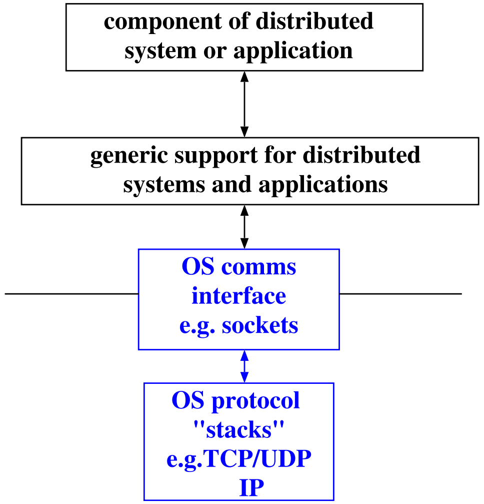
Communications Support for Distributed Systems and Applications



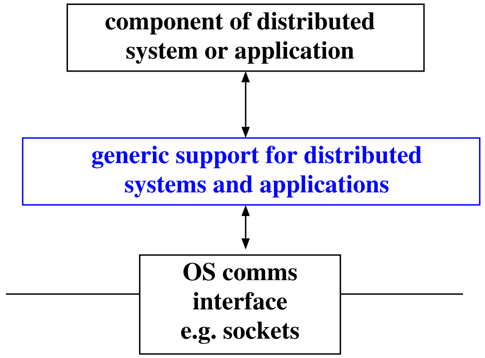
The OS interface:

sockets provide a programmer's interface to a selection of communications protocols

sockets are created and used by system calls to send to e.g. IP-address/port-number

- byte streams
- packets of unstructured bytes (datagrams)

Alternatively, the OS interface may be designed to support distributed objects
and the API may be defined in terms of objects' ports
with system-wide naming of port-IDs
e.g. Mach, Chorus,......



"generic support for distributed systems and applications" builds on OS-level communications services to support the development and execution of distributed software

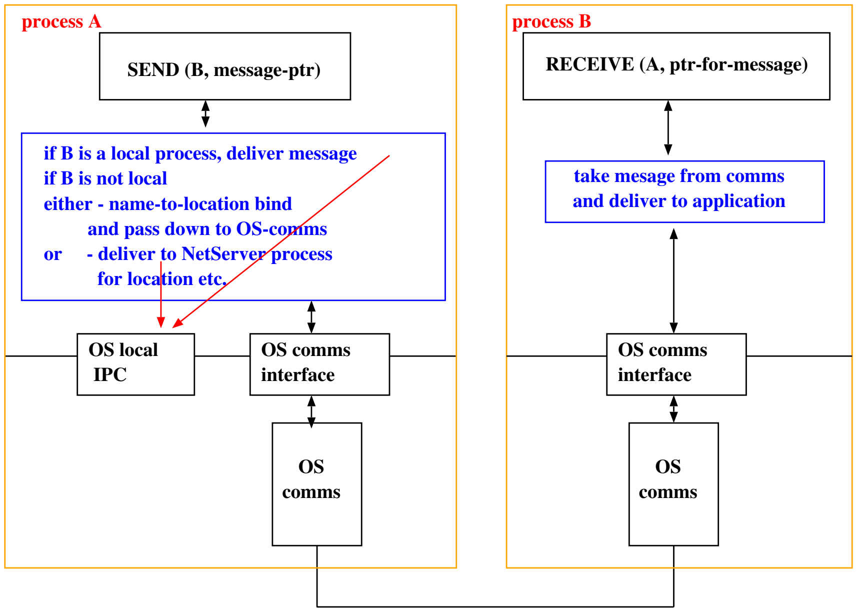
As discussed in the introduction, at this level we need support for
(depending on the software model which defines the communicating entities):

- naming
- location
- name to location binding
- communication
  - message transmission?
  - call to server-interface.procedure (args)?
  - object invocation?
- authentication
- access control

Asynchronous message passing

ref 1B concurrent systems, Dip/2G OS Foundations

Message passing maps naturally onto distributed communication, provided the communicating entities are named and located system-wide



Message-orient(at)ed middleware (MOM)

IBM Message service: MQSeries

one-to-one reliable message-passing
used under e.g. CICS transaction processing
naming is of queues, routing is via queues
http://www.software.ibm.com/ts/mqseries

messages are not typed but have some structure so that language-level type systems can be built above them

current interest in moving to XML

there is a JMS interface for MQ

MOM: publish-subscribe systems

any process who has subscribed to a subject receives messages on it

subscription may be subject-based or content (field/value)-based

need a subject naming scheme and a yellow pages service

TIBCO TIB/rendezvous message passing
Reuters news service (applications
Stock market quotation service rather than middleware)

Message systems have a larger proportion of the middleware market than O-O systems
e.g. IBM 24%, TIBCO 17% (1998?)
What has been the effect of the web services paradigm since then?

Early middleware research

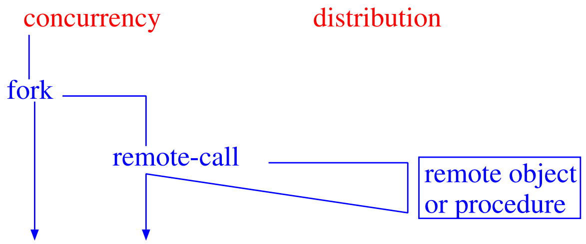
message-passing was thought to be difficult to program
- matching requests and replies

it was argued that software structure and pattern of use tends to be based on client/server or object models i.e. synchronous invocation

language-level communication paradigm is
  procedure call
  object invocation

* multi-threaded programming became more commonplace at the application level

so use a thread to make a blocking remote service call or remote object invocation and continue local work in other threads



RPC systems were developed in research projects
(e.g. Mayflower and Unison RPC, Cambridge CL, mid 80's,
ANSA RPC under Alvey, then APM, now Citrix Cambridge)
then became incorporated into standards such as
ISO-ODP, OSF-DCE

RPC is built above request-response message passing but message passing may not be visible to and programmable at the application level

* BUT multi-threading also makes the programming of message passing more tractable

the main distinction is synchronous, closely coupled communication
  (as in RPC and O-O)
versus asynchronous, loosely coupled communication
  (as in message-passing)

Give the application the choice?

with message passing only:
  doesn't extend language-level paradigm
  doesn't model service invocation well

with object invocation only:
  doesn't support large objects and streams well
  assumes components closely coupled (all up-and-running)
  difficult to get immediate response to events

suppose an object is a source of events to which an application should respond asap:

polling:
client polls server at some period
response is delayed by on average half that period
  either: overload comms with polling
  or: respond sluggishly

synchronous callback:
server calls interested clients on event occurrence
clients can delay server
need multi-threaded servers
complex to program for delayed threads

current O-O middleware platforms provide event services

Java RMI/Jini + events
  single language, proprietary

OMG-CORBA event notification service 1998
  multi-language, open interoperability

CEA (Cambridge Event Architecture)
  early 1990's research
  extend any O-O platform

These platforms give the choice of synchronous/asynchronous communication but they still assume closely coupled components are communicating.
General MOM is asynchronous and loosely coupled.

Cambridge Event Architecture (CEA) 1990s

- compatible with any style of middleware

- use standard data typing for named, parametrised events
  e.g. IDL -> ODL, XML?

- event sources publish the events they will notify

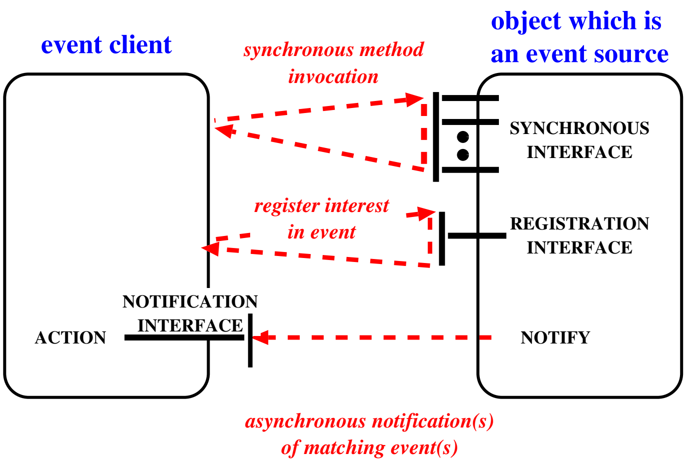
- clients register interest in events with sources
  indicating parameter values or wildcards

- sources notify clients with the stream of matching events

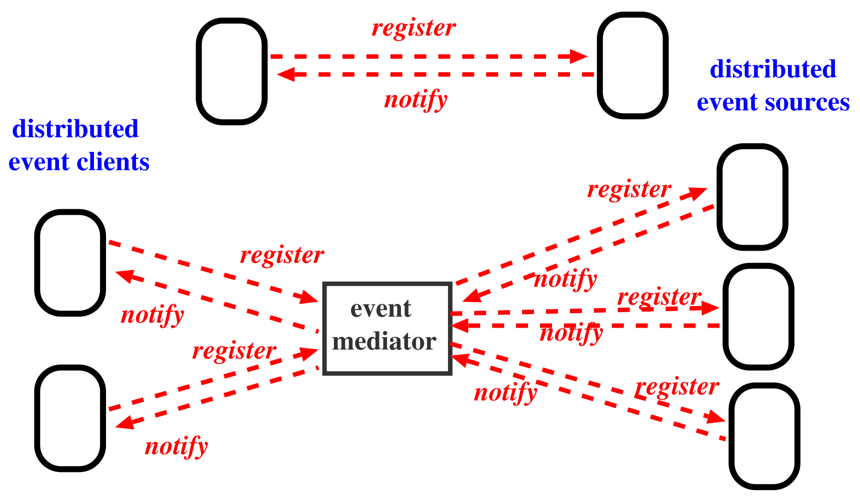
- event stores can be clients e.g. to log events
  note compatible transmission and storage technology

Cambridge Event Architecture (CEA)

CEA 1. The *publish-register-notify* paradigm



CEA 2. Direct and mediated notification



CEA 2

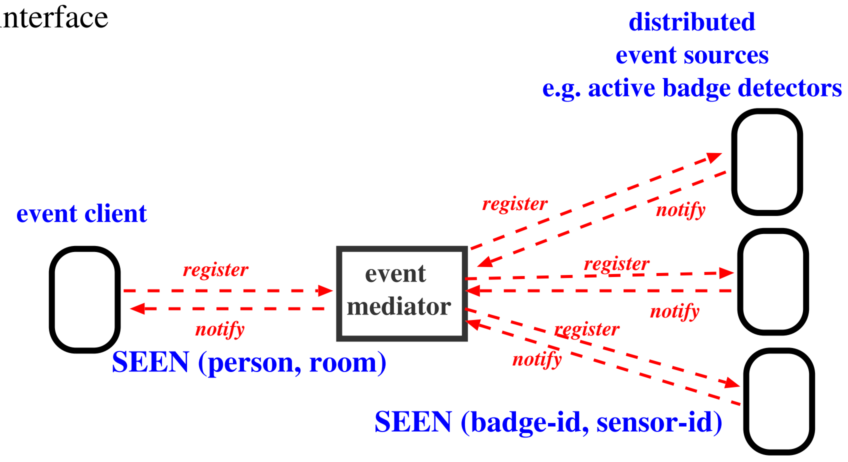
* avoid overload on primitive event sources

* decouple event source and client

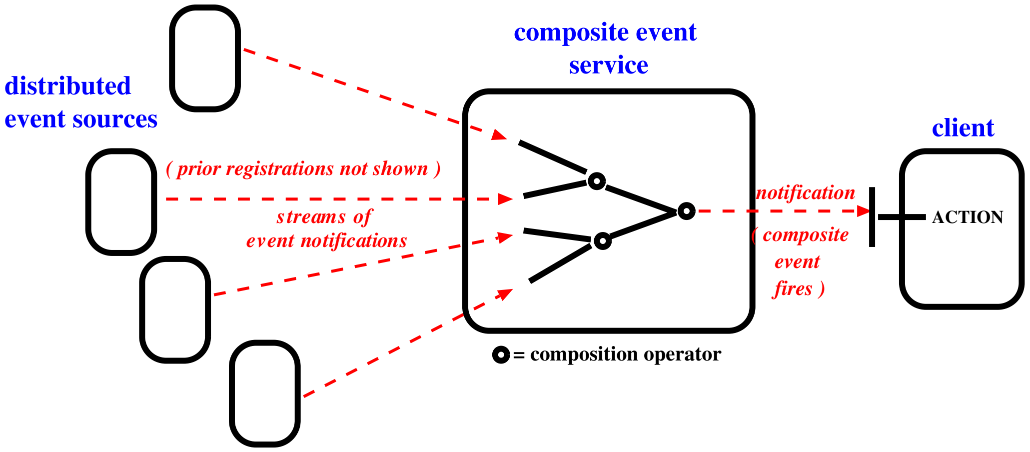
* one-to-many and many-to-many communication

  - multicast protocol may be exploited
    at event source or mediator

* mediated communication can be used to provide
  a higher-level interface



CEA 3. Event Composition - Composite event detection



CEA3 event composition operators

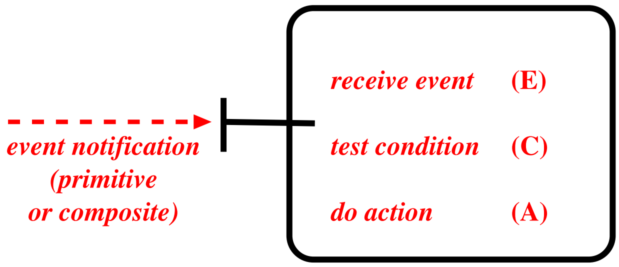
Without	**A - B**	*yields stream matching A until B occurs*
Sequence	**A ; B**	*A followed by B*
Or	**A | B**	*yields stream matched by A or B*
And	**A&B**	*yields stream matched by both A and B*
First	**First(A)**	*yields the first event that matches A*

- need to be tested in practical applications

- precise meaning? .... consumption policy?

CEA 4. Active programming: *event-condition-action*

(Component composition)



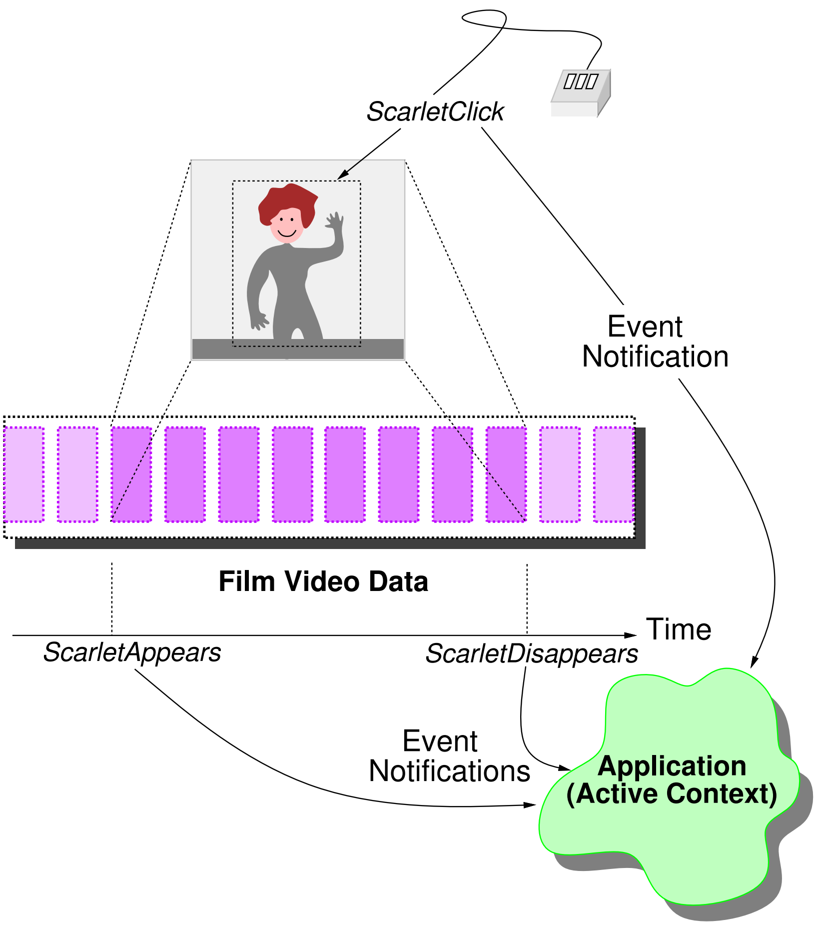
event composition embodies some conditions

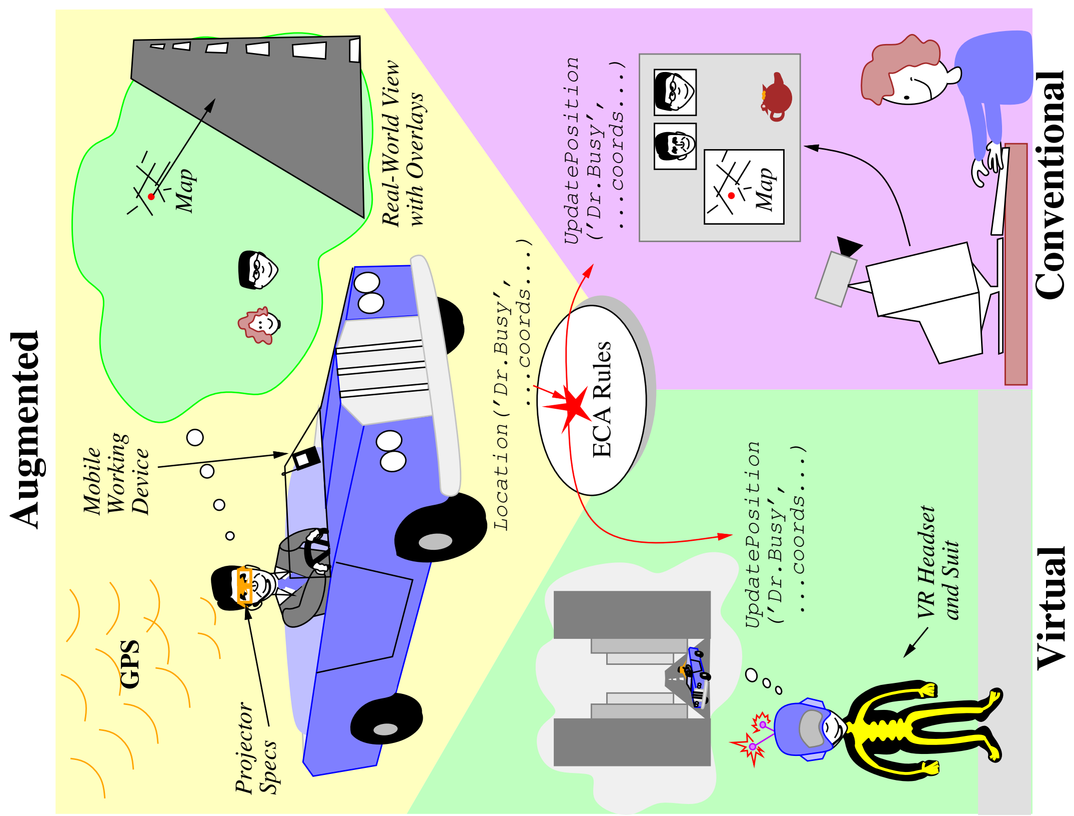
the receiver may also impose conditions

* **events are the glue for composing distributed software components**

- **active office, home, airport, city (sensor-rich environments)**
- **virtual reality, augmented reality**

C-14





Active Badge (electronic tag) Technology, Sensor-rich Environments

e.g. active house, office, hotel, airport, city

?✓ locked doors will open

✓ the nearest computer will fetch your environment, video streams, email, news, buffered events

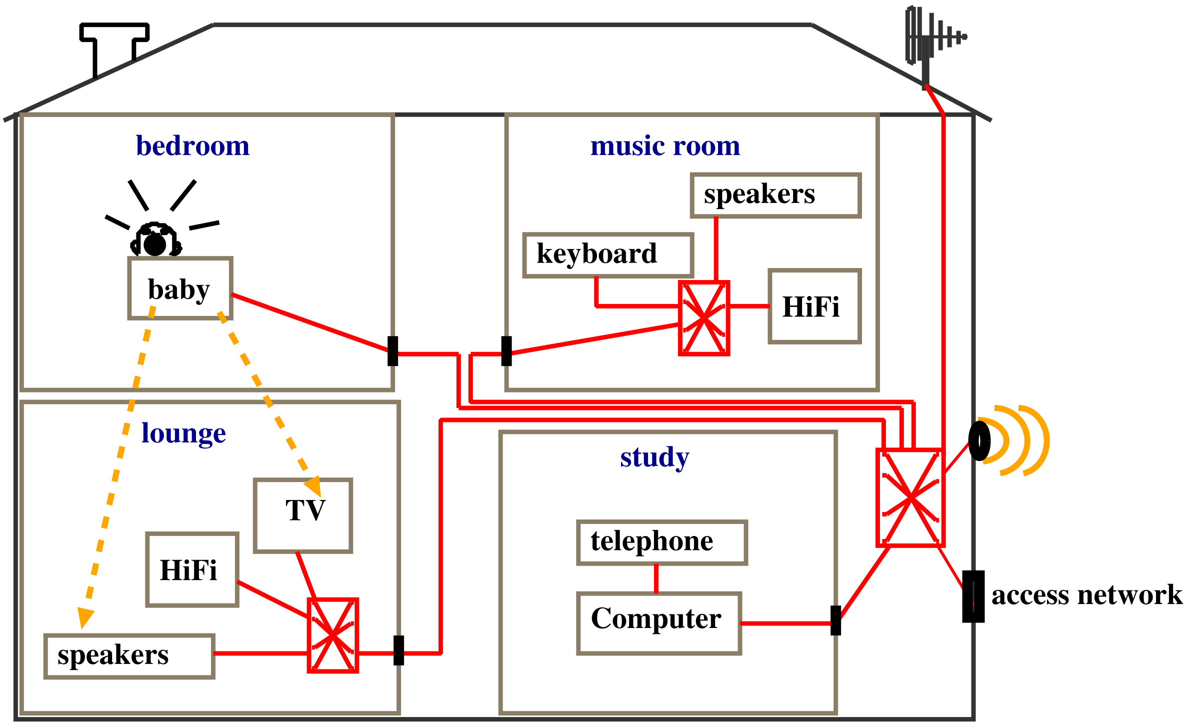
✓ equipment can be tagged for security - movement raises an alarm

✓ mobile objects can be tracked (buses, cars, taxis, ambulances)

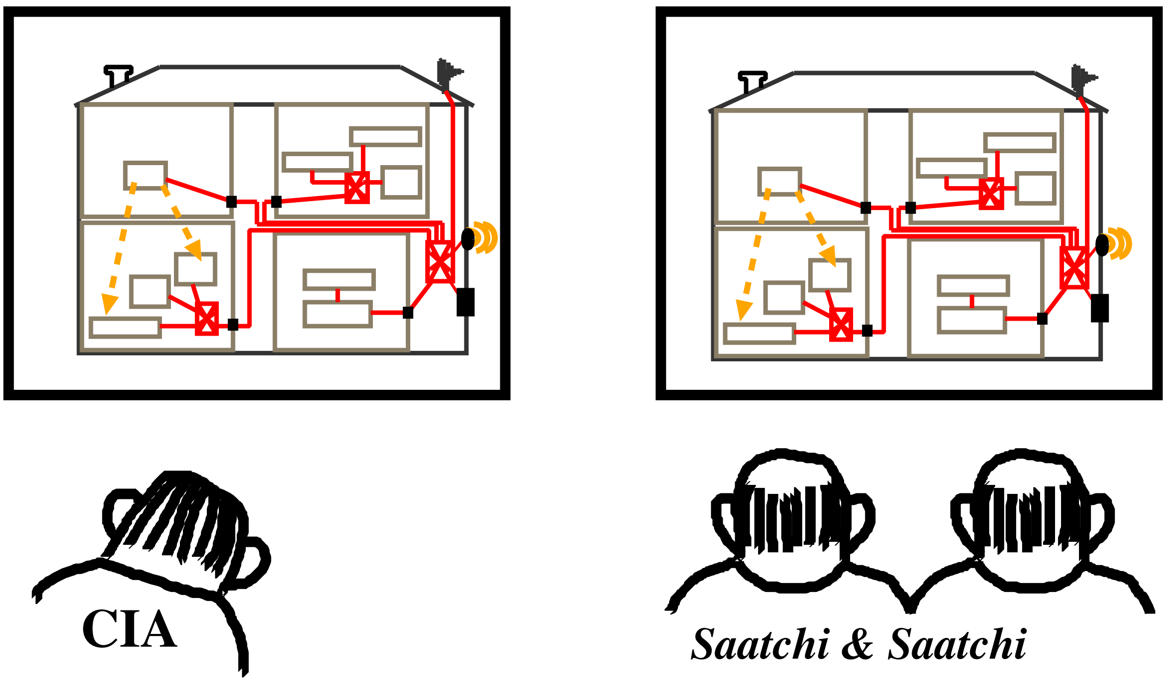
? people can be tracked, meetings can be detected

-> access control needed on registration and notification

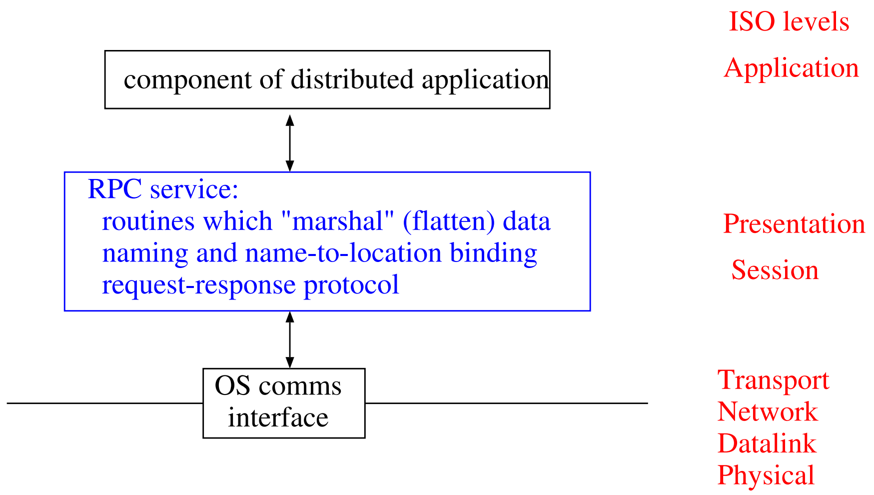
An Active Home



......but we can be monitored .....

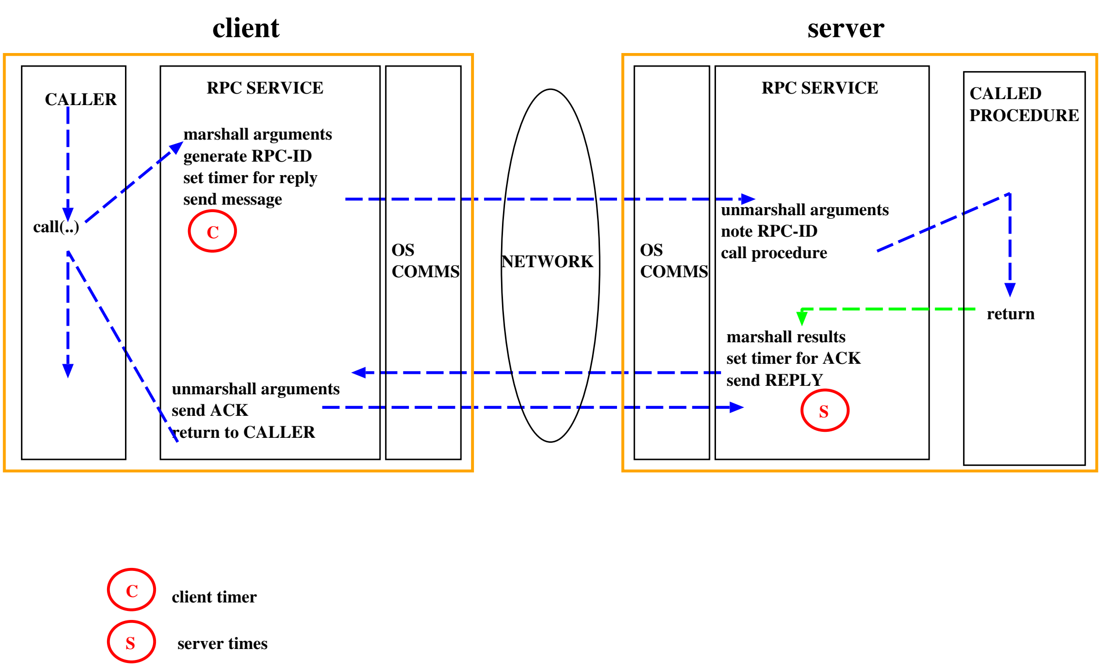


Remote Procedure Call (RPC)



examples: Mayflower/CCLU RPC, SUN RPC, ANSA RPC, MSRPC
Xerox Courier over XNS (SPP, Ethernet)
ISO-ODP, OSF DCE

RPC Request-Reply Acknowledge (RRA) protocol



RPC semantics

recall that client, server and network may be congested or may fail independently of each other
(fundamental property of Distributed Systems)

RPC systems may offer AT MOST ONCE or EXACTLY ONCE semantics

(C) if the client timer expires:

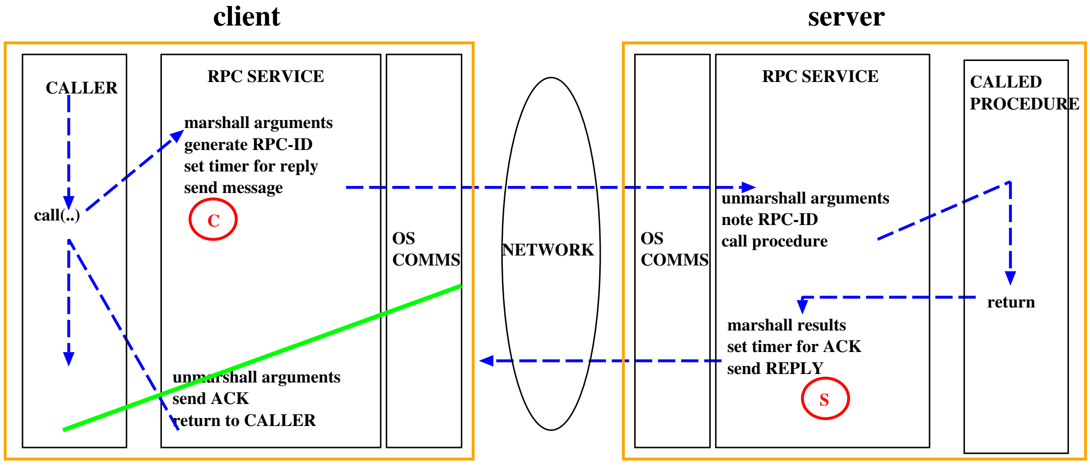
AT MOST ONCE semantics:
exception return to the application
it is likely to repeat the call but this is not detectable
i.e. it will have a new RPC-ID

EXACTLY ONCE semantics:
retry a few times
RPC-ID means that the server can detect repeats
if no reply, exeption return to client

(S) if the server timer expires:

resend results
RPC-ID means that the client can detect repeats

RPC client crash

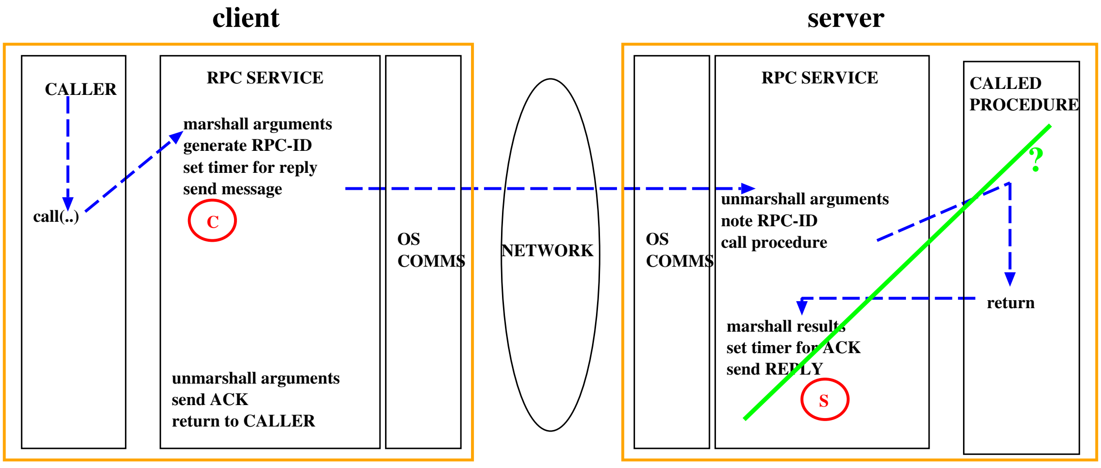


results are sent to crashed machine, are not acknowledged, and server timer S expires repeatedly on resend

persistent state may have been changed by the procedure call - should this be handled by RPC service?

NO - application-level transaction semantics (commit/abort) should be used.

RPC server crash



The server fails at some stage during the call. Results are not sent and the client timer C expires repeatedly

persistent state may or may not have been changed by the procedure call - should this be handled by RPC service?

NO - application-level transaction semantics (commit/abort) should be used.

Integration of Programming Languages and RPC (1)

* some early RPC systems aimed for complete distribution transparency

e.g. Xeroc PARC, Mesa language, Courier RPC
a preprocessor detects which calls are not to local procedures
and replaces them by calls to RPC support

problem of incorrect procedure names that don't exist anywhere
problem of call semantics for some arguments

* Cambridge Mayflower system, CCLU RPC - made distribution explicit

the compiler was changed
different syntax for definition and call of procedures that can be called remotely
BUT - this was still for a single language, CCLU

some RPC systems restricted the argument types
e.g. SUN RPC: C base-types only

CCLU RPC: most types including procedure names defined since developer supplies marshalling and unmarshalling routines for constructed types (recursive descent)

Integration of Programming Languages and RPC (2)

* ANSA RPC, was initially developed for C
  but later also supported C++ and Modula3 - a very early heterogeneous system
  - defined a Distributed Programming Language (DPL)
  - DPL statements are embedded in the programming language, and tagged
  - a preprocessor detects these statement, replaces them with calls to RPC service

All RPC systems automatically generate marshalling and unmarshalling routines to flatten call and return arguments into packet format suitable for transmission, and unpack them on receipt. These routines are programming-language-specific.

Now assume that we wish to support a number of different programming languages, i.e. components written in different languages can interoperate

* the standard approach (ANSA, ISO-ODP, OSF-DCE), O-O platforms
  - define an Interface Definition Language (IDL)
  - provide mappings for programming language's type systems onto IDL
  - (internally) define the transfer syntax for IDL types
  - IDL compilers generate marshalling and unmarshalling routines appropriate for the programming languages involved.

(CORBA calls the invoker's marshalling routine a STUB and the invoked object's unmarshalling routine a SKELETON)

Integration of Programming Languages and Middleware

* how do platforms that support objects and object invocation differ from the RPC schemes described above?

(as above for IDL and STUB/SKELETON generation)

RPC systems name and identify interfaces and procedures

e.g. ANSA IDL has base and constructed data types and the InterfaceRef type, an instance of which is a reference to a loaded and running instance of a service's interface

O-O systems name and invoke objects

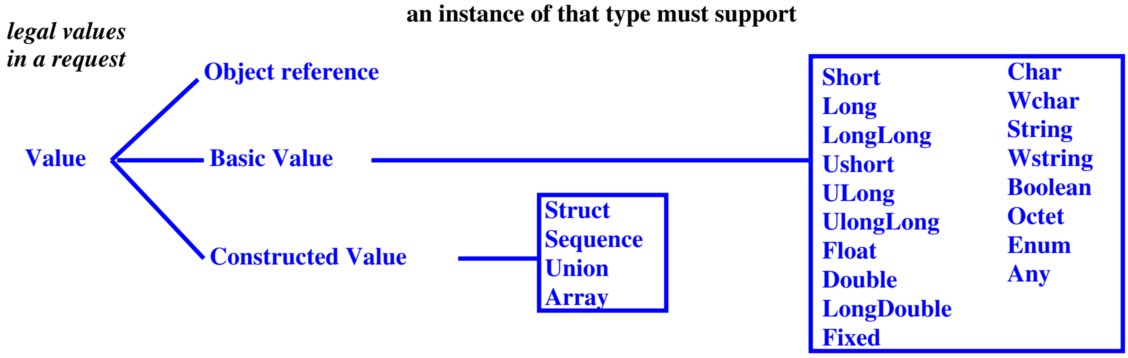
Externally invocable objects must be registered with the platform,
an object-ID is returned (and may be recorded in a name service)
The object becomes known globally and may be invoked remotely
Object-IDs are first-class values which may be passed as arguments

example: CORBA IDL

object type — **members are object references**

base types —
16, 32, 64 bit signed and iunsigned 2's complement integers
single (32), double and double-extended floating point
fixed-point decimal
characters
boolean
8-bit opaque, NOT converted on transfer between systems
enumerated types
string
any (container)
wide characters and wide character strings

constructed types —
record (struct) ordered set of (name,value) pairs
discriminated union
sequence
array
interface type - specifies the set of operations which an instance of that type must support



Where does XML fit in? http://www.w3.org/XML

SGML - standard generalised markup language
1985 document standard

XML - document standard (W3 consortium) compatible with SGML
DTD - document type description
- tag types - graph-structured document
XSL - style sheets indicate how to display the document e.g. in HTML
HTML - hypertext markup language
embedded tags are about how to display

XML is becoming widely used as a transfer syntax
- for documents - as expected
- for general typed messages (all types reduced to strings - external form)

SOAP - simple object access protocol
object invocation defined with call and return arguments as XML types

wide interest in XML for use in message oriented and database access middleware