Service-Orientation as a Paradigm of Computing

Wolfgang Reisig
Humboldt-Universität zu Berlin
What is the Basic Paradigm of Informatics?

2012 celebrated as the greatest computer scientist of the 20th century.

Basics of theoretical informatics:
Turing Machines (1936)
Theoretical Informatics in a nutshell

alphabet $\Sigma$; finitely many symbols a, b, c, ... ,z

words $\Sigma^*$; countably many ab, ca, aca, ...

functions $f: \Sigma^* \to \Sigma^*$; uncountably many

Some of those functions are “computable” (countably many).

Each computable function can effectively be computed
• by a computer (with unbounded store)
• by an amazingly simple kind of machine, a Turing machine.

Yet, no computer can compute more functions.
... lots of concepts

useful, undisputed:

equivalence,
abstraction/refinement
composition
complexity
logical characterizations

The computable functions pinpoint
deep theoretical results and famous open problems.

So, the theory of computable functions
is frequently considered **THE** theory of informatics.

This talk: Informatics comprises formal aspects
that can’t be explained as functions \( f: \Sigma \longrightarrow \Sigma^* \)

For example, *service oriented software architectures*
This talk:

1. Aspects that exceed classical Theoretical Informatics
2. A conceptual view on SOC
3. Towards a Theory of Services
4. Important aspect: Associative Composition
This talk:

1. Aspects that exceed classical Theoretical Informatics
2. A conceptual view on SOC
3. Towards a Theory of Services
4. Important aspect: Associative Composition
Informatics comprises *communication*

How establish reliable communication?
By sending acknowledgements, copies, etc.,
i.e. by means of *distributed algorithms* ("protocols").

*Complexity is not in computation but in communication.*
Informatics comprises *non-ending behavior*.

SOC “always on”

cloud
elevator control

business informatics “24/7”

classical view:
terminating behavior is intended,
infinite behavior is mistaken.

new view:
infinite behavior is intended.
terminating behavior is mistaken.
Informatics comprises causal independence
Informatics comprises causal independence

+ fairness assumption

motivated by “observation”
Distributed Systems and Distributed Runs
Avoid a naïve notion of “time” and of “observation”.

a deterministic system
no alternatives
one behavior (run, execution)

Causality structures the world
This talk:

1. Aspects that exceed classical Theoretical Informatics
2. A conceptual view on SOC
3. Towards a Theory of Services
4. Important aspect: Associative Composition
Paradigms of programming:

1. **Conventional (procedural) programs**
   - follows the *IPO model*: input-process-output
   - theoretical foundation/ expressive power:
     - the computable functions

2. **Object orientation**
   - attributes and methods
   - theoretical foundation:
     - abstract data types / algebraic specifications / signatures and structures (as for 1st order logic)

3. **Service orientation**
   - self contained components (reactive systems)
   - loosely coupled
   - theoretical foundation: missing
... intuitively ...

imperative programing

Object Orientation
... intuitively ...

Service Oriented Computing
The Cloud

... intuitively ...
What is a Service?

... an algorithmic component, frequently software.

**software to**
- book a journey,
- sell a ticket,
- offer cash at an ATM.

**a person’s role:**
- booking a journey,
- buying a ticket,
- withdrawing cash from an ATM.

**a technical system:**
- elevator
- self driving vehicle
- mobile phone

**an organization, providing**
- insurance
- medical surgery
Services interact goal oriented

Services interact *loosely coupled*.
In general: message passing; not handshaking.
Interacting services (instantiations) *jointly* pursue a *goal*.
They *reach* their *goal*, or *miss* it.

A frequent goal of composed services: to reach a final state together.

Often:
Two services play the role of a *provider* and a *requester*, together with a *broker*.
Instantiations

A service may spawn many instantiations.

*Input as last time*

does NOT yield output as last time

Two instantiations may

• temporally overlap,
• interact,
• behave differently.
• Different instantiation may start in different states.
A formal foundation is a base to ...

• describe semantics of implementations

• characterize expressivity of formalisms

• relate representations (equivalence, simulation)

• clarify the elementary notions of the area

• derive properties
  from structural and behavioral descriptions

• teach the area systematically
Structure of textbooks on SOA

First part:
in plain English:
“... SOA is an implementation independent concept ...”,
using many notions, poorly related.

Second part:
examples of implementations,
confusing conceptual aspects and language dependent aspects

Missing bridge:
implementation independent models.
This talk:

1. Aspects that exceed classical Theoretical Informatics
2. A conceptual view on SOC
3. Towards a Theory of Services
4. Important aspect: Associative Composition
Semantics of Services

We should do it
in analogy to programming languages:

The semantics of a service is a mathematical object!

True, this is presently not the case.

BUT WE SHOULD spend effort into this!
towards a generic modeling language

aim of, e.g.
BPMN:

\[ \bigcup \text{expressive power (L)} \]
\[ \text{all modeling languages L} \]
\[ \text{for services} \]

aim of a
\[ \text{generic Mod. language} \]

\[ \bigcap \text{expressive power (L)} \]
\[ \text{all modeling languages L} \]
\[ \text{for services} \]
Models of services

... a transition system with channels for asynchronous communication with its environment.

Semantics of $S$:
During a computation, each channel funnels a stream of data.

**Very convenient:**
Conceive the environment as a service, too!

Technically:
a relation on – infinite – streams
Interaction is represented as composition

Requirements:

The – elementary – notion of composition of services
is a (simple!) mathematical (or logical!) operation.

For services $S$ and $T$,
the composition $S \oplus T$
is a service again.

Frequently, $S \oplus T$ does not interact any more.

\[
ticketing =_{\text{def}} \text{sell\_ticket} \oplus \text{buy\_ticket}
\]
How to compose services?

Composition $S \oplus T$ has pending channels.
... is a service again.

The world consists of composed services
Requirements at composed services

Together, services may accomplish a requirement, \( \rho \).

\( S \oplus T \)

- \( S \) and \( T \) communicate boundedly
- \( S \) and \( T \) communicate responsively

... as CTL* formulas:
- \( S \oplus T \models AG \ n\text{-bounded} \)
- \( S \oplus T \models AGEF \ \text{responsive} \)

With target states:
- \( S \oplus T \) weakly terminates
- \( S \oplus T \) is deadlock free
- \( S \oplus T \) is lifelock free

- \( S \oplus T \models AGEF \ \text{terminal} \)
- \( S \oplus T \models AG \ (\text{terminal} \rightarrow \text{target}) \)
- \( S \oplus T \models AGEF \ \text{target} \)
For a requirement \( \rho \) ...

\[ S \oplus T \]

Def.: Let \( \rho \) be a requirement on services.

(i) \( S \) and \( T \) are \( \rho \)-partners

\[
\text{iff } S \oplus T \models \rho
\]

(ii) \( S \) is substitutable by \( T \)

\[
\text{iff for all } U, \quad S \oplus U \models \rho \text{ implies } T \oplus U \models \rho
\]

\( ! \text{ on-the-fly-substitution} \)

(iii) \( U \) is a \( \rho \)-adapter for \( S \) and \( T \)

\[
\text{iff } S \oplus U \oplus T \models \rho
\]

With \textit{target} states:

- \( S \oplus T \) weakly terminates
- \( S \oplus T \) is deadlock free
- \( S \oplus T \) is lifelock free

\( \text{... associative } \oplus \)
properties of services

Quests at the partners of a service, $S$, w.r.t a requirement $\rho$:

Does $S$ have $\rho$-partners at all?

Is $T$ a $\rho$-partner of $S$?

How construct a *canonical* $\rho$-partner of $S$?

How characterize *all* $\rho$-partners of $S$?

Controllability

Composability

“most liberal”

Operating Guideline
The algebraic structure of services

Given:

- a set $S$ of services,
- a composition operator $\oplus : S \times S \to S$,
- a set $Q$ of requirements $\rho_1, \ldots, \rho_n \subseteq S$.

This yields the algebraic structure

$$(S; \oplus, Q).$$

For $S, T \in S$, $\rho \in Q$,

$T$ is a $\rho$-partner of $S$,

iff $S \oplus T \vdash \rho$. $\rho (S \oplus T)$ \quad $S \oplus T \in \rho$

Let $\text{sem}_\rho(S) = \text{def} \text{ the set of all } \rho$-partners of $S$.

the “classical” requirement $\rho : \text{weak termination}$

derived notions (w.r.t some $\rho$):

$S$ may be substituted by $S' :$

$\text{sem}_\rho(S) \subseteq \text{sem}_\rho(S')$

$S$ and $T$ are equivalent:

$\text{sem}_\rho(S) = \text{sem}_\rho(T)$

$U$ adapts $S$ and $T$:

$S \oplus U \oplus T \vdash \rho$
This talk:

1. Aspects that exceed classical Theoretical Informatics
2. A conceptual view on SOC
3. Towards a Theory of Services
4. Important aspect: Associative Composition
Composing *many* services

\[
S \oplus T
\]

e.g. a supply chain
a simple idea

(S ⊕ T) ⊕ U
a simple idea

Identify two sets of channels: left port, right port.

\[(S \oplus T) \oplus U = S \oplus (T \oplus U)\]

associativity: inevitable for composition!
... and universally feasible:

in general: a channel may belong to no or to both ports
Example: vending machine

dad pays,  mom selects,
Example: vending machine

dad pays,  mom selects,  kid drinks.
A variant of the vending machine

dad pays, mom selects, kid drinks.

mom and kid must synchronize!
exclusive requester

\[ N_1 \text{ provider} \rightarrow R_1 \rightarrow \ \circ \quad \text{exclusive requester} \]

\[ N_2 \text{ requester} \]
exclusive requester

\[ N_1 \text{ provider} \rightarrow \text{requester} \rightarrow N_2 \]

*a variant:*
sharing requester

\[ N_1 \text{ provider} \rightarrow R_1 \]

\[ N_2 \text{ requester} \]

\[ \Lambda_2 \]

\[ R_2 \]

*a variant:*
sharing requester

\[ N_1 \quad \text{provider} \quad \rightarrow \quad \text{requester} \quad \rightarrow \quad N_2 \]

\[ \text{L}_2 \quad \text{R}_2 \quad \text{R}_1 \]
second sharing requester
second sharing requester
third sharing requester

N₁ provider

N₂₀ requester

N₂₁ requester

N₂₂ requester

skip the primes:
N₁ ⋅ N₂ ⋅ N₂ ⋅ N₂
generic sharing requesters

P

provider

D

D

D

M

Q requester

Q requester

Q requester

P \cdot Q \cdot Q \cdot Q

P \cdot Q \cdot Q

P \cdot Q

Q requester

Q requester

Q requester

L

D

M

generic requester Q:
prefer this variant?

generic requester Q:
prefer *this* variant?

![Diagram showing relationships between providers and requesters.]

- **P** provider
- **D**
- **Q** requester
- **M**
- **A**

**Questions:**
- What is the advantage of selecting this variant?
- How does the system handle requests and responses?

**Generic Requester Q:**
- Just make a member of **L**

**Notes:**
- P · Q · Q · Q
- P · Q · Q
- P · Q

---

49
Cyclic composition: The philosophers

This is $A \cdot B \cdot C \cdot D \cdot E$

The problem: How glue

Construct the closure $(A \cdot B \cdot C \cdot D \cdot E)^c$
Cyclic composition: The philosophers

This is \( A \cdot B \cdot C \cdot D \cdot E \)

The problem: How glue ?

Construct the closure \( (A \cdot B \cdot C \cdot D \cdot E)^c \)
... with a generic philosopher

algebraic form: $(p \cdot p \cdot p \cdot p \cdot p)^c$
The algebraic structure of services

Given:
• a set $\mathcal{S}$ of services,
• a composition operator $\oplus : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$,
• a set $\mathcal{Q}$ of requirements $\rho_1, \ldots, \rho_n \subseteq \mathcal{S}$.

This yields the algebraic structure

$$(\mathcal{S}; \oplus, \mathcal{Q}, ^c).$$

For $S, T \in \mathcal{S}, \rho \in \mathcal{Q}$

$T$ is a $\rho$-partner of $S$, iff $S \oplus T \models \rho$.

Let $\text{sem}_\rho(S) = \text{def} \text{ the set of}$

all $\rho$-partners of $S$. 
The algebraic structure of clouds

CLOUD COMPUTING
Service Orientation as a paradigm of computing

Wolfgang Reisig
Humboldt-Universität zu Berlin

1. Aspects that exceed classical Theoretical Informatics
2. A conceptual view on SOC
3. Towards a Theory of Services
4. Important aspect: Associative Composition