Ninja architecture for robust Internet-scale systems and services Gribble et. al., Computer Networks, 2001

Alan Sussman

CMSC 818S

April 3, 2007

Notes

- Project issues?
  - interim report due 4/18
- Date for final?

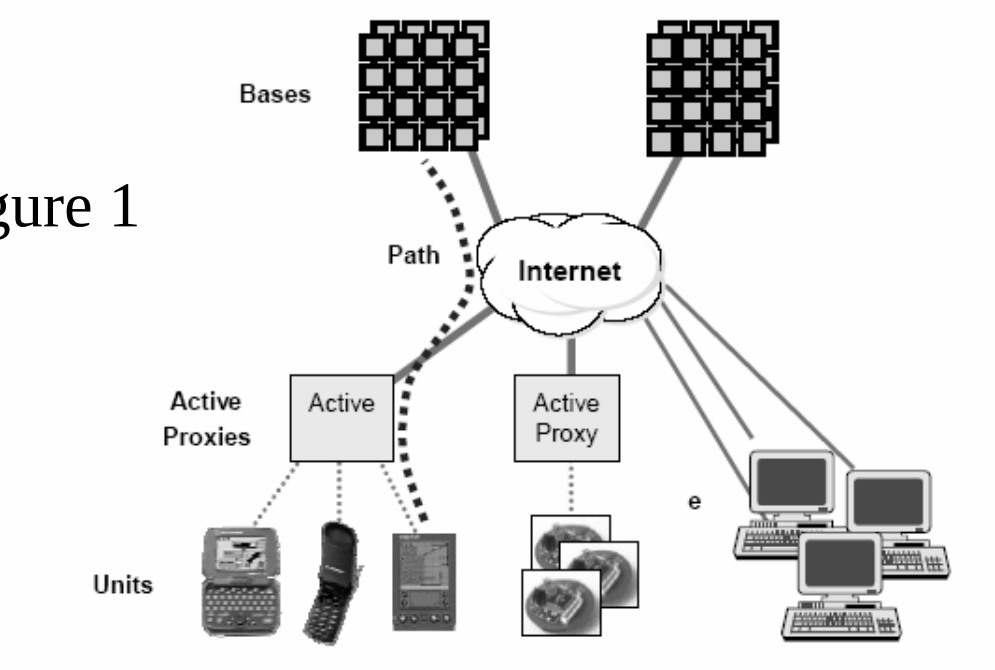
Overview/critique

- Note that this is before OGSA, Grid service-oriented architectures, etc.
- A comprehensive view of how the world "should be"
  - remember Legion?
  - clearly this view has not taken over the world
- But hindsight is easy …

Overall goals and architecture

- Two main goals:
  - broad innovation of service design
  - easy construction of scalable, robust services

Figure 1



Service architecture

- 3 tier network architecture
  - scalable service platforms
  - transformational intermediaries between devices and services
  - devices (clients)
- Processing power and persistent storage on PC/workstation clusters
- Soft state and functional transformations close to devices

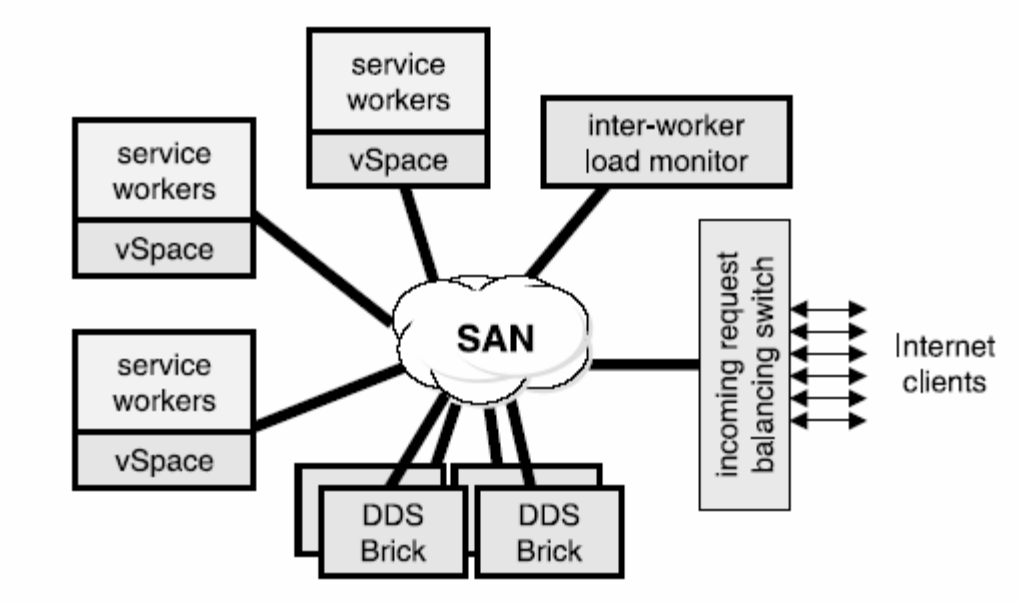
Service architecture (cont.)

- Language issues
  - all services written in Java for type safety, to reduce errors and ease composition of services through well-defined interfaces (components?)
- Services advertised on service discovery service (should sound familiar, like a Grid info service)
- Services composed to build new services
- Required service properties include scalability, availability, fault-tolerance, data consistency and persistence

Base

- A “well-engineered” cluster of workstations used to run services, connected via a high performance system area network
- Software platform is called **vSpace**
  - the class that provides scalability, fault-tolerance and composability, and all services inherit from
  - service authors only plug in application-specific functionality

Base software architecture



Units

- The set of client devices
  - PCs, laptops, PDAs, mobile devices, sensors and actuators (motes)
  - may have limited connectivity, bandwidth, processing power, use simple protocols and data formats
  - so may need surrogates to adapt content and protocols on their behalf

Active Proxies

- Placed between devices and services (bases), to adapt content and access protocols to the limited capabilities of devices
- Transformations include
  - distillation – transform data types
  - adapt protocols (e.g., simplified authentication)
  - adapt content (e.g., remove sensitive info before display on an untrusted access point)

Path

- For service composition, dynamically composed
- Defined as a flow of typed data through multiple services, dynamically chosen (and can change over time)
  - presumably eventually ends in a device
- Requires **service discovery**
  - to a service discovery service (SDS), with interfaces for both humans and programs, hierarchically structured

Services programming model

- 4 design patterns for stages
  - *wrap* – queue in front of stage, and threads within stage to service tasks in queue
    - conditions stage to load through buffering in queue
  - *pipeline* – split wrapped stage into 2 stages
    - decouples, and allows functional parallelism across cluster nodes or processors
  - *combine* – inverse of pipeline – fuse 2 stages into one wrapped one
  - *replicate* – duplicate a wrapped stage on different nodes
    - to eliminate bottlenecks by increasing throughput
- All network communication and disk I/O provided by library that supplies the patterns is nonblocking, and uses asynchronous, event-driven programming style (like a GUI)
  - and custom high performance Java I/O library for network communication

vSpace execution platform

- Use services programming model to construct graph of **workers**
  - each worker is a fixed-size thread pool, incoming event queue and a set of methods
  - corresponds to stage in programming model
- Worker instances pipelined and can execute in parallel on nodes in the cluster
  - share code with other copies of same worker, and share global persistent state through distributed data structures supported by vSpace

Distributed data structures

- Self managing storage layer that runs on a **base**
  - provides usual service properties, including strict data consistency, with a data structure interface
  - only implemented a DHT
    - not so easy to do for other data structures

More on **units**

- Concentrate on networked sensors and devices
- A **mote** is one example
  - microcontroller, radio, photo and light sensors
  - programming model is just like vSpace, called TinyOS
  - but power is the main concern, and the radio is the biggest power user
  - using event-driven system allows processor to be in standby, low-power mode between events

Active proxies

- To match client devices to services
  - adapt data and access protocols to the devices' and services' needs
- Provide 3 essential properties
  - dynamic service adaptation
    - e.g., to simplify network protocols, reformat content
  - secure access to information
    - to adapt security requirements of services to (limited) capabilities of a device
  - fusion of multiple devices
    - e.g., to use multiple input or output devices together

Paths

- A path is a sequence of **operators** that compute on data, and **connectors** that translate protocols between operators
  - all described with XML descriptors for the types and attributes of each one
- Long-lived operators are standard services, with data persistence, fault-tolerance, etc.
- Dynamically created operators are light-weight and short-lived to do transformations
  - only have soft-state and run in active proxies