

A Service-Oriented Infrastructure for Adaptive Systems Based on Specification-Carrying Code

Giovanna Di Marzo Serugendo
dimarzo@dcs.bbk.ac.uk

Birkbeck College, University of London
<http://www.dcs.bbk.ac.uk/~dimarzo>

Outline

- Framework
- « Specification-Carrying Code » (SCC)
- Applications
 - Unanticipated code evolution
 - Autonomic Computing
 - Dynamically Resilient Systems

Framework (1)

- Applications Domains
 - Wireless / Ad hoc Networks
 - Grid
 - Ambient Intelligence
 - Autonomic Computing
 - Large Scale Security Systems

Framework (2)

- Characteristics
 - Large scale open distributed systems
 - Autonomous and heterogeneous entities
 - Decentralised control
 - Large number of components

Framework (3)

- Characteristics
 - Dynamic and uncertain environment
 - Need for adaptability
 - Social dimension
 - Interactions, discovery, negotiations, transactions
 - Self-organisation (adaptability/robustness)
 - Emergent Phenomena

Issues

- **Interactions with unknown entities (semantics)**
 - Understanding
 - Interoperability
- Management of uncertainty (social)
 - Malicious entities (exhibit desirable characteristics, but ...)
 - Good faith entities (fail because: software error, lack of toner, paper jam, ...)
- **Design** of systems with decentralised control
 - How to realise self-organisation?
 - How to realise self-managing systems?
- Control of emergent phenomena
 - Desired or undesired, but with a causal effect on system

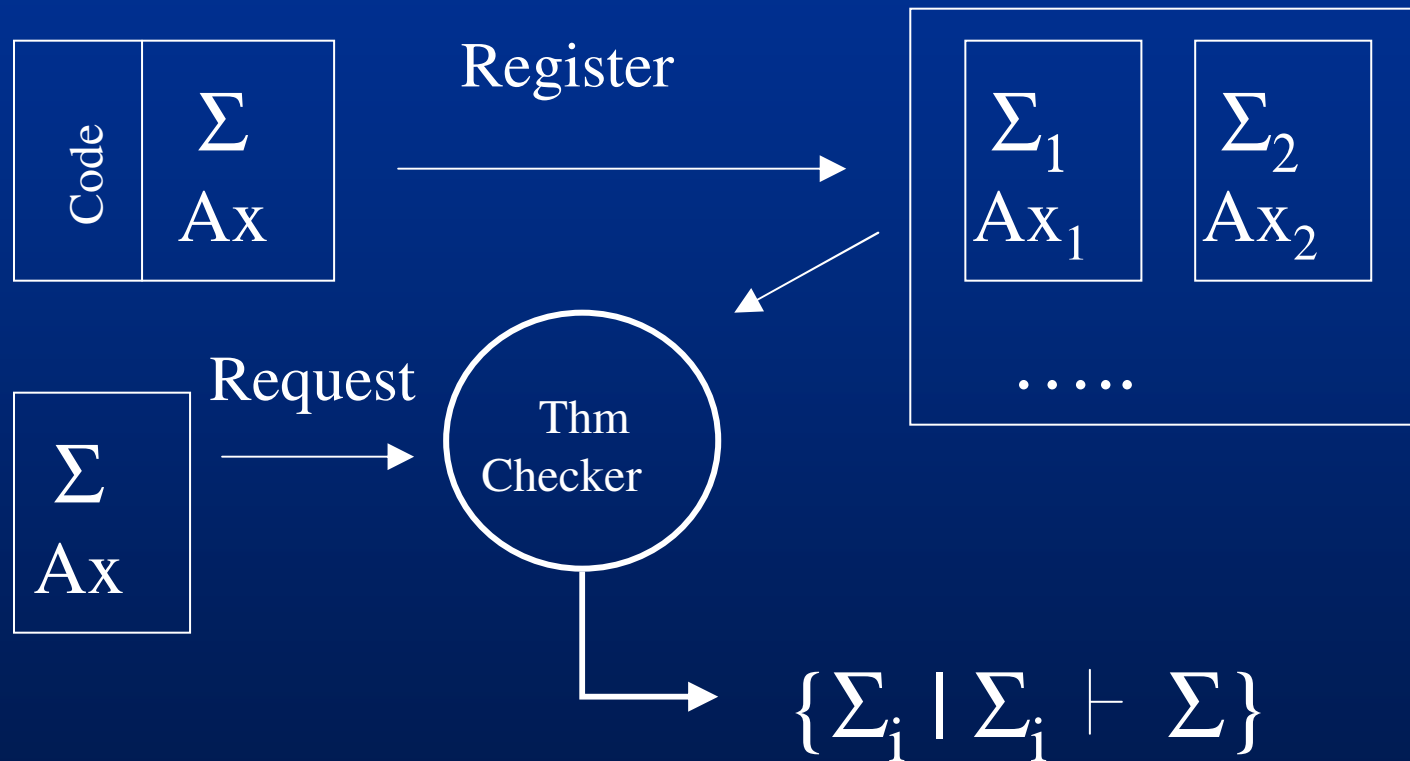
Specification-Carrying Code

- 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

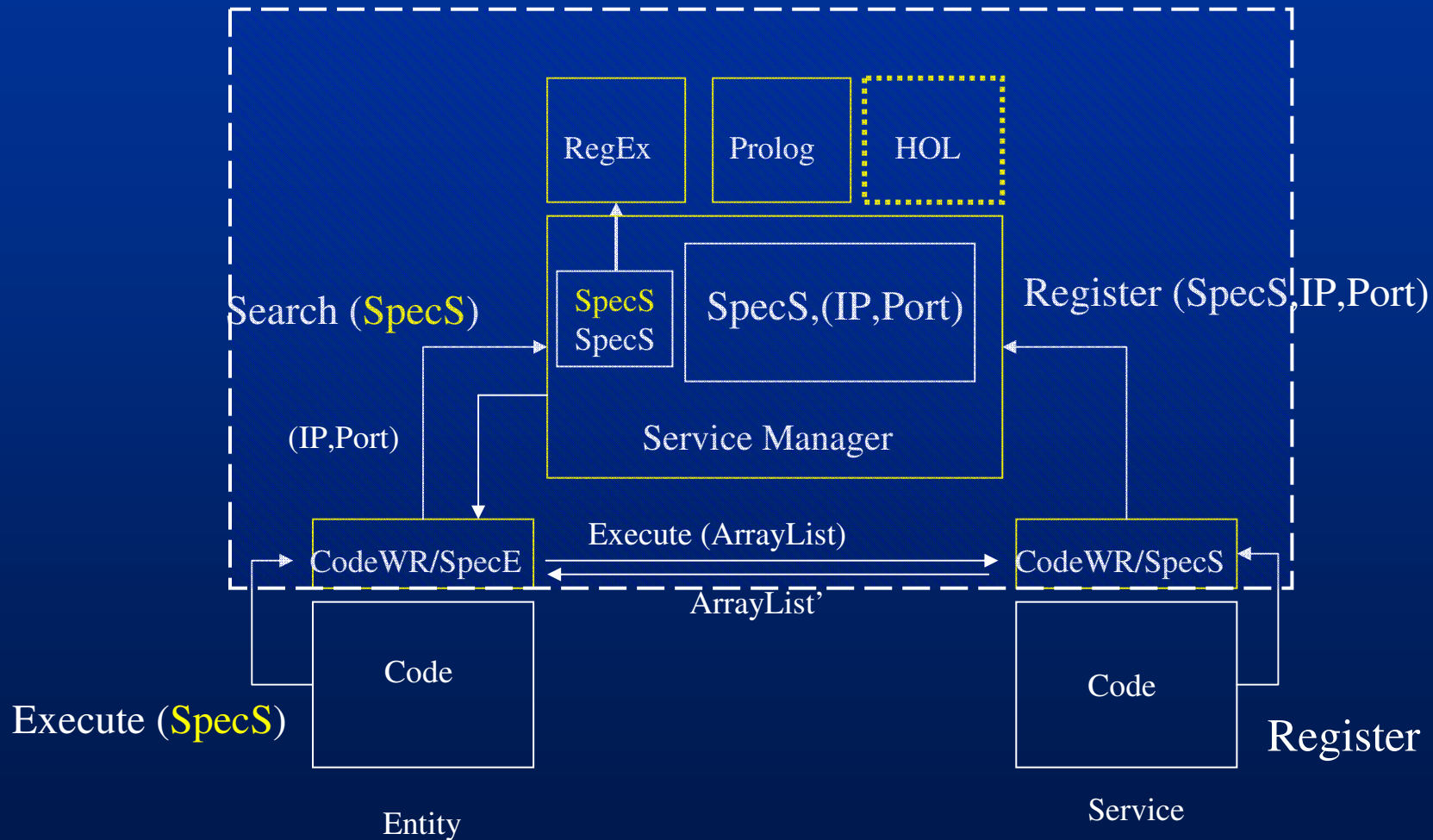
SCC - Principle

- Scenario
 - Publication of specifications
 - Services requested / Services proposed
 - Specification matching
 - Proposed service matches requested service
 - Service realised in an anonymous / asynchronous / non-deterministic manner
- Interest
 - Minimum basis for communication
 - Specification language (for expressing concepts)
 - Interaction with new software / with unknown software
 - No central control (self-assembly)

SCC - Principle



SCC - Architecture



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

```
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

```
public class UseReverseList extends Entity {  
  
    private void askForReverseList() {  
        // request a reverse list service  
        result = Entity.execute(SM_ADDRESS,  
        « specRequest.xml », parameters);  
    }  
}
```

SCC – Alternatives

- Specification
 - ✓ Keywords
 - ✓ Regular Expressions (syntactic)
 - ✓ Prolog (SWIProlog)
 - HOL (Isabelle Thm Prover – meta-ontology)
 - Jena (Logic + ontology)
 - Common Simple Logic
- Architecture
 - ✓ Publication of specifications (asynchronous / anonymous / non-deterministic)
 - Direct exchange of specifications (interaction decisions)
- Service Discovery
 - JXTA protocols
 - Géo-positioning
- Information contained in the specification
 - Functional
 - Non-functional, security, reputation, positioning, etc,

SCC vs PCC vs Trust

- SCC
 - Code is decoupled from specification
 - No guarantee that the code satisfies the specification
 - It is the same with APIs!
- Proof Carrying Code (PCC) [Necula04]
 - Code « carries » the proof that it is correct
 - Low level (no infinite loop, no division by zero)
 - Not at the functional level
 - No specification
 - What happens if the code/proof are malicious?
 - What happens if the code/proof are in good faith, but the code fails?
- Trust
 - Adaptation mechanism based on experience and observation

SCC – Interest and Issues

- No central control
 - Spontaneous registrations and requests
- Minimum basis for communication
 - Specification language (for expressing concepts)
- Interaction/Interoperability with unknown (or heterogeneous) peers
 - No common design / No common API
- Self-assembly
- Seamless Integration of new entities

SCC – Interest and Issues

- Robustness
- Service Composition
- Adaptability
 - Feedback available among services through specifications (modifications)
- Services can be:
 - Computation / data provider / manager
- Issues
 - Specification Language
 - Expressivity vs run-time processing
 - Matching tool

Applications

- Unanticipated Evolution of Code
- Autonomic Computing
- Dynamically Resilient Systems

Unanticipated Evolution of Code

- Code changes during its execution (without stopping the application)
- Non anticipated evolution
 - Non anticipated by the programmer
- Distribution on the fly
- Experiments
 - Web Server
 - 160 different versions of the server, with only 4 stops
 - Tic-Tac-toe for Open Days
 - Changes done to the application *during the play*

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

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.)

Autonomic Computing

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

“System automatically detects, diagnoses, and repairs localized software and hardware problems [Kephart03]”

- Generation of correct code from SCC
- Replace error code with code having matching specification
- Checking of code against specification

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 [Kephart03]”

- SCC expresses high-level security policies
 - Conditions regulating services delivery
 - Signatures of attacks / Response schema
- **Self-regulating schema**
 - Trust and reputation information

Autonomic Computing

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

Dynamically Resilient Systems

- Resilience
 - Capacity to resist to impairments
- Dynamic resilience
 - Dynamic adaptation to changing environment
 - Maintain an acceptable level of quality of service
- Predictable dynamic resilience
 - Dynamic resilience predictable at design time

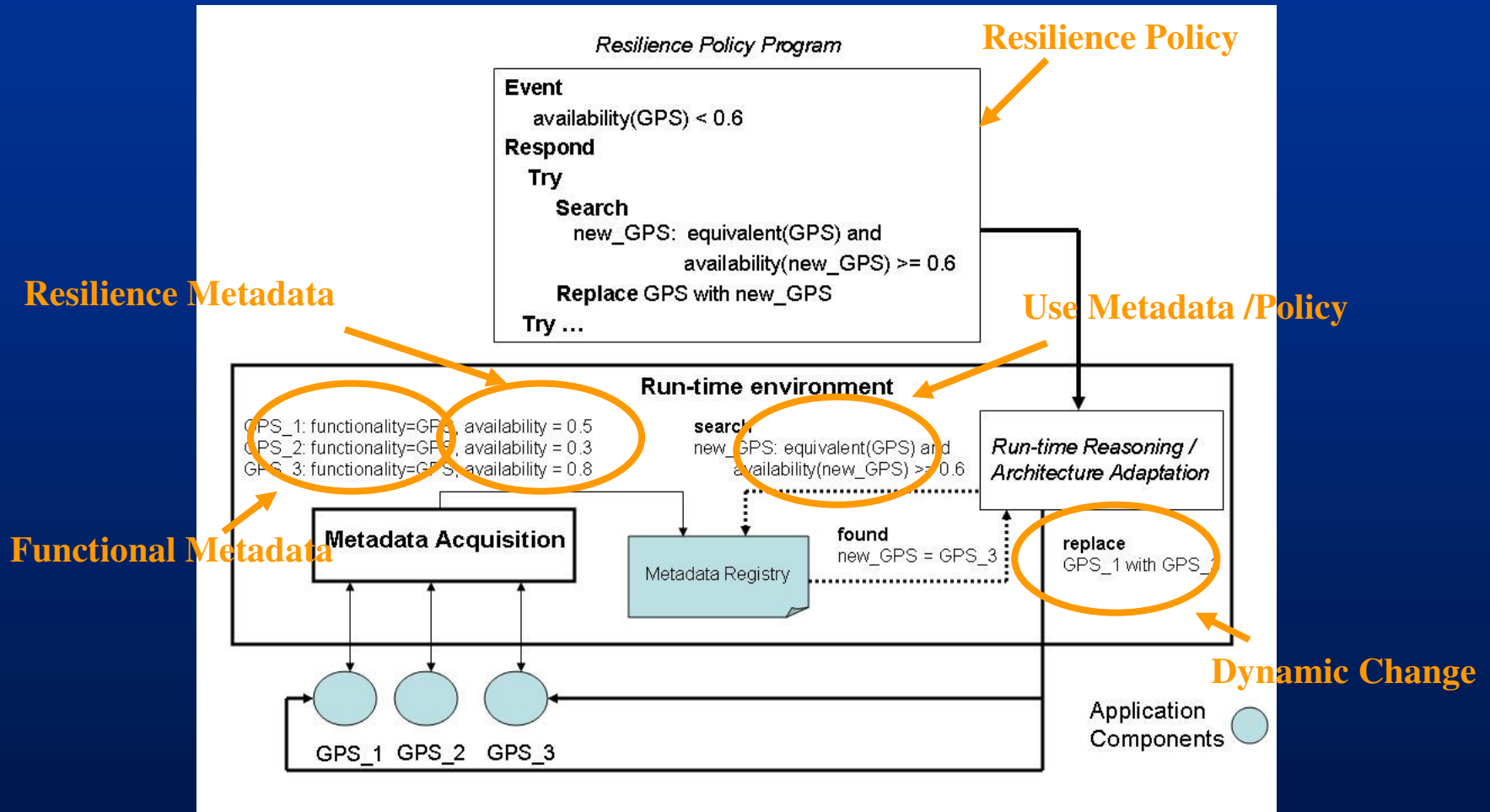
Dynamically Resilient Systems

- Metadata
 - Functional
 - Non-functional (QoS): permanently updated
 - Resilience metadata: permanently updated
- Resilience mechanism (design time)
 - “Design pattern for resilience”
- Resilience policies (run-time)
 - Policy used at run-time to enforce resilience
 - Formal language
- Reasoning and adaptation services
 - Run-time environment
 - Enforce resilience policies (matching)
 - Use metadata information
 - Adapt service (change)

Dynamically Resilient Systems

- GPS Example
 - System with 3 GPS
 - Resilience mechanism
 - “Switch to GPS with higher availability if current GPS has availability lower than 0.6”
- Metadata
 - Functional: “Functionality = GPS”
 - Non-functional resilient: “Availability”

Dynamically Resilient Systems



Other applications

- Grid computing
 - Resource allocations
 - Load balancing
- Digital Right Managements
 - “Who is allowed to view/modify/use digital document ?”
 - Policy specifies “who/when/what”
 - Dynamic change of policy at run-time
 - Modification of rights on the fly
- E-government
 - Law requirements → Policies
 - Change of law → Change of policy + No-change of code

Summary

- Goal
 - Design self-* systems with predictable behaviour
- Idea
 - Use of formal specifications / metadata at run-time:
 - Functional behaviour
 - Non-functional constraints
 - Rules/Policies/etc.
- Realisation
 - Run-time service-oriented middleware
 - Enforcing functional behaviour / constraints / policies
 - Dynamic changes of components
 - On the basis of formal specifications and metadata at run-time

Additional Activities

- IEEE SASO Conference
 - <http://projects.csail.mit.edu/saso2007/>
- Agentlink Technical Forum on Self-Organisation
 - <http://www.agentlink.org/activities/al3-tf>
- IEEE ETTC Organic Computing Task Force
 - <http://www.neuroinformatik.ruhr-uni-bochum.de/PEOPLE/igel/oc.html>
- ACM Transactions on Autonomous Adaptive Systems
 - Inaugural issue: September 2006
 - <http://www.acm.org/pubs/taas>

Papers

- G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, N. Guelfi. "*A Metadata-Based Architectural Model for Dynamically Resilient Systems*", ACM Symposium on Applied Computing (SAC'07) - Dependable and Adaptive Distributed Systems (DADS), 2007.
- G. Di Marzo Serugendo, M.-P. Gleizes, A. Karageorgos. "*Self-organisation and emergence in MAS: an overview*", Informatica 30(1): 45-54, Slovene Society Informatika, Ljubljana, Slovenia, 2006.
- M. Oriol, G. Di Marzo Serugendo, "*A Disconnected Service Architecture for Unanticipated Run-time Evolution of Code*", IEE Proceedings-Software, Special Issue on Unanticipated Software Evolution, Susan Eisenbach (Ed), 2004.
- M. Oriol: "*An Approach to the Dynamic Evolution of Software Systems*". PhD in Information Systems. Faculty of Economic and Social Sciences, University of Geneva, 2004.

References

- George C. Necula, “*Enforcing Security and Safety with Proof-Carrying Code*”.
Electr. Notes in Theoretical Computer Science 20(1):1-15, 2004
- Jeffrey O. Kephart, David M. Chess, “*The Vision of Autonomic Computing*”.
IEEE Computer 36(1): 41-50, 2003