System Composition and Reconfiguration in Ubiquitous Computing scenarios

Stefano Monti, Ph.D.
stefano.monti6@unibo.it

Bologna, September 17th, 2009
Agenda

- **Introduction**
  - Ubiquitous Computing issues
  - Disappearing Middleware approach

- **Service Composition**
  - rationale
  - state of the art and current trends
  - service composition in Ubiquitous Scenarios

- **System Reconfiguration**
  - rationale
  - state of the art and current trends
  - reconfiguration in Ubiquitous Scenarios

- **Conclusions**
Ubiquitous Computing scenario

**Ubiquitous computing**: services and contents *anytime, anywhere*, by means of *any device* at hand
Driving forces

Dynamic extension & heterogeneity
- novel contents and services
- novel communication channels
- novel interaction modes and media
→ Service composition

Dynamic tailoring → need to adapt
- contents/services
- communication patterns and interaction modes
- user needs
- context
- ...
→ System reconfiguration
Disappearing middleware approach

A very basic **kernel layer** with
- service composition features
- system reconfiguration features

**Application layer**: ubiquity-related logic
- e.g., content retrieving, adaptation, delivery

**Non-functional layer**: application support logic
- e.g., service coordination, naming, persistence

Both application non-functional layer need to be
- Extensible
- Reconfigurable
SERVICE COMPOSITION
Rationale

Service-oriented Architecture (SOA) promotes

- software modularization and reuse
- standardization

Web Services are the most widespread implementation of SOA

Service composition allows to aggregate and coordinate services (software modules) to build larger and more complex features
Static service composition

Static composition

- immutable specification of how services cooperate
- direct human intervention: users need to know
  - which services are available
  - what they do (and how): application domain knowledge

Workflows as a preeminent approach in the field (BPEL)

- services as “blocks” of an execution flow
- control blocks: forks/joins, conditional branches, loops
Workflow Management Systems

Workflow Management Systems (WFMS)

- **describe** complex workflows of services (e.g., BPEL standard)
- **execute** and **monitor** workflows (e.g., JBPM engine)

```xml
<process-definition name="web-adaptation-workflow">
  <start-state name="start">
    <transition name="begin" to="state_1"/>
  </start-state>
  <state name="state_1">
    <transition name="service_1" to="state_2">
      <action name="web_server_service"
             class="RequestHandler">
        <typology>
          <interception/>
        </typology>
      </action>
    </transition>
  </state>
  ...
</process-definition>
```
Static composition limitations

- Available services need to be **known in advance**
- Users need to have a full and solid **understanding** of service **business logic** and implementation

On the contrary, in Ubiquitous Computing:

- **Dynamic extension**: services/contents may become available or disappear
- **User transparency**: end-users are interested in high level business goals instead of how to achieve them

Dynamic service composition
Dynamic service composition

Service compositions

- **built** runtime (on demand):
  - no a-priori knowledge of available services
  - no a-priori configuration

- **automated** fashion (little or no human intervention)
  - high level requirements or business goals
  - “intelligent” service selection
  - correct composition of services

- **open**:
  - virtually, no limit on the kind of services involved
  - novel services can be involved into novel compositions
Dynamic composition tendencies

Semantic service composition

**Semantics** as a mean to

- convey high level metadata about services (different standards, e.g., OWL-S...)
- infer novel compositions from high-level business goals
  - openness
- reason on composition correctness
  - completeness

However, current proposals are **non user-friendly** and thus lack concrete adoption.

- billing=0€
- behavior=transcoding
- pool-size=1; blocking=true
- input=text; output=audio/x-wav
Dynamic composition tendencies

Web mashups

User-based content and services arrangement via rich and usable Web 2.0 interfaces (e.g., Google Maps Mashups, Yahoo Pipes...)

+ Extremely usable and user-friendly

- Nearly static approaches

- Vertical solution (web-based services & contents)
Service composition in Ubiquitous scenarios

Requirements in Ubiquitous Computing

- novel services, contents, communication channels $\rightarrow$ novel services
  - openness/extendibility
- users are the main target of service and content delivery
  - user friendliness
- scalability: reasonable number of solutions in a limited time
  - no exhaustive/inferential approaches
Key principles of our proposal

- **semantics** to convey metadata
  - strutturazione a layer, per gradi di astrazione
  - utenti vedono solo i livelli a maggior astrazione
  - nessun formato semantico specifico

- **modello formale** del meccanismo

- approccio algoritmico e **non inferenziale**
Ubiquitous service composition model

Our approach

- translucent and semantics-based to contemporarily
  - hide complexity and heterogeneity to users via a layered approach
  - keep a SOA-based flexible and extendible approach for automated semantics-based service composition

- formal composition model

- algorithmic and non-inferential approach
Business logic layer

Services
- basic application logic units
- e.g.: content retrieval, transformation, adaptation, dispatch

Workflows
- activity flow description
- aggregation and coordination of services
User semantics layer

Service metadata
- semantic data about the nature of application services
- high-level description of service features
  → service selection logic

Templates
- service aggregation models
- abstract workflows
  → service coordination logic

Interaction patterns
- interaction with templates
  → service interaction logic

Semantics fusion layer
- Rules
- Roles
- Semantic domains

Business logic layer
- Services
- Workflows

Service metadata
- Templates
- Interaction patterns

User semantics layer
Semantics fusion layer

Semantic domains
- group features of a certain service description area (e.g., syntax, behavior, ...)
- dynamically extensible

Rules
- drive translations from templates to concrete workflows

Roles
- reuse of rules for
  - different templates
  - different nodes within a template
Service composition in action

User requirements
- which (kind of) services/contents → semantic service metadata
- how to arrange them → composition templates
- how to interact with them → interaction patterns

Service composition
- translates user requirements into **concrete workflows**
- relies on **currently available** services

User requirements
- templates
- service metadata
- interaction pattern
A typical example – user perspective

1. Choice of an Aggregation template from a template catalogue

\[ R_{p2} = \{ \text{generator} \} \]
\[
\text{generator.behavior.typology} = \text{“generation”}
\]

\[ R_{p5} = \{ \text{aggregator} \} \]
\[
\text{generator.behavior.typology} = \text{“aggregation”}
\]
A typical example – user perspective

1. Choice of an Aggregation template from a template catalogue

2. Choice of an interaction pattern

\[ R_{p2} = \{ \text{generator} \} \]
\[ \text{generator.behavior.typology} = \text{“generation”} \]

\[ R_{p5} = \{ \text{aggregator} \} \]
\[ \text{generator.behavior.typology} = \text{“aggregation”} \]

\[ R_{p1} = \{ \text{userInput} \} \]

\[ R_{p6} = \{ \text{userOutput} \} \]
A typical example – user perspective

1 Choice of an Aggregation template from a template catalogue

2 Choice of an interaction pattern

3 Choice of kinds of services by means of semantic metadata

\[ R_{p2} = \{ \text{generator} \} \]
\[ \text{generator.behavior.typology} = \text{“generation”} \]
\[ p2.generationDomain.contentType = \text{RSS} \]

\[ R_{p5} = \{ \text{aggregator} \} \]
\[ \text{generator.behavior.typology} = \text{“aggregation”} \]

\[ R_{p1} = \{ \text{userInput} \} \]
\[ \text{userInput.interactionDomain.inputChannel} = \text{SMS} \]

\[ R_{p4} = \{ \text{generator} \} \]
\[ \text{generator.behavior.typology} = \text{“generation”} \]
\[ p4.generationDomain.contentType = \text{HTML} \]

\[ R_{p6} = \{ \text{userOutput} \} \]
\[ \text{userOutput.interactionDomain.outputChannel} = \text{phoneCall} \]
A typical example – system perspective

Aggregator output == XML
Phone call service input == audio/wav

NO MATCH!
A typical example – system perspective

Aggregator output == XML
XML-2-Voice input == XML
XML-2-Voice output == audio/wav
Phone call input == audio/wav
MATCH!

Middleware
Kernel layer
Application layer
Non-functional layer

Service composition
System reconfiguration

September 17th, 2009
System Composition and Reconfiguration in Ubiquitous Computing scenarios
SYSTEM
RECONFIGURATION
Rationale

In Ubiquitous scenarios, frequent changes in
- user requirements
- service availability
- operating conditions

Systems not only need to dynamically compose business logic

Need to support reconfiguration of
- non-functional (support) logic
- application logic
State of the art

Reflective middleware
- first approaches in system reconfiguration
- reflective middleware as a set of support components that hide heterogeneity
- in addition, applications built on top can runtime inspect and reconfigure middleware components

Main issues:
- application-driven reconfiguration
  how to deal with conflicts (e.g., different applications that require incompatible reconfiguration actions)
- no really structured approach to reconfiguration
- reflection comes at a price…
State of the art

Autonomic computing

Systems able to provide self-managing properties:

- **self configuration**: automatic system configuration according to high-level goals

- **self optimization**: automatic system/resources optimization

- **self healing**: problem detection and reaction

- **self protecting**: protection against malicious attacks or human errors
State of the art

Autonomic computing

IBM MAPE-K reference model stages:
- Monitoring
- Analysis
- Planning
- Execution
  - Common reference Knowledge base

Sensors/effectors concretely act on managed resources resources
Example

RSS-to-VoIP: service coordination
- RSS content retrieval
- voice synthesis (e.g., RSS to Wav)
- phone call (e.g., Asterisk gateway)

In case of network congestion / high load → **Bottlenecks**!
- voice synthesis service
- Workflow execution non-functional feature

---

**Application layer**

- Workflow execution
- Service registry
- ... (non-functional layer)
- Naming
- Persistence

**Kernel layer**

- Service composition
- System reconfiguration
Current limitations

In Ubiquitous Computing, **heterogeneity** and **dynamic adaptation** impact both on

- **application** logic
- **non-functional** support logic

Current autonomic solutions

- **partially cope** with application logic reconfiguration
- **do not face** non-functional logic reconfiguration
- lack **clear distinction** between logic types

Need for a middleware platform that supports reconfiguration of
- **application logic**
- **non-functional logic**
Reconfiguration - Main features

**Monitoring** features
- keep track of system current status

Reconfiguration **policy** management features
- determine *when* and *how* to reconfigure the system

Reconfiguration **enactment** features
- take actions to concretely reconfigure the system
Reconfiguration policies

Policies determine

- **what** to reconfigure
  - service/component properties
  - whole service/component
  - entire service composition (workflow)

- **when** to reconfigure
  - proactively vs. reactively

- **how** to reconfigure: different strategies depending on what to reconfigure; e.g.:
  - service/component substitution vs. multiple instance pooling
  - workflow invalidation and rebuild vs. runtime reconfiguration
System monitoring

Monitoring features
- crosscut main logic and span over different services/components
- need not be bound to a specific component → modularity

Aspect-oriented approach
- monitoring facilities “decorate” main logic
- modular approach

Service monitoring
- pluggable aspects to monitor different characteristics (e.g., QoS, availability,…)
- collects monitoring data from distributed components
Reconfiguration enactment

Reconfiguration enactment

- manages both
  - Application-level services
  - Non-functional services
- triggers service reconfiguration
  - service composition allows to runtime determine novel suitable compositions
Case study - runtime

Traffic news
- over SMS channel
- over PSTN network

Realization scenario
- different workflows realize **portions** of the scenario
- workflows can communicate via **asynchronous messages**

**Monitoring** keeps track of **non-functional** component status

**Enactment** keeps track of **application** service status
Case study - reconfiguration

Novel workflow instances
→ Performance loss

Bottlenecks
- voice synthesis service (WAV)
- asynch messaging (JMS-based)

Reconfiguration engine
- substitutes the Asynch Messaging component with an RTI-based implementation

Service composition engine
- reconfigures workflows
- substitutes WAV synthesis service with MP3 synthesis
CONCLUSIONS
Ubiquitous Computing scenarios

Ubiquitous Computing stresses the need for:

- **dynamic extension and heterogeneity**
  - need to cope with novel contents/services, devices, user interaction styles, ...

- **dynamic tailoring and adaptation**
  - services/contents and the middleware itself need to adapt to ever-changing operating conditions & user requirements
Disappearing Middleware

Middleware as a solution for Ubiquitous support
- hides **heterogeneity**
- disappearing approach
  - **minimal kernel** layer
    - composition features
    - reconfiguration features
  - novel features **plugged in** by need
Reconfigurable and Composable Middleware will be the basis for modern distributed applications, to let them

- **extend** (and also shrink) when in need
- **adapt** to changing conditions

Disappearing approach as the only viable solution when complexity, uncertainty, and flexibility rule the target scenarios.
Questions?

www.swimm.it
stefano.monti6@unibo.it
References

Service Composition


System reconfiguration