type-based
  - Hermes: each type is assigned to a node of the DHT, which is the root of a Scribe-like multicast tree.
- Content-based
  - Events are defined by attributes.
  - Clients subscribe to a range of values of each attribute.
  - Many approaches consider different degrees of flexibility for subscriptions

Resource prospector

- Looks for resources that fulfill a certain specification.
- To accomplish this, the resource prospectors present in the community keep a distributed index of resources.
- They must perform a range search. Systems that do this:
  - Willow
  - SDIMS
  - Cone

Persistence module

- Stores objects distributedly.
  - This is the typical functionality of DHTs.
- We need some more sophisticated functionalities:
  - Store mutable data
    - Store the state of services.
    - P2P filesystems.
  - Balance loading
    - Virtual servers

Service deployment module

- Main functionality of the system.
- Uses the other components to implement its mechanisms.
- It must keep the services available at all times (if provided resources are sufficient).
  - Stateless services.
  - Stateful services with some limitations.
    - Persistance module will provide some methods to store state explicitly.

Service deployment

- Subcomponents:
  - Service deployer: decides number of replicas according to availability required, and controls the overall deployment of the service.
  - Service allocator: finds nodes to execute the service and starts remote execution.
  - Service self-healing: reallocates services from failed nodes.
  - Service self-tuning: creates or destroys replicas to maintain the availability of the service and the performance of the virtual community.

Service deployment

- Subcomponents:
  - Service self-configuration: reacts to nodes joining and leaving and reallocates services in the most convenient way.
  - Service client: allows users to find services and contact them. Different communication patterns must be possible:
    - Transparent access to a service
    - Access to a specific replica
    - Others

Conclusions and future work

- We have presented an architecture for decentralized service deployment.
- Many components can be implemented with existent state-of-the-art technologies.
- Future work:
  - Precisely design the mechanisms of the Collective layer (service deployment, resource prospector, persistence module).
  - Refine the requirements of the upper layer for the lower ones.