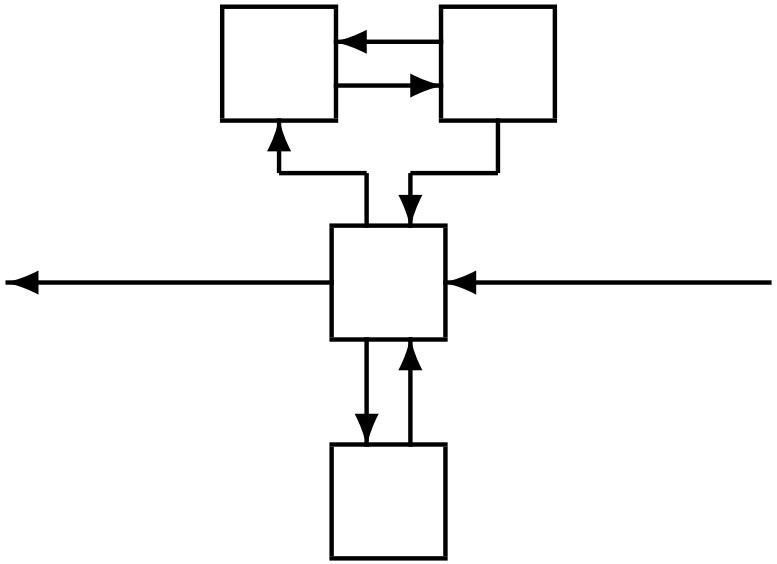
Implementing Services by Partial State Machines

Prof. Dr. Walter Dosch and Dr. Annette Stümpel

Institute of Software Technology and Programming Languages
University of Lübeck

SOFSEM 2009

Networks, Components, Services



- Complex system = network of communicating components
- Components provide contracted services through public interfaces.
- Systems evolve concurrently coordinated by service requests and responses.
- A service relates a stream of requests to a stream of responses.
- The service is only provided for input streams from the service domain.

Specification and Design of Service Components

Specification

communication-based view

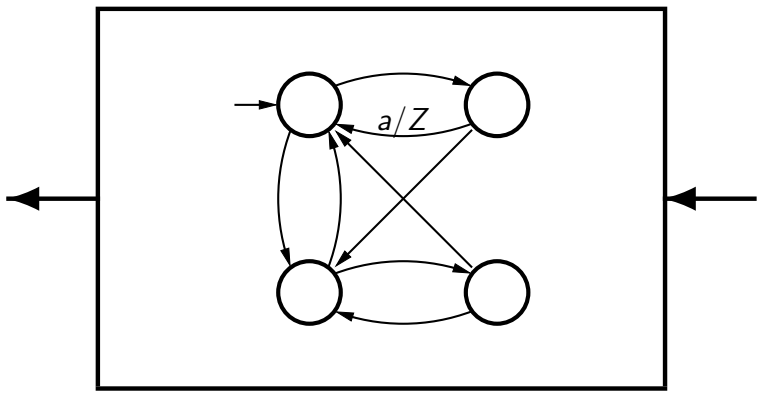
"black box"

partial stream function

Design

state-based view

"glass box"



partial state machine

Specification and Design of Service Components

Specification

communication-based view

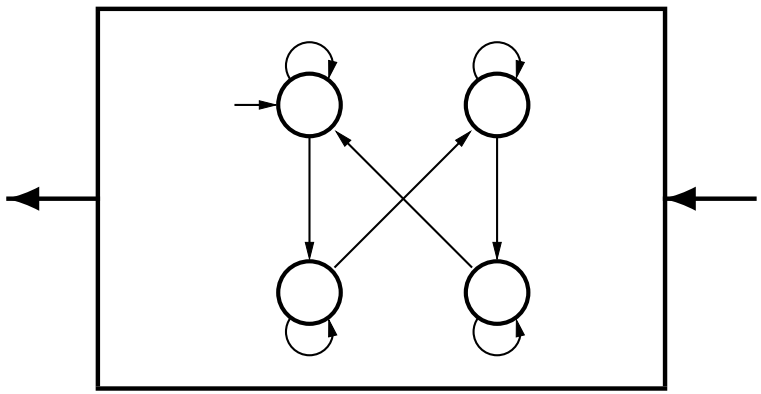
"black box"

partial stream function

Design

state-based view

"glass box"



partial state machine

Specification and Design of Service Components

Specification

communication-based view

"black box"

partial stream function

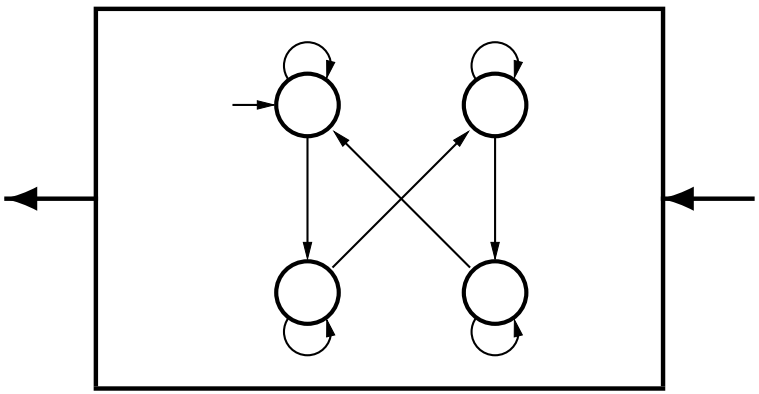
design

decision

Design

state-based view

"glass box"



partial state machine

Streams

Streams model communication histories on undirectional FIFO channels.

finite streams: $\mathcal{A}^\star = \{\langle x_1, \ldots, x_m\rangle \mid x_i \in \mathcal{A}, m \geq 0\}$
concatenation: $\langle x_1, \ldots, x_m\rangle \,\&\, \langle y_1, \ldots, y_n\rangle = \langle x_1, \ldots, x_m, y_1, \ldots, y_n\rangle$

A bounded stack service is used as a running example:

Input streams of a bounded stack service

input messages: $In = push(Data) \cup \{pop\}$
an input stream: $\langle push(1), push(2), push(3), pop, pop, push(4), pop\rangle$

Service Domains

A service usually does not work in an arbitrary environment.
It provides its functionality only on a subset of all input streams.

Definition (Service Domain)

A service domain $ServDom \subseteq In^\star$ over a set In of messages validates:

non-empty: $\langle\rangle \in ServDom$

prefix closed: $X \,\&\, Y \in ServDom \implies X \in ServDom$

Service domain of a bounded stack service

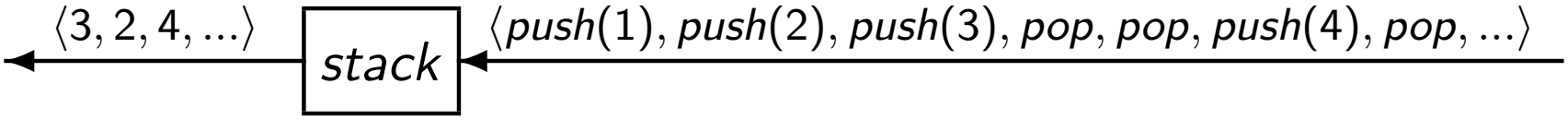
The service domain $ServDom \subseteq In^\star$ of a bounded stack service with capacity $n > 0$ is constructed by $(Psh_k \in (push(Data))^{\leq k})$:

$$Psh_n \in ServDom$$

$$Psh_{n-1} \,\&\, X \in ServDom \implies Psh_{n-1} \,\&\, \langle push(d), pop\rangle \,\&\, X \in ServDom$$

unexpected: $\langle pop\rangle \,\&\, X, \quad \langle push(d_1), \ldots, push(d_{n+1})\rangle \,\&\, X$

Services



Definition (Service)

For a service domain *ServDom* over the input messages *In*, a service $serv : ServDom \to Out^\star$ validates:

strict: $serv(\langle\rangle) = \langle\rangle$

monotonic: $X \& Y \in ServDom \implies serv(X)$ 'is prefix of' $serv(X \& Y)$

A service provides output only in response to input. $serv(X)$

Additional input can only prolong previous output: $serv(X \,\&\, Y)$

Bounded stack service *stack*: $ServDom \to Data^\star$ $(Psh_k \in (push(Data))^{\leq k})$

no pop: $stack(Psh_n) = \langle\rangle$

push/pop: $stack(Psh_{n-1} \& \langle push(d), pop\rangle \& X) = \langle d\rangle \& stack(Psh_{n-1} \& X)$

Partial State Machines

Partial state machines implement services.
The readiness set determines the acceptable inputs for each state.

Definition (Partial State Machine)

$$\mathcal{M} = (State, In, Out, Ready, next, out, init)$$

$State$	set of **states**
In	set of **input messages**
Out	set of **output messages**
$Ready \subseteq State \times In$	**readiness set**
$next : Ready \to State$	(single-step) **next state function**
$out : Ready \to Out^\star$	(single-step) **output function**
$init \in State$	**initial state**

Proposition: The processing domain containing all acceptable input streams is a service domain.
Proposition: The multi-step output function is a service.

Partial State Machines

(Minimal) partial state machine for the bounded stack service

$$\mathcal{M} = (Data^{\leq n}, In, Data, Ready, next, out, \langle\rangle)$$

Transition table for the state transitions *next*, output function *out*: $(D_k \in Data^{\leq k})$

$State$	In	$State'$	$Out^\star$
D_{n-1}	$push(d)$	D_{n-1} & $\langle d\rangle$	$\langle\rangle$
D_{n-1} & $\langle d\rangle$	pop	D_{n-1}	$\langle d\rangle$

The first two columns reflect the readiness set *Ready*

Aim

Specification

communication-based view

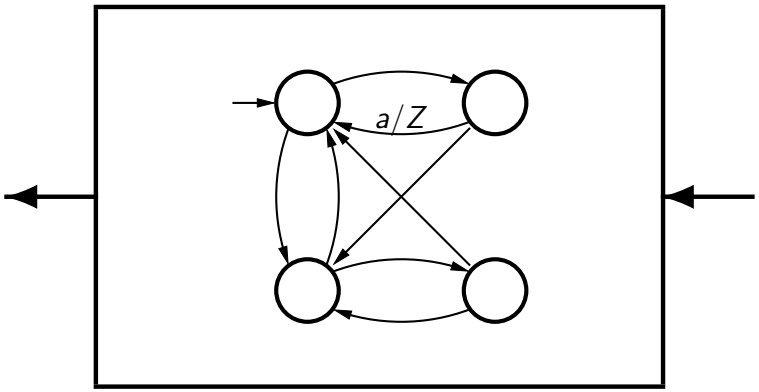
partial stream function

design

decision

Design

state-based view



partial state machine

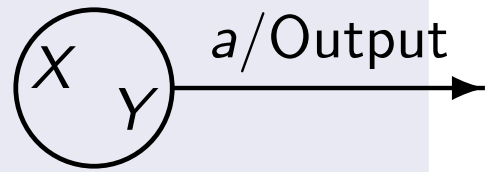
History Abstractions

History abstractions relate input streams with states.

Definition (History Abstraction)

A history abstraction $abstr : ServDom \to State$ for a service $serv$ is

- service compatible: $abstr(X) = abstr(Y) \implies (X \,\&\, \langle a \rangle \in ServDom \iff Y \,\&\, \langle a \rangle \in ServDom)$
- transition closed: $abstr(X) = abstr(Y), X \,\&\, \langle a \rangle \in ServDom \implies abstr(X \,\&\, \langle a \rangle) = abstr(Y \,\&\, \langle a \rangle)$



- output compatible: $abstr(X) = abstr(Y), X \,\&\, \langle a \rangle \in ServDom \implies serv(X \& \langle a \rangle)$ 'without' $serv(X) = serv(Y \& \langle a \rangle)$ 'without' $serv(Y)$

The requirements ensure that the identification of input histories as states is compatible with the service domain and with transitions.

History Abstractions

A history abstraction for a bounded stack

$$abstr : ServDom \longrightarrow Data^{\leq n} \qquad (Psh_k \in (push(Data))^{\leq k},\ m \leq n)$$

$$\text{no pop:} \quad abstr(\langle push(d_1), \ldots, push(d_m)\rangle) = \langle d_1, \ldots, d_m\rangle$$

$$\text{push/pop:} \quad abstr(Psh_{n-1} \,\&\, \langle push(d), pop\rangle \,\&\, X) = abstr(Psh_{n-1} \,\&\, X)$$

The history abstraction successively removes pairs of corresponding push and pop requests.

Construction of a Partial State Machine

The history abstraction is the design decision for a transformation:

Theorem

$$\boxed{\mathit{service\ serv} : \mathit{ServDom} \to \mathit{Out}^\star}$$

$$\mathit{history\ abstraction} \Big\Downarrow \mathit{abstr} : \mathit{ServDom} \to \mathit{State}$$

$$\boxed{\mathcal{M}[\mathit{serv}, \mathit{abstr}] \quad = \quad (\mathit{State}, \mathit{In}, \mathit{Out}, \mathit{Ready}, \mathit{next}, \mathit{out}, \mathit{init})}$$

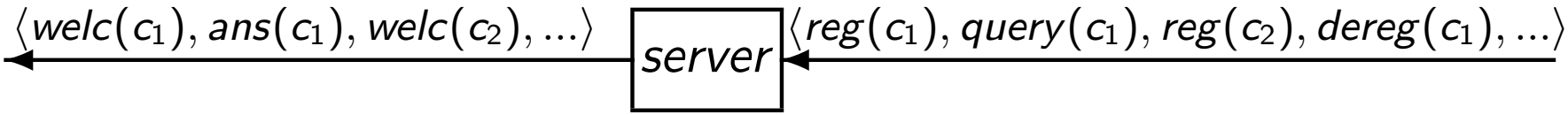
$$\begin{array}{rcl}
\mathit{init} & = & \mathit{abstr}(\langle\rangle) \\
(\mathit{abstr}(X), a) \in \mathit{Ready} & \Longleftrightarrow & X \,\&\, \langle a \rangle \in \mathit{ServDom} \\
\mathit{next}(\mathit{abstr}(X), a) & = & \mathit{abstr}(X \,\&\, \langle a \rangle) \\
\mathit{out}(\mathit{abstr}(X), a) & = & \mathit{serv}(X \,\&\, \langle a \rangle) \ \text{'without'}\ \mathit{serv}(X)
\end{array}$$

The machine's processing domain coincides with the service domain.

The machine's multi-step output function coincides with the service.

Application: Server with Registration

A server offers its services to a collection of clients $Client \neq \emptyset$:
It processes requests (register, query, deregister) of clients.



Interface

input messages $In = reg(Client) \cup query(Client) \cup dereg(Client)$
output messages $Out = welc(Client) \cup ans(Client)$

Interaction process of a single client $c \in Client$

$$Process(c) = \langle reg(c) \rangle \,\&\, query(c)^\star \,\&\, \langle dereg(c) \rangle$$

The subsequence of requests from the same client must be an initial sequence of the client's interaction process.

Service domain: $X \in ServDom$ iff for all clients $c \in Client$

$filter(X)(\{reg(c), query(c), dereg(c)\})$ 'is prefix of' $Process(c)^\star$

Application: Server with Registration

The behaviour of the server is specified with an auxiliary function on arbitrary input streams:

Generalization $serverGnrl : In^\star \to Out^\star$

$$
\begin{array}{lll}
\text{no input:} & serverGnrl(\langle\rangle) & = \langle\rangle \\
\text{register:} & serverGnrl(\langle reg(c)\rangle \,\&\, X) & = \langle welc(c)\rangle \,\&\, serverGnrl(X) \\
\text{query:} & serverGnrl(\langle query(c)\rangle \,\&\, X) & = \langle ans(c)\rangle \,\&\, serverGnrl(X) \\
\text{deregister:} & serverGnrl(\langle dereg(c)\rangle \,\&\, X) & = serverGnrl(X)
\end{array}
$$

The server $server : ServDom \to Out^\star$ is the specialization

$$server(X) = serverGnrl(X).$$

History abstraction

The history abstraction records the set of registered clients:

$$active : ServDom \to \mathcal{P}_{\text{fin}}(Client)$$

Application: Server with Registration

Constructed state machine

$$\mathcal{M} = (\mathcal{P}_{\text{fin}}(\mathit{Client}), \mathit{In}, \mathit{Out}, \mathit{Ready}, \mathit{next}, \mathit{out}, \emptyset)$$

Transition table for the state transitions *next*, output function *out*:

Precondition	*State*	*In*	*State'*	*Out**
$c \notin C$	C	$\mathit{reg}(c)$	$C \cup \{c\}$	$\langle \mathit{welc}(c) \rangle$
$c \in C$	C	$\mathit{query}(c)$	C	$\langle \mathit{ans}(c) \rangle$
$c \in C$	C	$\mathit{dereg}(c)$	$C \setminus \{c\}$	$\langle\rangle$

The predicates in the precondition document the readiness set of the server.

Conclusion

Interface: The input messages and output messages have types.
Service domain: Services accept only a subset of all possible input histories.

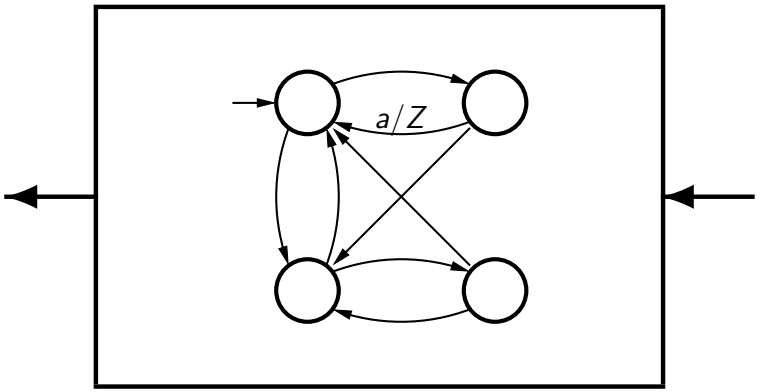
Specification

communication-based view

service

Design

state-based view



partial state machine

history

abstraction

Engineering Theory of Services . . .