Behavioural Modeling of Services: from Service-Oriented Architecture to Component-Based System

Marek Rychlý

Brno University of Technology, Faculty of Information Technology
Department of Information Systems
Božetěchova 2, 612 00 Brno, CZ
http://www.fit.vutbr.cz/∼rychly/

CEE-SET 2008
October 13–15, 2008, Brno

Outline

- **Introduction**
  Service-Oriented Architecture (SOA)
  Component-Based Development (CBD)
  A Calculus of Mobile Processes (π-Calculus)

- **Behavioural Modeling of Services**
  Services in SOA
  Services in CBD

- **Summary and Future Work**

Service-Oriented Architecture (SOA)

Definition (Service-Oriented Architecture)

SOA represents a model in which functionality is decomposed into small, distinct units (services), which can be **distributed** over a network and can be combined together and reused to create **business applications**.

(Thomas Erl, SOA: Concepts, Technology, and Design, 2005)

Services can communicate:

1. by passing data between two services,
   (**service contracts**, services receiving the data are **requesters**, while services sending the data are **providers**)
2. by coordinating an activity between two or more services.
   (a multi-party collaboration between services that is usually known as **service choreography**)

Levels of Abstraction in SOA

SOA can be described at three levels of abstraction:

1. **business processes**
   (a system is a hierarchically composed business process, represents sequence of steps in accordance with some business rules leading to **a business aim**)

2. **services**
   (an implementation of **a business processes** and their parts with well-defined interfaces and interoperability for the benefit of the business)

3. **components**
   (an implementation of **a service** as component-based systems with well-defined structure and description of its evolution for the benefit of the implementation)

Component-Based Development (CBD)

Definition (Software Component)

A software component is a unit of composition with contractually specified **interfaces** and explicit **context dependencies** only. It can be deployed independently and is **subject to composition** by third parties.

(Clemens Szyperski, Component Software: . . . , 2002)

The components can be:

1. **primitive components**,
   (realised directly, beyond the scope of architecture description)
2. **composite components**.
   (decomposable on systems of subcomponents at the lower level)

The interfaces can be:

1. **functional interfaces**,
   (for business-oriented services required or provided by a component)
2. **control interfaces**,
   (for binding of interfaces and changing of behaviour and structure)
3. **reference interface**.
   (for passing of references to components or references to interfaces)

A Calculus of Mobile Processes (π-Calculus)

- Algebraic approach to description of a system of concurrent and mobile processes.
- Two concepts: **agents** (communicating processes) and **names** (communication channels, data, etc.).

$\overline{x}\langle y\rangle.P$	output prefix
$x(z).P$	input prefix
$\tau.P$	unobservable prefix
$(z)P$	restriction of scope
$P + Q$	sum of capabilities of processes
$P \mid Q$	composition of processes
$!P$	an infinite composition of the process

$$\begin{array}{rcl} P & ::= & M \mid P \mid P \mid (z)P \mid !P \\ M & ::= & 0 \mid \pi.P \mid M + M \\ \pi & ::= & \overline{x}\langle y\rangle \mid x(z) \mid \tau \end{array}$$

Reduction, Abstraction and Application

Communication defined as a **reduction relation** $\rightarrow$, the least relation closed under a set of the reduction rules.

$$\textsc{R-Inter}\ \frac{}{(\overline{x}\langle y\rangle.P_1 + M_1) \mid (x(z).P_2 + M_2) \rightarrow P_1 \mid P_2\{y/z\}} \qquad \textsc{R-Tau}\ \frac{}{\tau.P + M \rightarrow P}$$

$$\textsc{R-Par}\ \frac{P_1 \rightarrow P_1'}{P_1 \mid P_2 \rightarrow P_1' \mid P_2} \qquad \textsc{R-Res}\ \frac{P \rightarrow P'}{(z)P \rightarrow (z)P'}$$

$$\textsc{R-Struct}\ \frac{P_1{=}P_2 \rightarrow P_2'{=}P_1'}{P_1 \rightarrow P_1'} \qquad \textsc{R-Const}\ \frac{}{K\lfloor\tilde{a}\rfloor \rightarrow P\{\tilde{a}/\tilde{x}\}}\ K \triangleq (\tilde{x}).P$$

- An **abstraction** of arity $n \geq 0$ is an expression of the form $(x_1, \dots, x_n).P$, where the x_i are distinct.
- A **pseudo-application** of an abstraction $F \overset{def}{=} (\tilde{x}).P$ is an expression of the form $F\langle\tilde{y}\rangle$, a process $P\{\tilde{y}/\tilde{x}\}$.
- A **constant application** of a process constant $K \triangleq (\tilde{x}).P$, is an expression of the form $K\lfloor\tilde{a}\rfloor$, reducible according rule R-Const. It allows **recursive definitions**.

Services in Service-Oriented Architecture

In the π-calculus, **a general service** *Service* with interfaces $i_1, \ldots, i_n$ can be described as a process abstraction

$$\begin{aligned} \mathit{Service} \quad & \overset{def}{=} \quad (i_1, \ldots, i_n).(s_1, \ldots, s_m) \\ & \qquad (\mathit{Svc}_{\mathit{init}}\langle i_1, \ldots, i_n, s_1, \ldots, s_m\rangle . \prod_{j=1}^{n} \mathit{Svc}_j \lfloor i_j, s_1, \ldots, s_m \rfloor) \end{aligned}$$

- The pseudo-application of $\mathit{Svc}_{\mathit{init}}$ initiates the service.
- The constant application of Svc_j interacts via the service's interface i_j and communicate via shared names $s_1, \ldots, s_m$.

$\mathit{Svc}_{\mathit{init}}$ and Svc_j represent an implementation of the service and describe its behaviour.

Service Broker

A service broker stores information about available service providers for potential service requesters.

$$\begin{aligned} Broker &\overset{def}{=} (a,g).\\ &\quad (p)(Add\lfloor p,a\rfloor \mid Get\lfloor p,g,a\rfloor)\\ Add &\triangleq (t,a).a(m,d).\\ &\quad (t')(Add\lfloor t',a\rfloor \mid \overline{t}\langle t',m,d\rangle)\\ Get &\triangleq (h,g,a).h(h',m,d).\\ &\quad (\overline{g}\langle m\rangle.(Get\lfloor h',g,a\rfloor \mid \overline{a}\langle m,d\rangle) \ + \ d) \end{aligned}$$

- **Publishing** a service accessible via interface x:

$$(d)(\overline{a}\langle x,d\rangle)$$

- **Requesting** the service's interface to y:

$$g(y)$$

Services in Component-Based Development

- Service as a component-based system (CBS).
- We can **modify description** of process constant:

$$Svc'_j \overset{def}{=} (i, s_{p_1}, \dots, s_{p_k}, s_{r_1}, \dots, s_{r_{(m-k)}}).Svc_j\lfloor i, s_1, \dots, s_m\rfloor$$

- Names $s_{p_1}, \dots, s_{p_{(m-k)}}$ and name i stand for **"provided" interfaces** as a selection of the service's provided shared names and its interface.
- Names $s_{r_1}, \dots, s_{r_k}$ stand for **"required" interfaces** as the rest of required shared names.
- The service can be **described as a CBS (a component)** with provided functional interfaces $i, s_{p_1}, \dots, s_{p_{(m-k)}}$ and required functional interfaces $s_{r_1}, \dots, s_{r_k}$.

Behaviour of Component-Based System

- Now, we are **ready to describe a CBS itself**, which implements a service´s behaviour and internal structure.
- The CBS is defined by its initial configuration, component hierarchy and components´ behaviour.
- **Description consists of**
  1. description of interface´s references and binding,
  2. description of control of a component´s life-cycle,
  3. description of component behaviour of primitive and composite components.

See the conference proceedings. . .

Summary and Future Work

- behaviour of services' interaction in SOA can be described in π-calculus,
- behaviour of services' implementation in CBD can be described in π-calculus,

Current and future work

- verification of properties of services and components,
- services modelling with constraints,
- model-checking in service-oriented architecture (compatibility of services, evolution of architecture, etc.).

Thank you for your attention!