Specification-Carrying Code for Self-Managed Systems

Giovanna Di Marzo Serugendo
University of Geneva, Switzerland
Outline

• Semantic Infrastructure
  – « Specification-Carrying Code » (SCC)
  – Service-oriented architecture

• SCC for Autonomic Computing
Specification-Carrying Code

• Idea: communication is based on a formal specification of the behaviour of a peer entity
  – Software « carries » a formal description of its own functional behaviour
  – Communication occurs without API
  – Formal specification defines the semantics of the behaviour
SCC - Principle

Register

\[
\Sigma_1 \quad \Sigma_2 \\
\text{Ax}_1 \quad \text{Ax}_2 \\
\cdots
\]

Request

Thm Checker

\[
\{ \Sigma_i \mid \Sigma_i \vdash \Sigma \}
\]
**SCC - Architecture**

- **Register**
- **Execute** *(SpecS)*
- **Search** *(SpecS)*
- **Execute** *(ArrayList)*
- **ArrayList’**
- **Code**
- **Entity**
- **Service**

- **RegEx**
- **Prolog**
- **HOL**
SCC – Prolog

• Registration

<specs>
   <description active="true">
      <content> Reverse List Service 
   </content>
   </description>
   <prolog active="true">
      <content>
         append([],L,L).
         append([H|T],L2,[H|L3]):-
            append(T,L2,L3).
         rev([],[]).
         rev([H|T],R) :- rev(T,RevT),
            append(RevT,[H],R).
      </content>
   </prolog>
</specs>

• Request

<specs>
   <description active="true">
      <content> ReverseList Request 
   </content>
   </description>
   <prolog active="true">
      <content>
         rev([],[]), rev([A|B],R), rev(B,RevB),
         append(RevB,[A],R), rev(R,[A|B]).
      </content>
   </prolog>
</specs>
SCC – Java (no API!)

• Registration

```java
public class ReverseList extends Service {

    public class static void main(String[] args) {
        // register reverse list specification
        new ReverseList().register("localhost", "specService.xml");
    }

    public ArrayList execute(ArrayList list) {
        Collections.reverseList(list);
        return list;
    }
}
```

• Request

```java
public class UseReverseList extends Entity {

    private void askForReverseList() {
        // request a reverse list service
        result = Entity.execute(SM_ADDRESS, "specRequest.xml", parameters);
    }

    public ArrayList execute(ArrayList list) {
        Collections.reverseList(list);
        return list;
    }
}
```
SCC - Advantages

• Interest
  – Minimum basis for communication
    • Specification language (for expressing concepts)
  – Interaction/Interoperability with new/unknown software
    • No common design / No common API
  – Self-assembly
  – Seamless Integration of new entities
  – Robustness
SCC for Autonomic Computing

- SCC expresses
  - Functional Behaviour
  - Non-Functional Aspects
    - Policies
    - Trust
    - Quality of Service
  - Execution Flow
SCC for Autonomic Computing

- Self-Configuration (installation, configuration, integration)

“Automated configuration of components and systems follow high-level policies. Rest of System adjusts automatically and seamlessly [Kephart03]”

- SCC expresses high-level configuration policies
  - High-level requests (goals) from human admin (installation needs)
  - High-level requests for configuration policies (Grid distribution)
  - Local-level: components express individual installation needs (CPU, memory, etc.)

- Unanticipated dynamic run-time evolution of code
  - Seamless integration of new components
  - Distribution of application on-the-fly
SCC for Autonomic Computing

• Self-Optimisation (parameters)

“Components and systems continually seek opportunities to improve their own performance and efficiency [Kephart03]”

– SCC expresses optimisation policies
  • Parameters description
  • Permanent optimisation of parameters depending on the context

– At each request
  • SCC Middleware seeks optimised service (most recent, most efficient, etc.)
SCC for Autonomic Computing

• Self-Healing (error detection, diagnostic, repair)

“System automatically detects, diagnoses, and repairs localized software and hardware problems [Kephart05]”

– Generation of correct code from SCC
– Replace error code with code having matching specification
– Checking of code against specification
SCC for Autonomic Computing

• Self-protection (detection and response to attacks)

“System automatically defends against malicious attacks or cascading failures. It uses early warning to anticipate and prevent systemwide failures [Kephart05]”

– SCC expresses high-level security policies
  • Conditions regulating services delivery
  • Signatures of attacks / Response schema

– Self-regulating schema
  • Trust and reputation information
Conclusion

• SCC
  – Specifications of behaviour
  – Implementation through a middleware infrastructure
  – Interoperability solution
  – No need for compatible interfaces

• SCC for Self-Managed Systems
  – Functional properties
  – Non-functional properties
  – Run-time (re)configuration policies/schemas
  – Run-time description of interaction protocols