

A decorative graphic consisting of a vertical line and a horizontal line intersecting at a point to the left of the text.

EECS 122: Introduction to Communication Networks

Unit 2: Fundamentals

Some fundamentals... before moving further

- Non-Zero delay between **sending** and **receiving** of a given message... (propagation delay!)
 - The speed of light – the highest possible is $c = 300\,000\text{ km/s}$ in vacuum!!
 - The speed of electricity and light in optical fiber is
$$v = \frac{2}{3} c$$
- A bounded amount of messages can be posted in a time unit...
 - Think in term of the speed of Morse'ing or typing... or...
 - *Think in terms of the speed of your internet access... ☺*
- Loss/errors in data transmission

➔ THIS ALL IS REALITY !!!

A Distributed System...

- A ***distributed computing system*** consists of multiple ***autonomous*** processors that do not share primary memory but ***cooperate by sending messages*** over a communication network.

Henri Bal/ Colouris

Comments:

- “ Each individual entity is . typically . a full-fledged, system operating on its own
- “ Individual entities might follow local policies, be subject to local constraints
- “ Each of them might fail (completely or operate wrongly) at any time .

Implication of the Real Message Passing

- No participant has complete information about the system state.
 - ➔ I do not know what you know – NEVER
- Participants make decisions based only on local information.
- There is no implicit assumption that a **precise** common understanding of time exists.
 - Can be elaborated using proper mechanisms...with some accuracy...

The Two Army Problem:



- Principles

- A and B have $2N$ soldiers each. The black army has $3N$ soldiers. In case of conflict the bigger force wins. The two orange forces have to communicate to synchronize their attack. Can they?
- For this communication they need:
 - A common language (possibly not understood by the black army)
 - A communication path - a messenger - who has to pass through the land occupied by the enemy (and could be intercepted with a given probability p). A successful trip takes time D .

Some difficulties....

- Assume that Commander A and Commander B send a messenger to the counterpart with different suggested time for an attack. Afterwards both agree for the suggestion of the partner.
The play goes on....
- Let the Commander A (senior) send a messenger with an order.
 - Did the Commander B receive the message? When? How does Commander A know?
 - Let request the commander B to acknowledge his readiness to follow the orders. Is the victory sure ?
How does B know that A got the acknowledgment?
- The attack of the Orange armies will **always** fail with probability p ...

Is reliable data delivery possible?

- Sender:
Send the information. If no acknowledgment has been received within time "b" → send again.
- Receiver:
Wait for an information. Acknowledge the received information.

This is called a "send and wait" protocol !

Note: *One possible mechanism to reduce losses*

Let us do some maths...

- Successful transmission is acknowledged after a constant time b .
- Assume that with probability Π there is NO acknowledgement.
- The time needed for a successful transmission is
 - b with probability $(1 - \Pi)$
 - $2b$ with probability $(1 - \Pi) \Pi$
 - $3b$ with probability $(1 - \Pi) \Pi^2$
 - $(k+1)b$ with probability $(1 - \Pi) \Pi^k$
- **Note:** In a **finite** time it is NOT possible to achieve the delivery with probability 1.0

A Bank Transfer...

- Consider a bank with headquarters/accounting in METROPOLY which has a branch in VILLAGE
- Client: Has 2500 \$ on his account. He enters the branch in VILLAGE to cash 1000\$
- Send-and-Wait will be used to transmit this request.
 - **Clerk 1** (branch) Posts and repeats upon timer expiration!
 "If Balance > 1000 \$
 then (subtract 1000 \$ and acknowledge),
 otherwise call the police in VILLAGE.
 - **Clerk 2** (headquarters): Executes the request.
- Imagine the acknowledgement is lost, several times in sequence!

The client – server model... (distributed comput.)

- The clerk 2 is „only“ executing the requests posted by the clerk 1
- Conceptually, clerk one might have done it himself
 - But did not have the data „handy“
 - And –possibly- could serve next customer while clerk 2 has been processing his request.. (pipelining!)
- Such schema of operation is called „**client- server**“ model
 - **Service:** Any act or performance that one party can offer to another that is essentially intangible and does not result in the ownership of anything. Its production may or may not be tied to a physical product.

D. Jobber, Principles and Practice of Marketing
 - Focus is on the *output*, the *result* of the service NOT the means to achieve it

D. Jobber, Principles and Practice of Marketing

But –we have had the problem...

Remedy?

- One way to go: Replace the Send-and- wait by some better solution (without duplications!)
 - Will show you later in the class how to achieve it!
- Another way to go: Change the way of interaction between the clerks!
 - The next slides

Restructuring the Interaction (1st Option)

- Check the Balance
 - **Clerk 1** Posts a following message and sets a timer!
Post me the balance , Account No. xxxxx
Comment: can be repeated several times
 - **Clerk 2** Executes the request
- Set New Balance
 - **Clerk 1** Computes the new balance;
 - **Clerk 1** Posts and repeats upon timer expiration!
New Balance for Account No xxxx is 4000 \$, confirm
 - **Clerk 2** : Executes the request.
- **Idempotent actions:** Activities which result does not change in case of repetitive execution.

Restructuring the Interaction (2nd Option)

- Add to each request of the Clerk 1 a ***transaction identifier***
 - Use the same transaction identifier for an original request and for any possible repetition of this request
 - Do not use the same identifier for different transactions!
- Good news. It seems to work
- What happens if messages might be heavily delayed
 - The Clerk 2 should memorize the list of all the “already used” transaction numbers for each account number
 - He has something to memorize....

The Concept of **State**: intuitively


- Information relevant to the progress of some activity. Access to this information is frequently necessary to assure proper continuation of this activity
- Send-and- wait
 - Sender has to be aware of the timer!!!
 - Receiver is **stateless**
- In the second solution – the being stateless is a desired feature of a server!
 - Consider changing the server ? Server Collapse?
 - Consider **Scalability** if there are many clients and many accounts - state has been memorized for each of them?

- A system is scalable if it will remain (efficiently) operable in case of a significant increase in the number of **resources** and **users**:
 - Controlling the cost of resources
 - Controlling the performance loss
 - Preventing software resources running out (e.g. addresses)

Scalability has to be assured BY DESIGN!!

Hard State vs. Soft State

- Introducing state might be necessary. Typically if some resources have to be kept available...
 - Consider the restructured banking example. If several clients would use the same account a „lock“ on access would be needed between Checking Balance and setting New Balance.
- Hard State Approach:
 - State information has to be deleted as result of a proper action!
 - What happens with the “Lock” if the computer of CLERK 1 brakes down after “checking the balance”?
- Soft State Approach:
 - The state information removed if not reactivated since XXXX
 - *Like putting shoes in the store on hold for 2 days...*

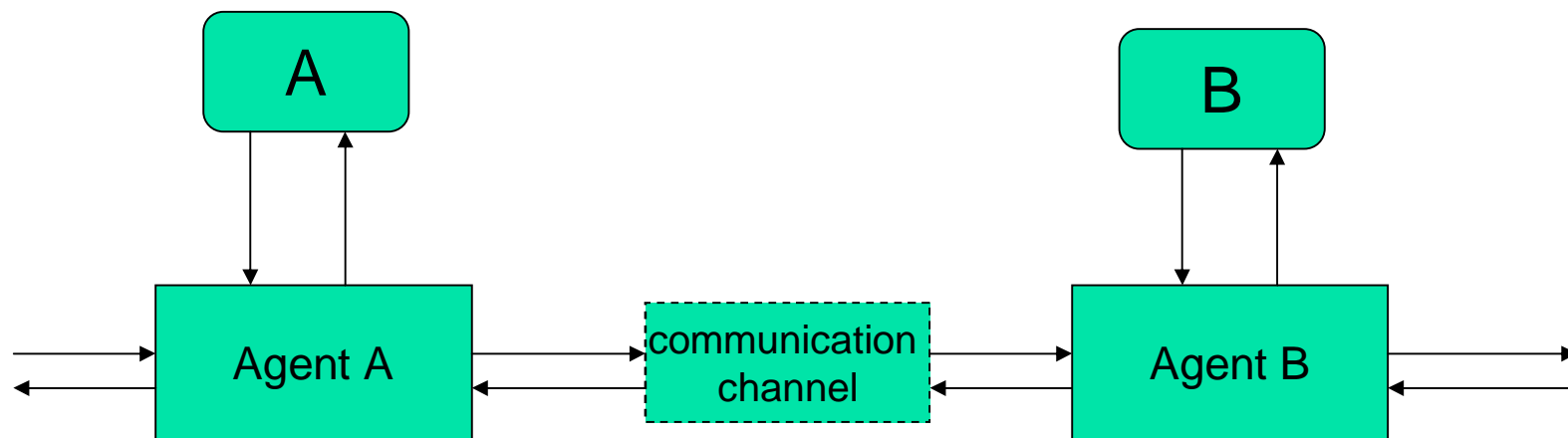


Communication as a Service

Service vs. Protocol

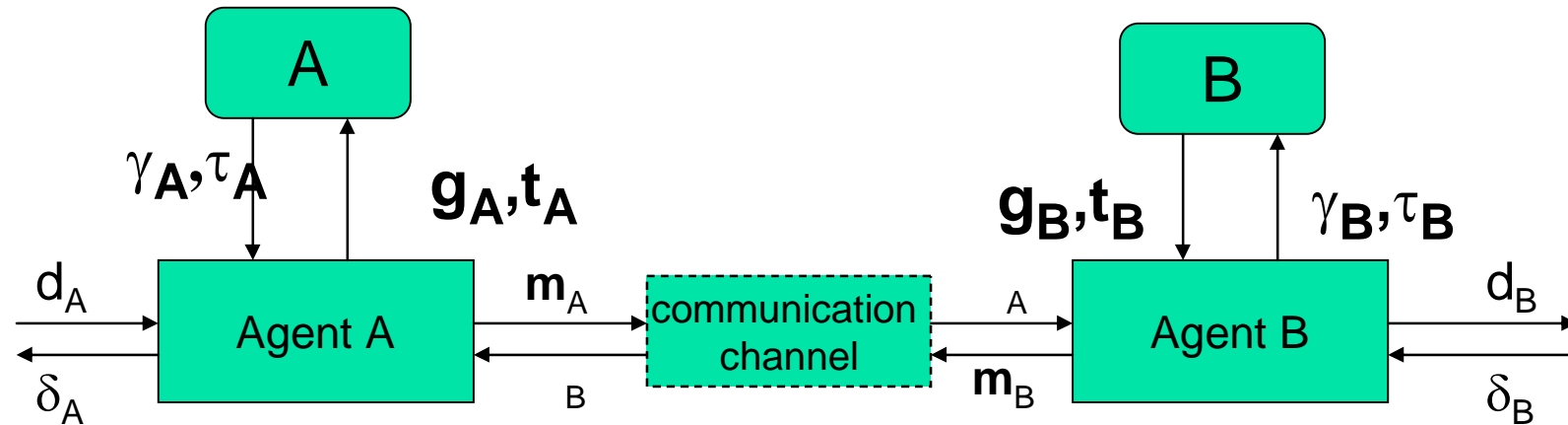
Communication as a service

- Entities A, B use a **communication service** provided by a pair of INTERLOCUTORS



- Note the
 - Interactions between an agent and the service user
 - Agent's interactions with the local environment
 - Data exchanged by the agent with the remote partner

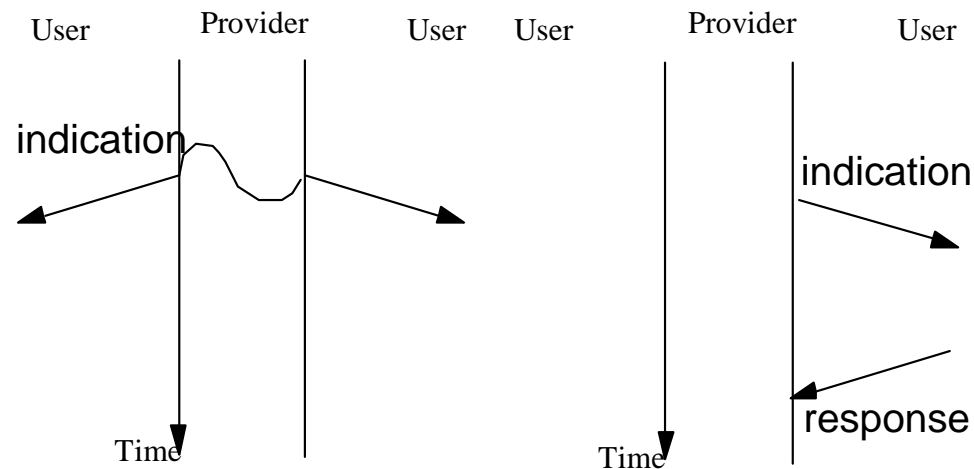
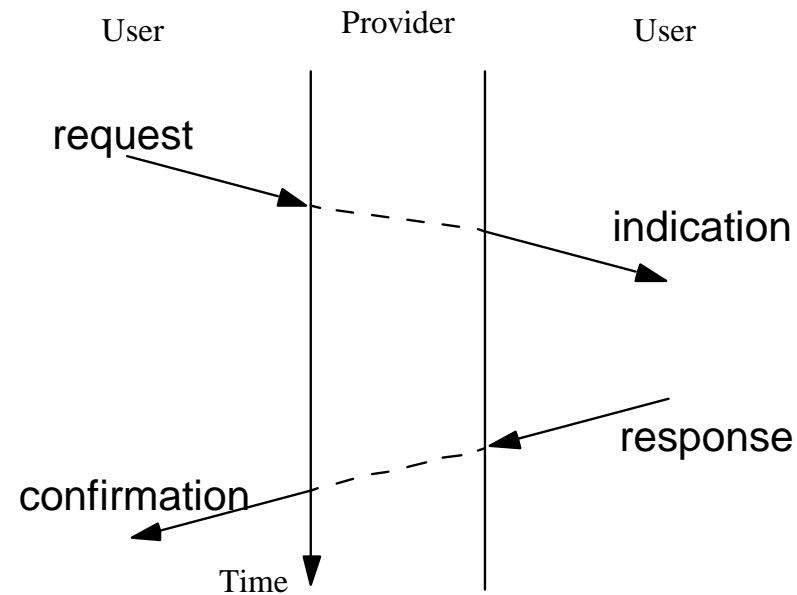
The Service Interface



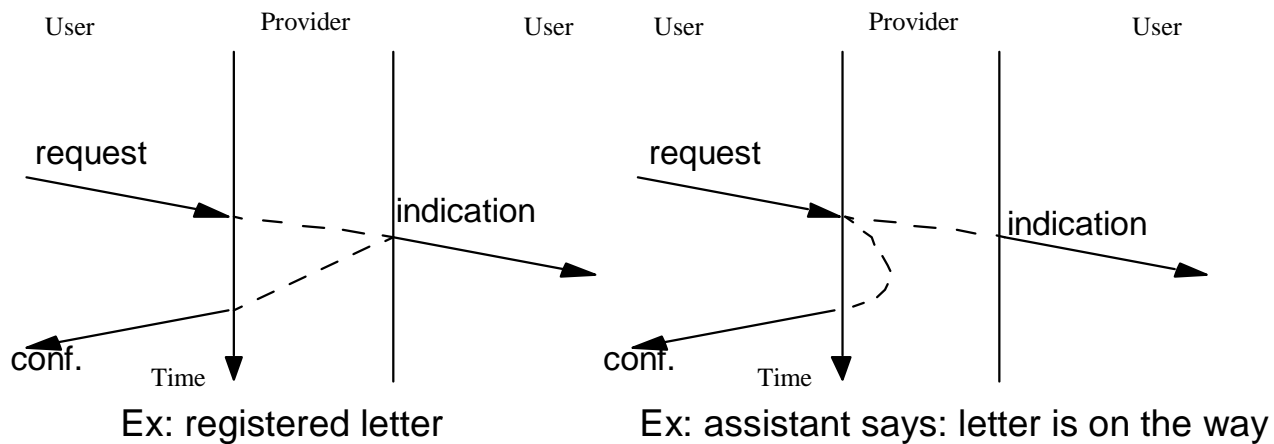
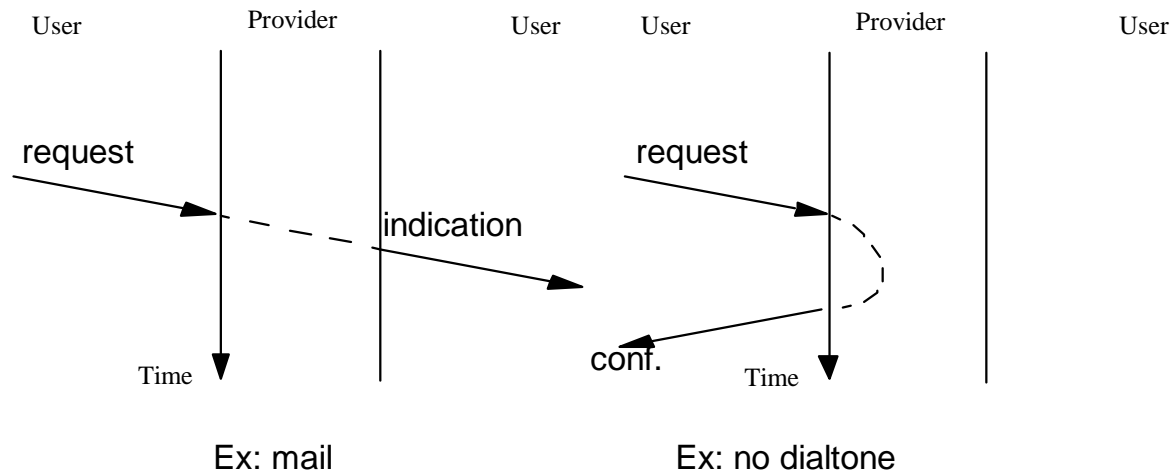
- Local interactions service user - agent
 - γ_x - commands (from the service user)
 - τ_x - data for transmitting (messages), coming with γ_x
 - g_x - return signals (to the service user)
 - t_x - received data (messages), coming with g_x
 - Service offered to the user is defined by the semantics of the set of commands and signals permitted at the local interface

- SERVICE PRIMITIVES (occur completely or not at all)
 - Request
 - Issued by the service user to invoke some procedure
 - Response
 - Issued by the service user to complete a procedure invoked (locally) by an indication
 - Indication
 - Issued by the service provider to indicate that a procedure has been invoked by the service user at the peer access point or to invoke some procedure can be issued at any time
 - Confirmation
 - Issued by the service provider to complete a procedure can be issued at any time

Service Primitives- Graphical Representation



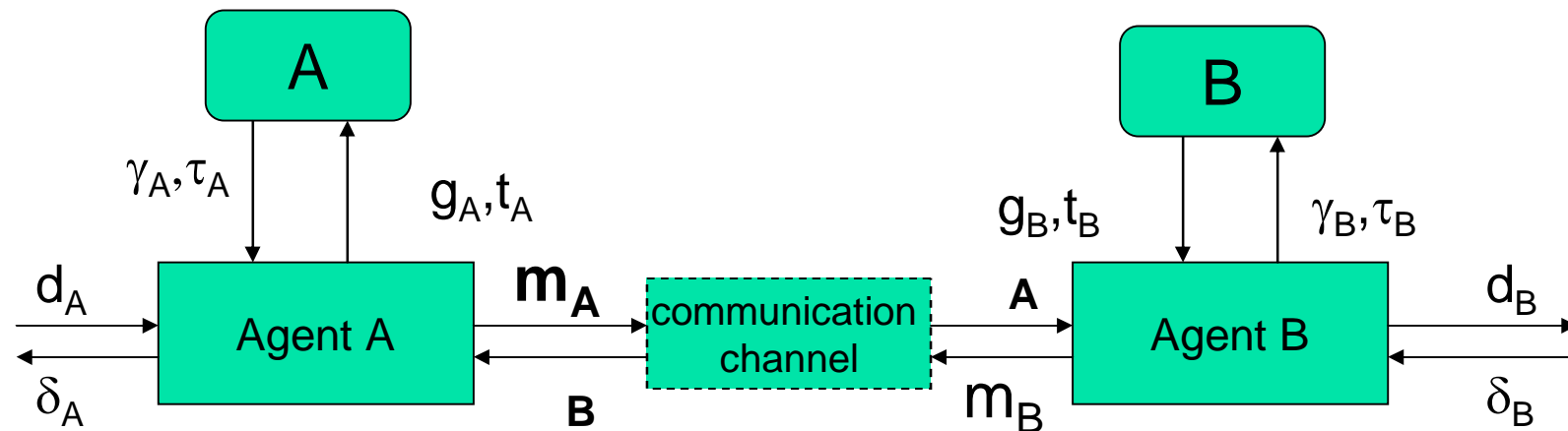
Service Semantics Defined via Primitives



Application Programming Interface (API)

- Primitives define the service semantics.
 - Car analogy: turn right, turn left, brake, accelerate
- The API is the detailed way in which this is done – access syntax...
 - A wheel? A stick? Take a car for a handicapped?
- Why are the “APIs” of cars so similar?

Communication Protocols



- Data exchanged **between** the (peer) agents
 - $m_x = m_{xp} + m_{xe}$ (send: the payload + the envelope)
 $\mu_x = \mu_{xp} + \mu_{xe}$ (Received: the payload + the envelope)
 - The (optional) payload contains information from the messages (possibly part of). Semantic has to be preserved - the syntax may be different!

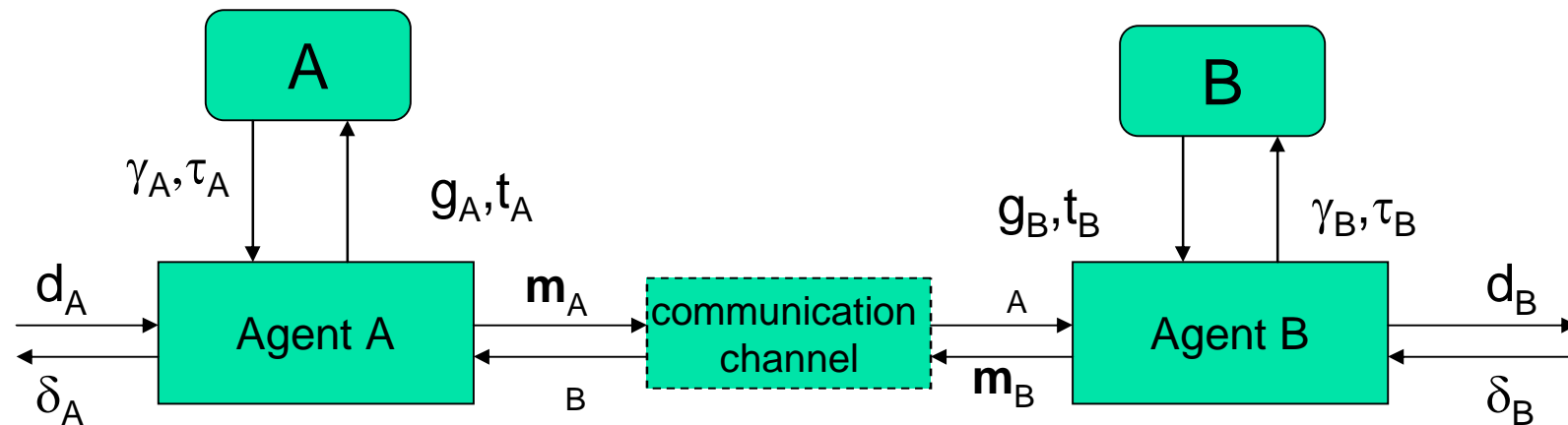
Common to both partners

- Vocabulary

Permitted values of m_x and μ_x belong to finite, strictly defined vocabulary M_x and N_x , which must be *common to both agents*.

- Encoding of the packets must be understandable by both interlocutors.
- Operational procedures defining the reaction of each partner on the incoming stimulus $(\gamma_x, \delta_x, \mu_x)$ have to be defined for both partners in a consistent way.
- Operational procedures may be dependent upon the features of the communication channel.

The protocol execution environment



- Interactions with the local environment
 - δ_x - calls of local supporting functions (e.g. timer setting, starting the channel operations)
 - d_x - return signal from local functions (e.g. timer expiration, signaling completion of channel operations)
 - $g_x, \gamma_x, d_x, \delta_x$ belong to finite, locally defined, sets of permitted values. These sets do not have to be identical (neither semantically nor syntactically) for both partners

Elements of a Communication Protocol

- Definition of the communication protocol
The definition of the communication protocols consists of five essential elements:
 - The service to be provided by the protocol
 - The functionalities needed from the execution environment
 - The vocabulary of envelops
 - The encoding (format) of the envelops and payload
 - The operational procedures

➔ *Compare: The way human interaction goes?*

Services vs. Protocols. Standardization.

- Service specification is what matters for the user!
- Protocol specification is what matters for the peer entities providing the service.
- **Peer entities have to adhere precisely to the same protocol!!**
- Standardization is crucial
- Note: Several protocols (open or proprietary) can support a given service and assure the desired service semantics.

Protocol Specifications - FSMs

- Finite State Machines (FSM)
 - $\Sigma = (S, E, D, S_0)$
 - $S = \{s^j\}$: countable state space
 - $E = \{e^j\}$: countable set of events
 - D : transition function
 - $D: S \times E \rightarrow [S \cup \phi, A]$
 - ϕ : undefined transition (zero element)
 - $A = \{a^j\}$: countable set of actions
 - S_0 : initial state
 - $E^f(s)$: feasible event set for state S
 - $\{e_k\}_k = 0, 1, \dots,$
 - $\{s_k\}_k = 0, 1, \dots,$

**Please
compare with
the
interactions of
the
interlocutors
and their
users!**

Send_and_Wait Specification

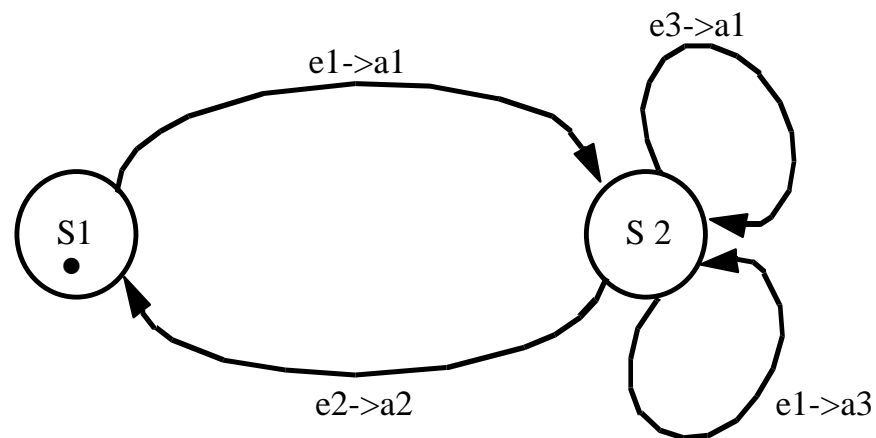
- Example - Send and Wait

- s^1 - idle e^1 - data to send (Req)
- s^2 - waiting e^2 - get acknowledge
- $s^0 = s^1$ e^3 - timer expired
- $a^1 = \langle \text{send data, set timer} \rangle$
- $a^2 = \langle \text{acknowledge transfer, clear timer} \rangle$ (Conf)
- $a^3 = \langle \text{response: busy} \rangle$ (Conf)

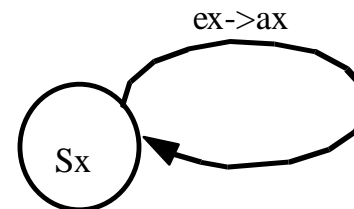
- Transition function

	State	
Event	S1	S2
e^1	a^1, s^2	a^3, s^2
e^2	0	a^2, s^1
e^3	0	a^1, s^2

Protocol Specification: Graph usage



Receiver



- S^x - idle
- e^x - data arrived
- a^x - acknowledge (+Ind)
- Note:
 - Timers count is irrelevant
 - Infinite loop if permanent transmission errors!
 - Duplication if acknowledge gets lost