Architecture-based Dependability Prediction for Service-oriented Computing

Vincenzo Grassi

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)

In this work, focus on dependability issues
Compositional and automatic QoS (dependability) prediction (1)

- need of a **QoS language** for SOC
  - machine-processable
  - integrated with existing SOC languages
  - supporting compositionality

- built to express which concepts?
  - syntax
  - semantics
Compositional and automatic QoS (dependability) prediction (2)

- Contributions from different areas and communities

- Software Architecture and Component based approaches
- QoS (dependability) prediction for SOC applications
- QoS modeling and analysis
- Description and composition languages for SOC
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 (e.g. reliability)

  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

search service provider

search
sort

sort service provider

sort

cpu
net 1-2
cpu
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 (QoS) properties

  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
Contributions from each area (3)

- QoS modeling and analysis

- main features
  - analysis techniques
  - QoS specification languages
    - QML (Frolund - Koistinen, 1998), HQML (Gu et al., 2001), CQML (Aagedal, 2001), CQML+ (Rottger - Zschaler, 2003), …
    - UML QoS Profile

  but …

  - weak connection between QoS specification languages and QoS analysis techniques
  - unsatisfactory support for compositionality in existing QoS languages
Integration of contributions (1)

- a **QoS language** for SOC

- built around a unifying “service+connector” model
  - for both “high level” and “low level” services
    - more flexibility
    - simpler description language definition
Integration of contributions (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 (1)

- "abstract" service description:

  \[
  \text{Search}(\text{in } i : T; \text{in } l : \text{list of } T; \text{out } \text{res} : \text{boolean})
  \]
  
- "functional" description abstraction

  \[
  \text{Search}(l : \text{integer})
  \]

- "abstract" service request flow:

  \[
  \text{Search}(\text{in } i : T; \text{in } l : \text{list of } T; \text{out } \text{res} : \text{boolean})
  \]

  \[
  \{\text{if not_ordered}(l) \text{ then Sort}(l); \text{res} := \text{do_search}(i, l)\}
  \]

  "functional" description

  \[
  \text{Start} \quad \text{cpu(log(#list))} \quad \text{Sort(#list)} \quad \text{End}
  \]

  abstract flow

  abstract service requests

  abstraction
Example (2)

- abstract request flows of the Sort and RPC services

Sort (in-out:list):

Start → 1 → cpu(#list·log(#list)) → 1 → End

RPC(in:ip*, out:op*):

Start → 1 → cpu(ip*) // marshal ip*
    → 1 → net(ip*) // transmit ip*
    → 1 → cpu(m(ip*)) // unmarshal ip*
    → 1 → cpu(op*) // marshal op*
    → 1 → net(op*) // transmit op*
    → 1 → cpu(m(op*)) // unmarshal op*
    → 1 → End
Dependability prediction

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

- addition of a “failure structure”

  reliability = probability of reaching the End state
  crucial issue: evaluation of p(node, Fail)
“Dependability semantics” issues (1)

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

\[ \text{node} = \{R_1, R_2, \ldots, R_n\}, \text{ where:} \]

\[ R_j = \text{request}(S_j, ap_j^*) \]

\[ S_j = \text{required service specification} \]

\[ ap_j^* = \text{list of actual (abstract) parameters} \]

node failure probability: depends on :

- failure probability of each \( R_j \)
- completion model for \( R_1, R_2, \ldots, R_n \)
  - \( \text{AND}, \text{OR}, \ldots \)
- dependencies among \( R_1, R_2, \ldots, R_n \)
  - no dependence (e.g. no service sharing), dependence (e.g. service sharing)

failure probability of \( R_j \) depends on :

- internal failure prob for \( R_j \) \( (P_{\text{fail\_int}}(R_j)) \) (definition?)
- connector failure prob for \( R_j \) \( (P_{\text{fail\_connect}}(R_j)) \)
- service failure prob for \( R_j \) \( (P_{\text{fail\_service}}(R_j)) \)

\[ P_{\text{fail\_int}}(R_j) \times P_{\text{fail\_connect}}(R_j) \times P_{\text{fail\_service}}(R_j) \ ? \]
“Dependability semantics” issues (2)

- \( R_i = \text{request}(S_i, \text{api}^*) \)
  \( R_j = \text{request}(S_j, \text{apj}^*) \)
  - what if \( S_i = S_j \) ? (i.e., the two requests are connected to the same service \( S \))

- failure prob \( \{R_i\} = P_{\text{fail\_int}}(R_i) \times P_{\text{fail\_connect}}(R_i) \times P_{\text{fail\_service}}(R_i) \) ?
  failure prob \( \{R_j\} = P_{\text{fail\_int}}(R_j) \times P_{\text{fail\_connect}}(R_j) \times P_{\text{fail\_service}}(R_j) \) ?

flow graph node: AND completion model

- \( R_i = \text{request}(S, \text{api}^*) \) \( R_j = \text{request}(S, \text{apj}^*) \)

flow graph node: OR completion model
Conclusions

issues for dependability (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!)

- dependability (QoS) semantics deserves special care
  - example: dependability analysis methodologies should not be based on \textit{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