Defining Services -

What is a service and what is it good for?
What we do

• Analysis and Specification
  – Service-based Requirements Analysis
  – Product-line engineering
  – Model-based requirements engineering

• Deployment and Test
  – Event-triggered to time-triggered computation
  – ECU-Networks
  – Model-based Testing

• Automotive Design
  – Integrating reactive/computational behavior
  – Model-based design
  – Validation/transformation

  Tool-Support: AutoFocus
  – Validation/Verification
  – Test case generation
  – Code generation
Automotive Services

- Vehicle Speed
- Door Ajar Sensor
- Valet Switch
- Mechanical Key Identification
- Driver Door Lock
- Passenger Door Lock
- Left Rear Door Lock
- Right Rear Door Lock
- Trunk Lock
- Security Alarm
- Crash Detection
- Ignition Key
- Remote Key Control
- Exterior Light Control
- Interior Light Control
- Tuner Control
- Seat Control
- Mirror Control
Context: Analysis and Specification

Automotive software complexity
- More than 60 connected ECUs per car
- More than 100 pages spec per ECU

Mastering complexity by using functions
- Distributed functionality
- Interacting functionality

"Have we a good specification (yet)?"
- Have we conflicting functionalities?
- Have we covered all cases?
Describing Functionality: Services

- Related to interface
- Defines behaviour
- Is a partial description
- Independent of platform
- Composable
- Applicable

• System description: interface
• System description: behaviour
• Partial description
• Combination and implementation
• Completeness and consistency
• Example of application
Overview

• **System description: interface**: “How to communicate”

• **System description: behaviour**

• **Partial description**

• **Combination and implementation**

• **Completeness and consistency**
System description: Interface

- **Interface:**
  - Access point (*channel, port,*)
  - Exchangeable messages (*channel type*)
  - Directed (*input/output*)

AS : {OffHook, Dial X, OnHook}
SA : {DialTone, LineBusyTone, Ring X, …}
Overview

• System description: interface

• **System description: behaviour**: “Observing the interactions”

• Partial description

• Combination and implementation

• Completeness and consistency
System description: Observations

- **Channel history:**
  - Observation of a single channel
  - Exchanges messages of a system execution
System description: Histories

- **Timed channel history:**
  - Splitting observation into time intervals
  - One/no message per interval ("nil")
  - History: Infinite sequence of messages/nil
### System description: Executions

#### System execution:
- One channel history per channel
- Tuple of channel histories

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OffHook</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DialTone</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dial A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AudRing C</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ring C</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>CS-Kanal</th>
<th>SC-Kanal</th>
<th>BS-Kanal</th>
<th>SB-Kanal</th>
<th>AS-Kanal</th>
<th>SA-Kanal</th>
</tr>
</thead>
<tbody>
<tr>
<td>OffHook</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DialTone</td>
<td>-</td>
<td>DialTone</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Dial A</td>
<td>-</td>
<td>Dial A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>AudRing C</td>
<td>-</td>
<td>AudRing C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Ring C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>OffHook</td>
<td>-</td>
<td>-</td>
<td>Ring C</td>
</tr>
<tr>
<td>DialTone</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>DialTone</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Dial A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Dial A</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
System description: Behavior

- **System behaviour**
  - Set of system executions
  - Additional properties
    - Input completeness
    - Causality constraints

\[ (I_{AS} \times I_{BS} \times I_{CS} \times O_{SA} \times O_{SB} \times O_{SC})^{\infty} \]
System description: Relations

- Set-like system descriptions:
  - Set of (infinite) executions
    \[ B_S \subseteq (I_{AS} \times I_{BS} \times I_{CS} \times O_{SA} \times O_{SB} \times O_{SC})^\infty \]
  - Possible behaviour (execution) of a system:
    \[ b \in B_S \]

- **Relational Description**
  - Relation between input/output histories
    \[ R_S : (I_{AS} \times I_{BS} \times I_{CS})^\infty \times (O_{SA} \times O_{SB} \times O_{SC})^\infty \rightarrow \text{Bool} \]
  - Possible input/output behaviour of a system:
    \[ R_S((i_{AS},i_{BS},i_{CS}),(o_{SA},o_{SB},o_{SC})) = \text{true} \]
System description: State based

- State transition diagrams:
  - Precondition (local variables)
  - Input pattern (input ports)
  - Output pattern (output ports)
  - Post condition (local variables)

OffHook State

<table>
<thead>
<tr>
<th>AS</th>
<th>SA</th>
<th>BS</th>
<th>SB</th>
<th>CS</th>
<th>SC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BusyB = False / AS?Dial(B) /

SA!AudibleRing B; SB!Ring(A) / BusyB = True

Ring State
System description: State based

BusyB = False / AS?Dial(B) /

OffHook State

SA!AudibleRing B; SB!Ring(A) / BusyB = True

Ring State

• State-based description:
  – Relational description: single-step relation
    \[ S_S : State \times (I_{AS} \times I_{BS} \times I_{CS}) \times (O_{SA} \times O_{SB} \times O_{SC}) \times State \rightarrow Bool \]
  – Corresponding history:
    \[ R_S(i,s,o) = \forall n \in Nat. \ S_S(s_n,i_n,o_n,s_{n+1}) \]
    with \( s_n \) n-th element of sequence s
Example of application: Chisel-Diagrams

- Chisel: BellCore 1987 for Telecom-Services
System description: Overview

- System description:
  - Interface: Set of input/output ports
  - Behaviour: Set of infinite sequence of assignments
  - Description techniques: SDs, STDs

- What distinguishes a component from a service?
Overview

• System description: interface

• System description: behaviour

• Partial description: “It’s good to know what we do not know”

• Combination and implementation

• Completeness and consistency
Partial Description: Non-Determinism

- **Non-Determinism:**
  - In general caused by abstraction from the state of the system/environment

- In the development process:
  - System (component): Implementations should be deterministic
  - Service: Requirements might be non-deterministic due to missing details
Partial description: Determinism

- Useful property of a component:
  - **Deterministic (functional):**

\[
\forall i \in (I_{AS} \times I_{BS} \times I_{CS})^\infty \forall o_1, o_2 \in (O_{SA} \times O_{SB} \times O_{SC})^\infty.
(R_S(i,o_1) = true \land R_S(i,o_2) = true) \implies o_1 = o_2
\]

„For each behaviour of the environment there is at most one reaction of the system“
Partial Description: Totality

- **Totality:**
  - Implementations are always total, therefore undefined behavior leads to unexpected results in implementations

- **In the development process:**
  - System (component): Implementations should be totally defined
  - Service: Requirements might be partial due to missing scenarios
Partial description: Totality

- Useful property of a component:
  - Input completeness (total):
    \[ \forall i \in (I_{AS} \times I_{BS} \times I_{CS})^\infty \exists o \in (O_{SA} \times O_{SB} \times O_{SC})^\infty. \]
    \[ R_S(i, o) = true \]
    „For each behaviour of the environment there is (at least) one reaction of the system“
Partial description: How it is dealt with

• *Partial description*:
  – „For (at least) one behaviour of the system there no reaction of the system is defined“

• Formal interpretations:
  – „Do what you like“:
    • Under-specification = Non-determinism (totality)
    • Forget undefined behaviour

  – „No reaction is assigned“:
    • Under-specification = Partiality (no totality)
    • Remember undefined behaviour
Partial description: Component/Service

- Using it: Finding “holes” in the specification

  - **System/component:**
    - Interface: \((I_{AS}, I_{BS}, I_{CS}), (O_{SA}, O_{SB}, O_{SC})\)
    - Behaviour: \(R_{S} : (I_{AS} \times I_{BS} \times I_{CS})^\infty \times (O_{SA} \times O_{SB} \times O_{SC})^\infty \rightarrow \text{Bool} \)
      \((R_{S} \text{ total})\)

  - **Service:**
    - Interface: \((I_{AS}, I_{BS}, I_{CS}), (O_{SA}, O_{SB}, O_{SC})\)
    - Behaviour: \(R_{S} : (I_{AS} \times I_{BS} \times I_{CS})^\infty \times (O_{SA} \times O_{SB} \times O_{SC})^\infty \rightarrow \text{Bool} \)
      \((R_{S} \text{ partiell})\)

- Note: A system is also a service (but not the other way round)
Example of application: Partiality

- Partial description:
  - “OffHook” signal in state “OffHook”?
  - “OffHook” signal and “Dial A” signal at the same time?
Partial description: Overview

- Components vs. Services:
  - Component: (Input-)total relation
  - Service: (Partial) relation

- How do I combine a service (or a component)?
- How do I implement a service by a component?
Overview

• System description: interface

• System description: behaviour

• Partial description

• Combination and implementation: “Building systems”

• Completeness and consistency
Combination and implementation: Composition

- Construction of component networks:
  - Composition by connection of channels
  - Behaviour by signal-based communication (time-synchronous, message-asynchronous)
Combination and implementation: Composition

- Construction of component networks: Relational description:

\[ R_A : (I_{UA} \times I_{SA})^\infty \times (O_{AU} \times O_{AS})^\infty \mapsto \text{Bool} \]

\[ R_S : (I_{AS} \times I_{CS})^\infty \times (O_{SA} \times O_{SC})^\infty \mapsto \text{Bool} \]

\[ R_{AIS} : (I_{UA} \times I_{AS} \times I_{CS})^\infty \times (O_{AU} \times O_{SA} \times O_{SC})^\infty \mapsto \text{Bool} \]

\[ R_{AIS} ((i_{UA}, i_{AS}, i_{CS}), (o_{AU}, o_{SA}, o_{SC})) \equiv R_A ((i_{UA}, i_{AS}), (o_{AU}, o_{SA})) \land R_S ((i_{AS}, i_{CS}), (o_{SA}, o_{SC})) \]
Combination and implementation: Combination

- **Combination** of services
  - Combination by sharing interfaces
  - Behaviour by delegation to the corresponding service(s)
Combination and implementation: Combination

- **Combination** of services
  - Relational description:
    \[ R_{S_1}, R_{S_2} : (I_{AS} \times I_{BS} \times I_{CS})^\infty \times (O_{SA} \times O_{SB} \times O_{SC})^\infty \rightarrow \text{Bool} \]
    \[ R_{S_1} \upharpoonright R_{S_2} (i, o) \equiv (i \in \text{dom}(R_{S_1}) \lor i \in \text{dom}(R_{S_2})) \land \]
    \[ (i \in \text{dom}(R_{S_1}) \Rightarrow R_{S_1} (i, o)) \land (i \in \text{dom}(R_{S_2}) \Rightarrow R_{S_2} (i, o)) \]
Example of Application: Service Combination

\[ R_{S_1} \mid R_{S_2} (i,o) \equiv (i \in \text{dom}(R_{S_1}) \lor i \in \text{dom}(R_{S_2})) \land \\
(i \in \text{dom}(R_{S_1}) \Rightarrow R_{S_1}(i,o)) \land (i \in \text{dom}(R_{S_2}) \Rightarrow R_{S_2}(i,o)) \]

\[ R_{S_1} \mid R_{S_2} (s,i_1,...,i_m,o_1,...,o_n,t) \equiv (\text{dom}_{R_1}(s,i_1,...,i_m) \lor \text{dom}_{R_2}(s,i_1,...,i_m)) \land \\
((\text{dom}_{R_1}(s,i_1,...,i_m) \Rightarrow R_{S_1}(s,i_1,...,i_m,o_1,...,o_n,t)) \land \\
(\text{dom}_{R_2}(s,i_1,...,i_m) \Rightarrow R_{S_2}(s,i_1,...,i_m,o_1,...,o_n,t))) \]

\textit{where} \hspace{1em} \text{dom}_{R}(s,i_1,...,i_m) \equiv (\exists o_1,...,o_n,t.R(s,i_1,...,i_m,o_1,...,o_n,t))

- Combination of services (using \(\mu\)-calculus model checker \(\mu\cke\)):
  - Behavior agrees to one service if other is undefined
  - Behavior agrees to both services if both are defined
  - Note: behavior undefined for inconsistent services
Example of application: Service configuration

<table>
<thead>
<tr>
<th>Subscriber</th>
<th>Service</th>
<th>Instance: Originator, further parties</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>TCSC</td>
<td>B, A</td>
<td>Screen List = {B}</td>
</tr>
<tr>
<td>A</td>
<td>TCSC</td>
<td>C, A</td>
<td>Screen List = {B}</td>
</tr>
<tr>
<td>A</td>
<td>CFBL</td>
<td>B, A, C</td>
<td>Forward = C</td>
</tr>
<tr>
<td>A</td>
<td>CFBL</td>
<td>C, A, C</td>
<td>Forward = C</td>
</tr>
<tr>
<td>B</td>
<td>POTS</td>
<td>A, B</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>POTS</td>
<td>B, B</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>POTS</td>
<td>C, B</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>CFBL</td>
<td>A, C, B</td>
<td>Forward = B</td>
</tr>
<tr>
<td>C</td>
<td>CFBL</td>
<td>B, C, B</td>
<td>Forward = B</td>
</tr>
</tbody>
</table>

- Configuration:
  - Service: Partial behaviour for 2 or 3 parties
  - Combined service: Partial behaviour of all parties
- Formalisation:
  - Application of relation to state space
  - Combination of instantiated service-requirements
Combination and implementation: Refinement

- **Behavioural refinement:**
  - Used for components
  - Removes non-determinism
Combination and implementation: Refinement

- **Set-like description:**
  \[ B_{S_1}, B_{S_2} \subseteq (I_{AS} \times I_{BS} \times I_{CS} \times O_{SA} \times O_{SB} \times O_{SC})^\infty \]
  \[ B_{S_1} \leq B_{S_2} \equiv B_{S_1} \subseteq B_{S_2} \]

- **Relational description:**
  \[ R_{S_1}, R_{S_2} : (I_{AS} \times I_{BS} \times I_{CS})^\infty \times (O_{SA} \times O_{SB} \times O_{SC})^\infty \mapsto \text{Bool} \]
  \[ R_{S_1} \leq R_{S_2} \equiv \forall i \in (I_{AS} \times I_{BS} \times I_{CS})^\infty, o \in (O_{SA} \times O_{SB} \times O_{SC})^\infty. \]
  \[ R_{S_1}(i,o) \Rightarrow R_{S_2}(i,o) \]

22.10.2003  B.Schätz - What's a service (good for)?
Combination and Implementation: Implementation

- Implementing:
  - Components/Services: Component can supply services
  - Services: Service becomes more deterministic and defined
Combination and implementation: Implementation

- **Implementation:**
  - Refinement plus "Improved input behaviour"

- Relational description
  \[
  R_{S_1}, R_{S_2} : (I_{AS} \times I_{BS} \times I_{CS})^\infty \times (O_{SA} \times O_{SB} \times O_{SC})^\infty \rightarrow \text{Bool}
  \]
  \[
  R_{S_1} \prec R_{S_2} \equiv R_{S_1} \leq R_{S_2} \land \text{dom}(R_{S_2}) \subseteq \text{dom}(R_{S_1})
  \]
  where \( \text{dom}(R_S) \equiv \{i \mid \exists o \in (O_{SA} \times O_{SB} \times O_{SC})^\infty . R_S(i, o) = \text{true} \} \)

- Note: Implementation of components = Refinement

22.10.2003
B. Schätz - What's a service (good for)?
Combination and implementation: Overview

- Applying services:
  - Implementation: Component offers service
  - Combination: Services form a more defined service

- Which services are (reasonably) combinable?
- What makes a service a component?
Overview

- System description: interface
- System description: behaviour
- Partial description
- Combination and implementation
- Completeness and consistency: “Done enough? Done too much?”
Completeness and consistency: Completeness

- Implementing several services by a component:
  - Completeness: Each behaviour is covered
  - Informal: Combination of services is not underspecified
Completeness and consistency: Completeness

Relational description:

\[ R_S : (I_{AS} \times I_{BS} \times I_{CS})^\infty \times (O_{SA} \times O_{SB} \times O_{SC})^\infty \rightarrow \text{Bool} \]

\[ \text{dom}(R_S) \equiv (I_{AS} \times I_{BS} \times I_{CS})^\infty \]

Methodical: Combination of services describes a component

\[ R_S : (I_{AS} \times I_{BS} \times I_{CS})^\infty \times (O_{SA} \times O_{SB} \times O_{SC})^\infty \rightarrow \text{Bool} \]
Example of application: Completeness Analysis

\[ \text{dom}(R_S) \equiv (I_{AS} \times I_{BS} \times I_{CS})^\infty \]

\[ \forall s. \text{Reach}(s) \Rightarrow \forall i_1, \ldots, i_m. \text{dom}_R(s, i_1, \ldots, i_m) \]

\[ \mu \text{Reach}(s) \equiv \text{Init}(s) \lor \exists i_1, \ldots, i_m, o_1, \ldots, o_n, t. \]
\[ \text{Reach}(t) \land R(t, i_1, \ldots, i_m, o_1, \ldots, o_n, s) \]

- Partiality detection:
  - Reachable states
  - Partiality: Input without a defined transition
Example of application : POTS/POTS

- μcke-model checking
  - Configuration:
    - POTS(A,B)
    - POTS(B,A)
  - Undefinedness:
    - B calls A when A goes offHook
    - Precedence caller vs.callee
Completeness and consistency: Consistency

- Implementing several services by a component:
  - Consistency: Services fit together
  - Informal: No contradiction of service requirements
Completeness and consistency : Consistency

- Relation description:

\[ R_{S_1}, R_{S_2} \text{ consistent } \equiv \forall i \in (I_{AS} \times I_{BS} \times I_{CS})^\infty . (i \in \text{dom}(R_{S_1}) \cap \text{dom}(R_{S_2})) \Rightarrow \exists o \in (O_{SA} \times O_{SB} \times O_{SC})^\infty . R_{S_1}(i,o) \land R_{S_2}(i,o) \]

- Methodical: Combination of services is implementation of each combined service

\[ R_{S_1}, R_{S_2} \text{ consistent } \Rightarrow R_{S_1} \mid R_{S_2}(i,o) \land R_{S_1}(i,o) \land R_{S_1}(i,o) \land R_{S_2}(i,o) \land R_{S_1}(i,o) \land R_{S_2}(i,o) \]
Example of application: Conflict detection

\[ \forall i \in (I_{AS} \times I_{BS} \times I_{CS})^\infty. (i \in \text{dom}(R_{S_1}) \cap \text{dom}(R_{S_2})) \Rightarrow \]

\[ \exists o \in (O_{SA} \times O_{SB} \times O_{SC})^\infty. R_{S_1}(i,o) \land R_{S_2}(i,o) \]

\[ \forall s. \text{Reach}_{R_1 \mid R_2}(s) \Rightarrow \forall i_1, \ldots, i_m. \]

\[ (\text{dom}_{R_1}(s, i_1, \ldots, i_m) \land \text{dom}_{R_2}(s, i_1, \ldots, i_m)) \Rightarrow \]

\[ \exists o_1, \ldots, o_n, t. \ R_1(s, i_1, \ldots, i_m, o_1, \ldots, o_n, t) \land R_2(s, i_1, \ldots, i_m, o_1, \ldots, o_n, t) \]

where \[ \text{dom}_R(s, i_1, \ldots, i_m) \equiv (\exists o_1, \ldots, o_n, t. R(s, i_1, \ldots, i_m, o_1, \ldots, o_n, t)) \]

- Conflict detection:
  - Reachable states
  - Conflict: Input without the defined transitions
    - Conflict by conflicting output
    - Conflict by conflicting successor states
Example of application: TCSC/CFBL

<table>
<thead>
<tr>
<th>Sub-</th>
<th>Service</th>
<th>Instance: Originator, further par-</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>TCSC</td>
<td>B, A</td>
<td>Screen List = {B}</td>
</tr>
<tr>
<td>A</td>
<td>TCSC</td>
<td>C, A</td>
<td>Screen List = {B}</td>
</tr>
<tr>
<td>A</td>
<td>CFBL</td>
<td>B, A, C</td>
<td>Forward = C</td>
</tr>
<tr>
<td>A</td>
<td>CFBL</td>
<td>C, A</td>
<td>Screen List = {B}</td>
</tr>
<tr>
<td>B</td>
<td>POT5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>POT5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>CFBL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>CFBL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- µcke-model checking
  - Configuration:
    - TCSC(?,A,{B}): B is not accepted by A
    - CFBL(?,A,C,{C}): A forwarded to C if “Busy”
  - Conflict:
    - TCSC(B,A,{B}): Screen-signal to B
    - CFBL(B,A,C,{C}): Busy tone to B

```
A    B    C    Switch

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

---

22.10.2003  B.Schätz - What's a service (good for)?
Completeness and consistency: Overview

- In the development process:
  - Completeness: Combination of services describes component
  - Consistency: Combination of services is implementation
Summary

• Context:
  – Developing functionalities for embedded systems
  – Describing services as scenarios

• Approach:
  – Applying a state-based notion of service
  – Applying combination, consistency, completeness

• Support:
  – Automatic detection of inconsistency, incompleteness
  – Light-weight/fast technique
  – Mechanism invisible to user