Architecture-based QoS Prediction for Service-oriented Computing

Vincenzo Grassi, Raffaela Mirandola

Università di Roma “Tor Vergata”, Italy

Service-oriented Computing

- emerging paradigm for designing, architecting and delivering distributed applications
  - applications built as a composition of Internet accessible, independently developed and delivered “services”
  - “service”: unit of composition, spans high level functionalities (some complex business logic) and basic functionalities (processing, storage, …)
- strong overlapping with component-based approaches
  - distinguishing feature: **automatic** service advertisement, discovery and composition
    - need of agreed on and machine-processable service description languages
    - need of automatic discovery, selection and composition tools

QoS-driven service selection and composition

- Non obvious correlation between service assembly QoS and individual services QoS
  - assembly QoS ***monitoring*** to assess the fulfillment of some QoS goal, **after** the service selection and composition

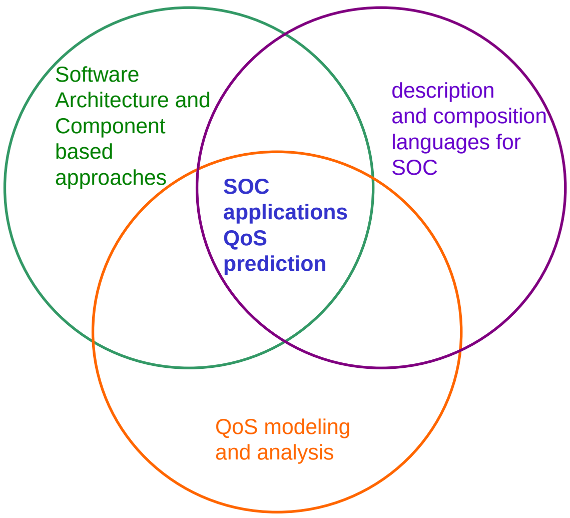
  - assembly QoS ***prediction*** **to drive** the selection of services

need of QoS prediction methodologies

- ***compositional*** (to exploit the SOC application structure)
- ***automatic*** (to be compliant with the SOC requirements)

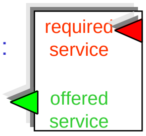
Compositional and automatic QoS prediction

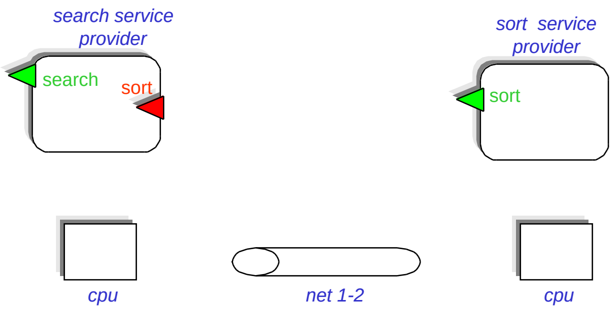
- Contributions from different areas and communities



Example

- “search an item in a list" service
  - can require a "sort" service if the list is not ordered
- symbols :





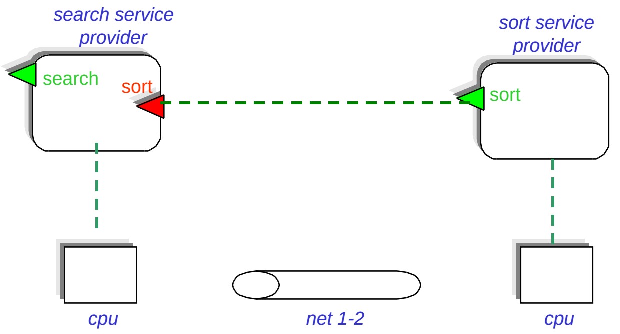
Contributions from each area (1)

- description and composition languages for SOC
  - built on top of basic XML-based languages and protocols (WSDL, SOAP, UDDI)
  - examples
    - OWL-S (formerly DAML-S): promoted by BBN Technologies, Nokia, and several academic institutions (CMU, Stanford, USC, MIT, Vrije Univ., …)
    - BPEL4WS (formerly WSFL and XLANG): promoted by BEA, IBM, Microsoft, SAP AG, Siebel Systems
- main features
  - machine-processable and interoperable
  - support the definition of non functional properties (QoS)

  but …

  - no explicit description of the "interaction infrastructure"
  - QoS values mainly expressed as absolute values (no platform dependent parameterization)
  - lack of support for compositional analysis

Example



Contributions from each area (2)

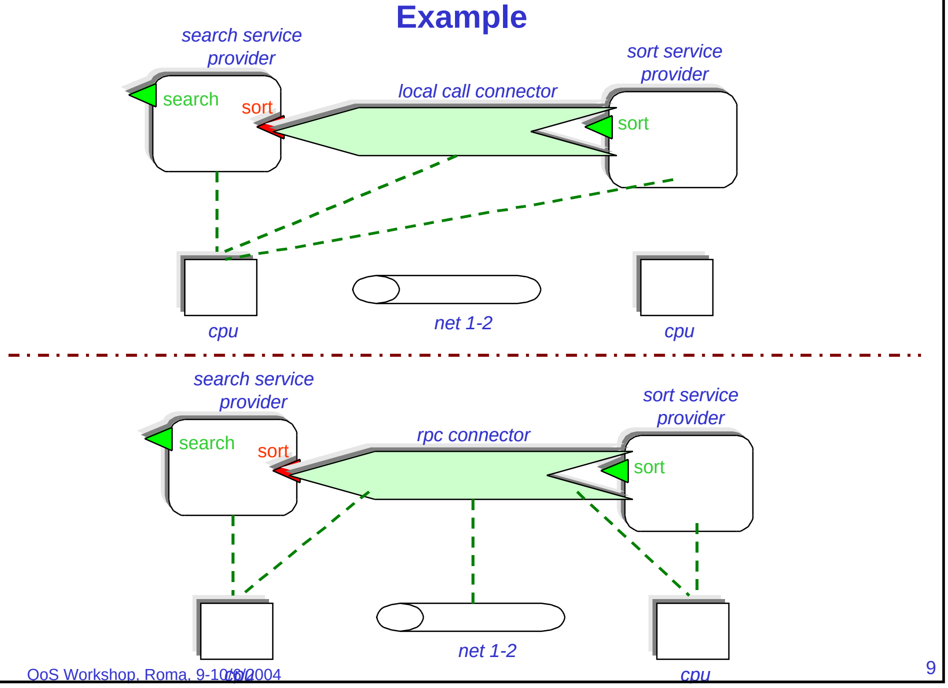
- Software Architecture and Component based approaches

- main features
  - the "interaction infrastructure" is a first class concept
    - ***connector*** concept
  - explicit consideration of dependencies between offered and required services
  - attention given to non functional properties (QoS)

  but …

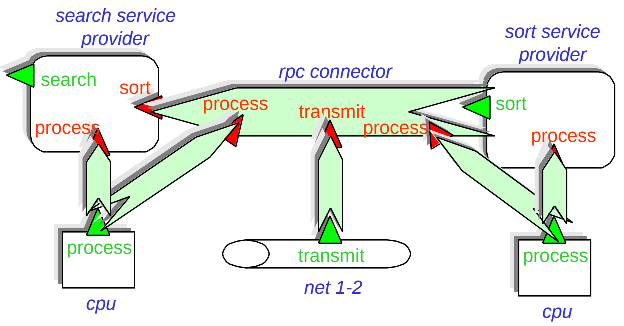
  - several (too many?) "experimental" architecture description languages (ADLs)
    - some unification/interoperation effort
  - need of a better integration of QoS analysis techniques
    - non well defined “QoS semantics” for existing ADLs

Example



SOC & Architecture-based QoS prediction (1)

- need of a **QoS language** for SOC
  - machine-processable
  - integrated with existing SOC languages
- proposed approach: unifying "service+connector" model
  - for both "high level" and "low level" services
    - more flexibility
    - simpler description language definition



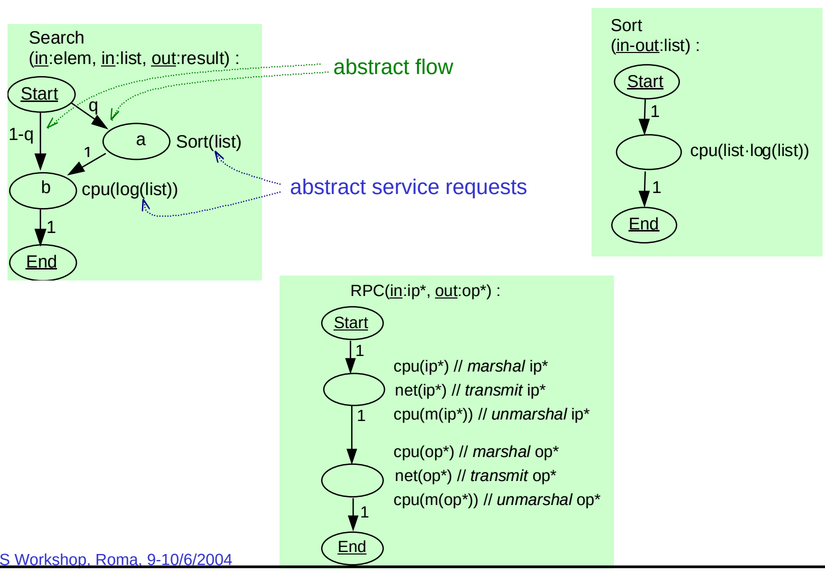
SOC & Architecture based QoS prediction (2)

- “analytic interface” associated with each offered service
  - general concept proposed by CMU-SEI (PECT: Prediction Enabled Component Technology)
  - suitable abstraction of the “constructive (functional) interface”
  - allows a structured approach to compositional analysis

- in our approach:
  - consider services offered by both resources (components) and connectors
  - “abstract” service representation
    - abstract service description
      - abstract parameter domains
    - (for non basic services) abstract service request flow addressed to other resources/connectors: stochastic model
      - abstract flow: probabilistic graph
      - abstract service request: actual parameters as (parametric) random variables

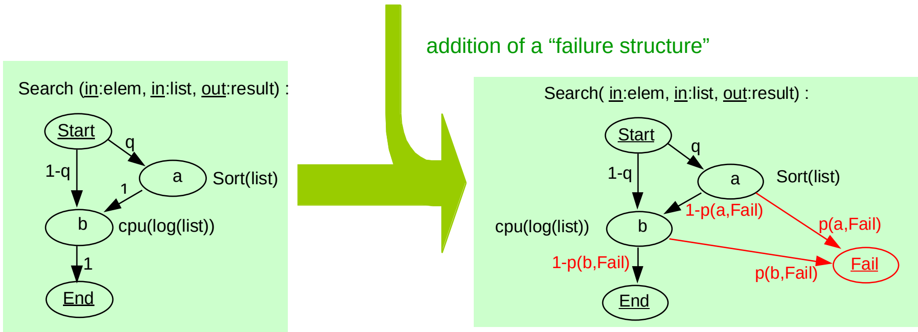
Example

- abstract request flows of the Search, Sort, and RPC services



QoS (reliability) prediction

- the presented concepts provides the (mainly “linguistic”) support for QoS compositional prediction
- addition of QoS related information (possibly specialized for some QoS dimension, e.g. dependability) with well defined semantics
  - example: composite service **reliability** analysis



  - reliability = probability of reaching the End state
  - crucial issue: evaluation of p(***node***, Fail)

“Reliability semantics” issues (1)

- ***node*** of a service request flow graph: collection of service requests

node = {R1, R2, …, R***n***}, where:

R***j*** = ***request***(***Sj***, ***apj****) ***Sj*** = required service specification

apj* = list of actual (abstract) parameters

node failure probability: depends on :

- failure probability of each R***j***
- completion model for R1, R2, …, R***n***
  - ***AND***, ***OR***, ...
- dependencies among R1, R2, …, R***n***
  - no dependence (e.g. no service sharing), dependence (e.g. service sharing)

failure probability of R***j*** depends on :

- ***internal*** failure prob for R***j*** (***Pfail_int***(R***j***)) (definition?)
- ***connector*** failure prob for R***j*** (***Pfail_connect***(R***j***))
- ***service*** failure prob for R***j*** (***Pfail_service***(R***j***))

Pfail_int(R***j***) × ***Pfail_connect***(R***j***) × ***Pfail_service***(R***j***) **?**

“Reliability semantics” issues (2)

- Ri = **request**(Si, api*)
  Rj = **request**(Sj, apj*)
  - what if Si = Sj ? (I.e., the two requests are connected to the same service S)
- failure prob {Ri} = **Pfail_int**(Ri) × **Pfail_connect**(Ri) × **Pfail_service**(Ri) ?
  failure prob {Rj} = **Pfail_int**(Rj) × **Pfail_connect**(Rj) × **Pfail_service**(Rj) ?

OK

Ri = **request**(S, api*)   Rj = **request**(S, apj*)

flow graph node: ***AND*** completion model

NO

Ri = **request**(S, api*)   Rj = **request**(S, apj*)

flow graph node: ***OR*** completion model

Conclusions

issues for QoS prediction in a SOC framework

- inclusion of a well structured "analytic interface" into existing XML-based service description and composition languages
  - based on concepts from Software Architecture approaches (connectors!)
- QoS semantics deserves special care
  - example: dependability analysis methodologies should not be based on a priori (prior to service composition) independence assumptions
    - service composition or F-T features can introduce dependencies among services
- reuse existing work on algorithmic methods for the automatic generation of QoS analysis models
  - mostly from UML models
  - idea: express the QoS semantics of XML-based SOC languages in terms of appropriate UML models
    - UML Profile for Modeling QoS and F-T