The AMbient Open Services Architecture

Johan Lukkien

Melissa Tjong, Remi Bosman, Shudong Chen, Richard Verhoeven
System Architecture & Networking
Department of Mathematics & Computer Science
Eindhoven University of Technology

This work is funded partially by project EES.5413 and Freeband project ISHARE

4/22/2007

spring days on service oriented architecture
Agenda

• Background and motivation
• AMOSA architecture
• Conclusion
Examples

• Consider a meeting
  – setup
    • powerpoint presentation: laptop – beamer
    • PDAs and laptops in the audience
  – scenarios
    • upon a local ‘double-click’ someone else takes control over the presentation
    • limit this behavior according to a policy
    • select the beamer for a different laptop

• Consider a slideshow
  – setup
    • picture directory, image processor and viewer on separate locations
  – scenarios
    • select and modify viewpoint and control dynamically
    • select viewpoint and control, based on context

• Consider embedded electronics (in sensors, consumer electronics)
  – scenario (‘Ambient Intelligence’)
    • combine these, to establish joint behavior
      – (e.g. follow-me functionality, combining movement sensor and heating)
Scenario thinking

- We can realize such a scenario
  - given enough time and money we can make just about anything
- But:

  Can we make system & software components such that future, as yet unforeseen, cooperations and adaptations are simply realizable, and actually work?

- Notes:
  - including ‘extra-functional’ properties: ...ilities, security
  - need to re-think the role networking / distribution plays in system design
    - regard applications as compositions of components
  - during development, do not only look (only) at typical end-user functionality
  - two cuts in the software architecture matter:
    - inside devices
    - between devices
Service Oriented

- S.O.A. fits the examples nicely
  - CE standards, like UPnP, support a service oriented concept
    - though more 'remote control' than orchestration
  - also Web services are used in 'small service' experiments
Slideshow (discovery)

Repository (URI)

Orchestrator
Slideshow (discovery)

Repository (URI) → ? → Orchestrator
Slideshow (discovery)

- Repository (URI)
- Orchestrator
- Mouse 1
- Mouse 2
- Image Source 1
- Slideshow viewer

4/22/2007

spring days on service oriented architecture
Slideshow (setup)

- Repository (URI)
- Orchestrator
- Mouse 1
- Mouse 2
- Image Source 1
- Slideshow viewer
- Subscribe
- Subscribe

4/22/2007 spring days on service oriented architecture
Slideshow (setup)

- Repository (URI)
- Orchestrator
  - Subscribe
  - Mouse 1
- Image Source 1
- Slideshow viewer
- Mouse 2

4/22/2007 spring days on service oriented architecture
Slideshow (setup)

Repository (URI) -> Orchestrator

Orchestrator -> Subscribe

Mouse 1

Image Source 1

Slideshow viewer

Mouse 2
Slideshow (setup)

- Repository (URI)
- Orchestrator
- Mouse 1
- Mouse 2
- Image Source 1
- Slideshow viewer
Slideshow (setup)

- Repository (URI)
- Orchestrator
- Image Source 1
- Slideshow viewer
- Mouse 1
- Mouse 2

Bind
Slideshow (setup)

- Repository (URI)
- Orchestrator
- Mouse 1
- Mouse 2
- Image Source 1
- Slideshow viewer
Slideshow (setup)

Repository (URI) — Orchestrator

Image Source 1 — Slideshow viewer

Mouse 1

Mouse 2

Bind


Slideshow (setup)

- Repository (URI)
- Orchestrator
- Image Source 1
- Slideshow viewer
- Mouse 1
- Mouse 2

Connections:
- Repository (URI) connected to Orchestrator
- Orchestrator connected to Mouse 1
- Mouse 1 connected to Slideshow viewer with a "Subscribe" arrow
- Image Source 1 connected to Slideshow viewer

Additional information:
- 4/22/2007
- spring days on service oriented architecture

Image Source

Orchestrator

Subscribe
Slideshow (interaction)

- Repository (URI)
- Orchestrator
- Mouse 1
- Mouse 2
- Image Source 1
- Slideshow viewer

4/22/2007 spring days on service oriented architecture
Slideshow (interaction)

- Repository (URI)
- Orchestrator
- Mouse 1
- Mouse 2
- Image Source 1
- Slideshow viewer

Click(btn1)
Slideshow (interaction)

Repository (URI) - Orchestrator - Mouse 1

Mouse 2

Image Source 1 - Slideshow viewer

4/22/2007
Slideshow (interaction)

Repository (URI) → Orchestrator → Mouse 1

Image Source 1 → Orchestrator

Mouse 1 → Mouse 2
Slideshow (interaction)

Repository (URI) → Orchestrator → Mouse 1

Click(btn1)

Image Source 1 → Mouse 2

spring days on service oriented architecture
Slideshow (interaction)

Repository (URI) -> Orchestrator
Orchestrator -> Mouse 1
Orchestrator -> Mouse 2
Image Source 1 (GetImage(1))
Slideshow (interaction)

- Repository (URI)
- Orchestrator
- Mouse 1
- Mouse 2
- Image Source 1
- Image Source 2

4/22/2007

spring days on service oriented architecture
Slideshow (cleanup)

Repository (URI) — Orchestrator — Mouse 1

unbind

Image Source 1 — Image Source 1

Mouse 2

4/22/2007
Slideshow (cleanup)

Repository (URI) → Orchestrator

Mouse 1 → Slideshow viewer

Image Source 1

Unsubscribe
‘Normal’ outline of interaction

UDDI Directory

Service Requestor

SOAP protocol Transport (http or ...)

Service Provider

Bind, execute

from: Mark Greenwood University of Manchester

4/22/2007
3rd party binding

Repository service (provider)

Service user (may be another service)

Orchestrator (service user)

publish & describe

find

bind

bi-directional use

publish & describe

spring days on service oriented architecture
spring days on service oriented architecture
Scenario thinking

• We can realize such a scenario
  – given enough time and money we can make just about anything
• But:

  Can we make system & software components such that future, as yet unforeseen, cooperations and adaptations are simply realizable, and actually work?

• Notes:
  – including ‘extra-functional’ properties: ...ilities, security
  – need to re-think the role networking / distribution plays in system design
    • regard applications as compositions of components
  – during development, do not only look (only) at typical end-user functionality
  – two cuts in the software architecture matter:
    • inside devices
    • between devices
Inside and between devices

Interoperability focused on protocol; no language or platform binding besides message structure and semantics.

Hide network details by extending platform services Binding mostly through libraries.
Simplified view on running component

- Four dependencies
  - (In fact: GUI can be a separate component)
- Single processor application: 3, 4
- Client: 1, 3, 4
- Server: 2, 4
- Example: DNS
- S.O.A.: also 1, 2, 4
Service Oriented

- Standards like UPnP and WS focus on the network protocols
  - leaving service development to tools (‘stack’ + development kit)
    • with particular approaches, often not very ‘component-like’
    • diversity in realization of the standards (interoperability problems between stacks)

- The standards provide a platform, but also introduce concepts and particular approaches
  - Service Discovery scope
    • in the slide-show example you want to limit this somehow...
    • ... but not just to the local net (like, for example, UPnP)
  - Concepts like control point, service, interface, description, eventing, composition, ...

- Extra-functional properties evolve slowly as part of the standard
  - security, reliability

- Resource management is usually out of scope

4/22/2007

spring days on service oriented architecture
From Spring days 2005: Challenges

- Focussed service discovery
  - in combination with locality, qualities, ownership, membership, ...
  - ... just general queries

- Embedded decision procedures
  - interpretation of descriptions, selection, learning
    - essential in service discovery

- Security, privacy, ownership

- Evolution path
  - include legacy, even currently developed legacy
From Spring days 2005: Challenges

• Service development
  – self-containedness, granularity, performance
  – interfaces exposing mechanisms for non-functional properties

• Application development
  – ‘language’ having S.O.A. elements as primitives
  – specify policies
  – deal with mobility and connection failure as regular behavior
    • time/space separation
  – exception handling
  – analysis: visualization / simulation
Agenda

• Background and motivation
• AMOSA architecture
• Conclusion
AMOSA

• Aim
  – ‘lego’-like way of defining networked services
  – limit dependence on used standards, and their evolution
  – experimentation platform
    • example scenarios, understanding the (SOA) programming paradigm
    • facilitate the process of developing new services and compositions
    • limit binding to language, SOA standard
    • experimenting with the extra-functional properties

• Features
  – explicit requires/provides interfaces
    • called directly, or by another service via external (“3rd party”) binding
  – grouping into virtual communities
    • service discovery and access at community level
  – a set of standardized service interfaces, bound independently
    • possibly to different services
  – mapping to UPnP & Web services, as a carrier
Service development: A_MAP and A_ITF

- **A_ITF**
  - a set of standard service interfaces

- **A_Map**
  - given:
    - a service description (file)
    - service implementation code in language $D$ (Java, C++)
  - generate:
    - an intermediate AMOSA service, language $D$ based, with standard interface handlers added
  - add mapping information:
    - generate a deployable component in the specified standard
    - mapping intended to be transparent for the standard

---

4/22/2007

spring days on service oriented architecture
AMOSA – what is it

- An interface definition: A_ITF
  - language independent

- Mappings/bindings – currently to UPnP and Web services

- Tooling, (A_MAP) to facilitate the mapping

- An orchestration language
  - to combine services into applications, program with them

- Several special services
  - repository, community maintenance, device manager
Virtual community services

Applications/Services

Member Management

Service Management

Security Management

Orchestration

Recommender

Credit Management

service service service service

Device Device
Operationally (partly implemented)

- Special services run on some devices
  - community services:
    - community membership is implementation of discovery scoping & service access control
    - the community is implemented as a set of services
  - repository: (community bound)
    - capable of registering services and answering queries
  - device manager: (each device)
    - registers services at repository, start/stops services upon request, resource control
- Note: community can be world-wide
Operationally (partly implemented)

• Operation
  – *device manager*
    • searches community, registers and obtains community access points, access key
      – bootstrapping: find community
      – the service discovery mechanism of the standard still works and may be used
    • registers at community repository, becomes accessible for community
  – *orchestrator*
    • searches community, registers with it
    • searches services at repository, binds them according to its program
Current state

- Initial version of A_MAP developed
  - Python-based development of both services and orchestrators
- Design of virtual community
- Repository
- Set of sample services and scenarios
- Mix of languages and tools
  - and problems....
Conclusions

- Third-party binding is key to success of S.O.A.
  - service needs no knowledge about operating context

- A more abstract component model is needed than just a S.O.A.-standard
  - preserving coding efforts
  - interoperability, standardizing on some system tasks
  - separate mapping helps to adjust an entire code base
    - to new versions of standards
    - to requirements of external behavior (e.g., concurrency in interface access)
    - adding new interfaces to services
    - adding resource management

- Discovery scoping is good for scalability and more secure collaboration

- Intelligence can go into the orchestrator as to what service compositions are possible

- Given a proper language, an “orchestrator interpretation function” is feasible as a separate service