

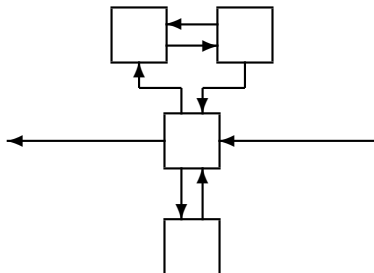
Implementing Services by Partial State Machines

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”

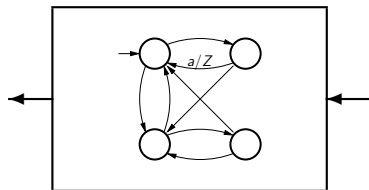


partial stream function

Design

state-based view

“glass box”



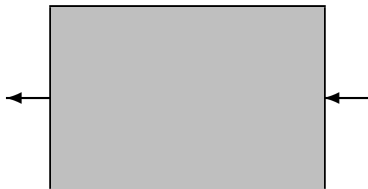
partial state machine

Specification and Design of Service Components

Specification

communication-based view

“black box”

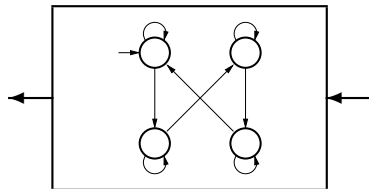


partial stream function

Design

state-based view

“glass box”



partial state machine

Specification and Design of Service Components

Specification

communication-based view

“black box”



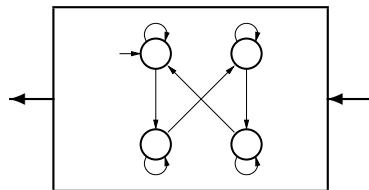
partial stream function

design
→
decision

Design

state-based view

“glass box”



partial state machine

Streams

Streams model **communication histories** on unidirectional FIFO channels.

finite streams: $\mathcal{A}^* = \{\langle x_1, \dots, x_m \rangle \mid x_i \in \mathcal{A}, m \geq 0\}$

concatenation: $\langle x_1, \dots, x_m \rangle \& \langle y_1, \dots, y_n \rangle = \langle x_1, \dots, x_m, y_1, \dots, y_n \rangle$

A bounded stack service is used as a running example:

Input streams of a bounded stack service

input messages: $In = \text{push}(Data) \cup \{\text{pop}\}$

an input stream: $\langle \text{push}(1), \text{push}(2), \text{push}(3), \text{pop}, \text{pop}, \text{push}(4), \text{pop} \rangle$

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** $ServDom \subseteq In^*$ over a set In of messages validates:

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

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

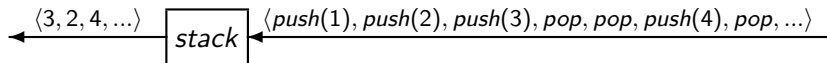
Service domain of a bounded stack service

The service domain $ServDom \subseteq In^*$ 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: $\langle pop \rangle \& X, \langle push(d_1), \dots, push(d_{n+1}) \rangle \& X$

Services



Definition (Service)

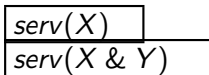
For a service domain $ServDom$ over the input messages In , a **service** $serv : ServDom \rightarrow Out^*$ validates:

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.

Additional input can only prolong previous output:



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

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)

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

<i>State</i>	set of states
<i>In</i>	set of input messages
<i>Out</i>	set of output messages
$\text{Ready} \subseteq \text{State} \times \text{In}$	readiness set
$\text{next} : \text{Ready} \rightarrow \text{State}$	(single-step) next state function
$\text{out} : \text{Ready} \rightarrow \text{Out}^*$	(single-step) output function
$\text{init} \in \text{State}$	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 :
 $(D_k \in Data^{\leq k})$

<i>State</i>	<i>In</i>	<i>State'</i>	<i>Out*</i>
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

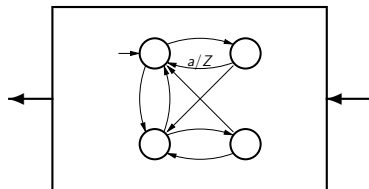


partial stream function

design
→
decision

Design

state-based view



partial state machine

History Abstractions

History abstractions relate input streams with states.

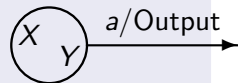
Definition (History Abstraction)

A **history abstraction** $abstr : ServDom \rightarrow State$ 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:

$$abstr(X) = abstr(Y), X \& \langle a \rangle \in ServDom \implies serv(X \& \langle a \rangle) \text{ 'without' } serv(X) = serv(Y \& \langle a \rangle) \text{ 'without' } serv(Y)$$

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

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

no pop: $abstr(\langle push(d_1), \dots, push(d_m) \rangle) = \langle d_1, \dots, d_m \rangle$

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

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

$$\boxed{\text{service } \text{serv} : \text{ServDom} \rightarrow \text{Out}^*}$$

$$\text{history abstraction} \Downarrow \text{abstr} : \text{ServDom} \rightarrow \text{State}$$

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

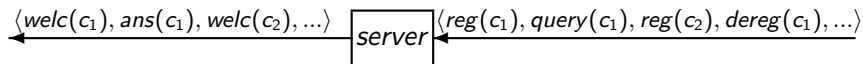
$$\begin{aligned} \text{init} &= \text{abstr}(\langle \rangle) \\ (\text{abstr}(X), a) \in \text{Ready} &\iff X \ \& \ \langle a \rangle \in \text{ServDom} \\ \text{next}(\text{abstr}(X), a) &= \text{abstr}(X \ \& \ \langle a \rangle) \\ \text{out}(\text{abstr}(X), a) &= \text{serv}(X \ \& \ \langle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

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 $c \in Client$

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

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' $Process(c)^*$

Application: Server with Registration

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

Generalization $serverGnrl : In^* \rightarrow Out^*$

no input:	$serverGnrl(\langle \rangle)$	=	$\langle \rangle$
register:	$serverGnrl(\langle reg(c) \rangle \& X)$	=	$\langle welc(c) \rangle \& serverGnrl(X)$
query:	$serverGnrl(\langle query(c) \rangle \& X)$	=	$\langle ans(c) \rangle \& serverGnrl(X)$
deregister:	$serverGnrl(\langle dereg(c) \rangle \& X)$	=	$serverGnrl(X)$

The server $server : ServDom \rightarrow Out^*$ is the specialization

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

History abstraction

The history abstraction records the set of registered clients:

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

Application: Server with Registration

Constructed state machine

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

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

<i>Precondition</i>	<i>State</i>	<i>In</i>	<i>State'</i>	<i>Out</i> [*]
$c \notin C$	C	$\text{reg}(c)$	$C \cup \{c\}$	$\langle \text{welc}(c) \rangle$
$c \in C$	C	$\text{query}(c)$	C	$\langle \text{ans}(c) \rangle$
$c \in C$	C	$\text{dereg}(c)$	$C \setminus \{c\}$	$\langle \rangle$

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

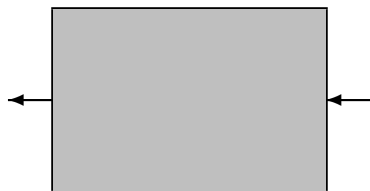
Conclusion

Interface: The input messages and output messages have types.

Service domain: Services accept only a subset of all possible input histories.

Specification

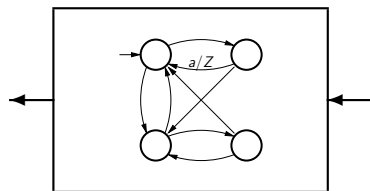
communication-based view



service

Design

state-based view



partial state machine

history
→
abstraction

Engineering Theory of Services ...