Middleware Technologies

Franco Zambonelli

March 2014
Università di Modena e Reggio Emilia

Outline

- Why Middleware?
  - Problems of open and situated distributed computing systems
  - Specific problems of service-centric systems
- What is Middleware?
  - Basic features
  - Middleware Models
    - Interaction models
    - Services to be provided
    - Implementation models
- Overview of Technologies
  - J2EE
  - CORBA
  - .NET

Modern Distributed and Service Systems

- Modern distributed systems (as well as non-distributed ones), need to be ***adaptive***, that is
- Open
  - New components and services join the systems, while others may leave
  - We cannot re-configure and re-design the system any time it changes (economically and practically unfeasible)
  - We cannot rely on "a priori" information about each and every component that will be part of the system (impossible, new specification may be known only a posteriori)
  - In addition, components may be heterogeneous (developed with different technologies or for different platforms)
- Situated and Context-aware
  - Components of a system may be deployed (or services exploited) in partially unknown environment (computational of physical)
  - Some info about the actual environment must be dynamically retrieved at deployment time or at run-time (cannot be a priori coded)
  - The environment can have its own dynamics, and it is necessary for the components of a system, and for the system itself, to dynamically adapt to such dynamics

3

An Example: a Problem…

- Consider a distributed application
  - Made up of a set of components on different computers
  - That interact with each via TCP Sockets

```
Socket s = new Socket("155.185.3.2", 143)
s.write("print me this line please!")
// I must know a priori that such a print server exist
// at a specific IP and at a specific port!!!
// I can do nothing if – at some time – a better printer
// gets installed in the network…

JFrame jf = new JFrame("Hello");
Jf = setBounds(0,0,360,140);
Jf.show();
// I must a priori know that the user display supports such
// a dimension of the frame. But what if the users wants
// to run its application on a 120*120 Nokia phone display?
```

4

…and a Possible Solution

- Suppose we have a "Dr.Know" able to discover on demand:
  - Which printer services are available
  - The characteristics of the PC
- Then, Dr.Know would be a middleware…
  - Indeed, in service-oriented architectures the "discovery" problems is usually seen as the key problem
  - But the is much more to middleware than that…

```
Vector printers = DrKnow.Discover("print_services")
// returns a Vector with all the printers currently available
For (int i=0; i<Vector.length; i++)
    printers[i].getSpeed();
//select fastest printer available and print
s.write("print me this line please!")

Display disp = DrKnow.getContext("Display");
Jf = setBounds(0,0,disp.x,disp.y);
// size according to the dimension of the current display
```

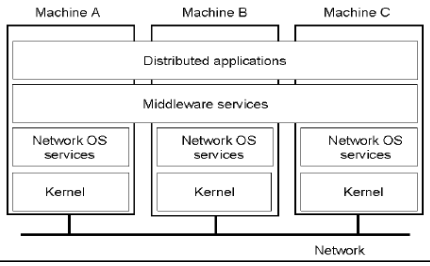
What does Middleware?

- Enabling Interactions
  - Acting as the uniform glue that collate components in the systems
  - Facilitate components interactions (e.g., supporting naming and dynamic discovery of services) → the discovery of service-oriented architectures
  - Dealing with heterogeneous components (e.g., components and services developed using different technologies)
- Supporting Interactions
  - Provide solutions for common problems of interactions (e.g. inconsistencies and synchronization in accessing shared resources, persistence)
  - Provide support for openness (new components getting in the system)
  - Provide support for problems (fault recovery, replication)
  - Provide support for system manager (monitoring, and logging)
- Promoting Context-Awareness
  - Have components be aware of "what's happening" (e.g., a new component has connected, a component changed its state, the room temperature has changed)
  - Virtualization of environmental resources into digital resources (e.g., a thermostat as a software object)

6

Where is Middleware?

- Middleware act as an “middle” layer between the “os-ware” and the application software
  - Applications uses the services of the middleware
  - The middleware uses the services of the network layer of the operating system (and of the operating system in general)
- To some extent, middleware can be considered as a sort of
  - “operating system” for “distributed systems” instead of
  - operating system for a computer



7

DNS as a Basic Middleware Service

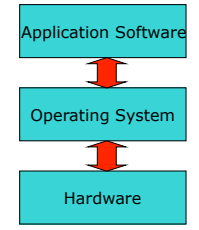
- DNS decouples actual IP from symbolic name
  - Enables dynamicity of IP
  - Enables interactions to be based on “names” of resources rather than on “IPs”
  - Sometimes, this enables an automatic forwarding to the best resource
  - E.g., replicated web services with policies for IP selection (Google)
- But this is definitely not enough…
  - The names must be known and are fixed
- In general, the current Internet and Web architecture
  - does not provide middleware services, but only basic mechanisms to enable interactions

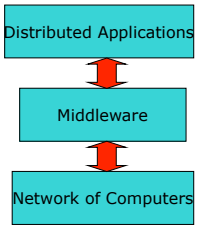
```
Socket s = new Socket(www.printersite.com, 1234);

s.writeln(“write this line”);
```

Operating Systems vs. Middleware

- Operating Systems
  - Provide high-level abstractions for the resources of a computer
  - Facilitate and orchestrate access to resources
- Middleware
  - Provide high-level abstractions for the resources of a network
  - Facilitate and orchestrate access to distributed resources

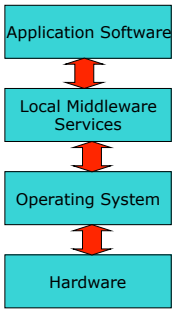




9

Middleware for Local Services

- Specific types of software infrastructures also act as “local middleware”
  - An additional layer above the operating system
  - To complement it with additional special purpose service
  - To add support for openness and orchestration of local programs
- Examples
  - TomCat and the Servlet Context
  - Java
  - Typically as components of a “larger” middleware environment (e.g., Java → J2EE)



10

TomCat as a Middleware for Service Composition

- TomCat (i.e., J2EE) provides several services that can be considered as “local” middleware services
- Servlet Context
  - A sort of “shared dataspace” to enable Web services to share data and contextual information
  - A local “naming” and “discovery” service, to enable services to share Java objects (attributes)
- Servlet Sessions
  - A sort of additional “contextual information”
  - Enable services to keep track of “history”, and to adapt their execution depending on such history
- All of this make the Web server “adaptive”
  - New services can be deployed and adapt their execution to the context
  - Services can adapt their behavior depending on history

11

JAVA as a Middleware

- The JAVA environment, per se, can be considered as a sort of middleware
- In fact, it provides a number of additional services over the operating system
  - Supporting dynamic class loading: new classes can enter a Java program dynamically (openness)
  - Take care of finding classes autonomously in the file system
  - Provide a service for freeing memory (garbage collection)
  - Provide support for heterogeneity: its applications can execute on any type of computer, thanks to the JVM
  - Provide support for event and exception handling: a primitive form of context-awareness (system level and user level events can be caught)
  - In general, provide support for service-oriented systems and associated standards/mechanisms
- But Java – in its “Enterprise Edition” is also a truly middleware for distributed systems

12

Middleware Models

- RPC and Object-based
  - Rooted in the "Remote Procedure Call" paradigm
  - Support distributed object applications (e.g., remote method invocations)
- Event-based
  - Rooted on interactive computing models
  - Support a reactive context-aware model
- Shared Dataspaces
  - Rooted on shared memory models
  - Support sorts of "stigmergic" interactions between components
- Most middleware technologies, handles both objects, shared memories, and events
  - And provides different services for the different types of models
- These model, more than "interaction models" are best known as "**coordination models**" in that they provide
  - Communication between components
  - Synchronization of activity between components
  - In general, orchestration (i.e., coordination) of activities between components

13

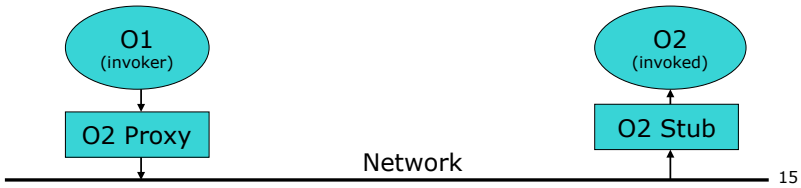
RPC and Object-based Middleware

- Support
  - Application based on Distributed objects
  - That invoke each other methods as if they were local objects
- Basic services to be provided
  - RPC (remote procedure call) or remote method invocation (RMI)
  - Publication and discovery of objects and their methods (also called "Naming" or "Lookup" services)
- Additional services
  - "attribute-based" discovery
  - Heterogeneous Interactions
  - Transactions, Recovery, Load Balancing, Replication

14

Remote Method Invocation

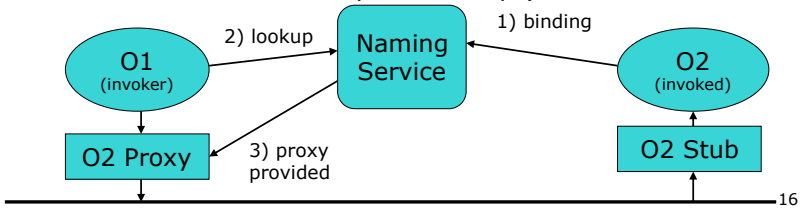
- A Service that enable an object to invoke the services of another non-local object as if it were local
  - The object that must be invoked remotely must
    - be compiled with special tools provided by the middleware (e.g., the rmic compiler in Java RMI)
    - to generate a “stub” that receive remote method invocations and transforms them into local ones
  - The invoking object must receive a special reference to the objects, that act as “proxy copy”, and that will provide to forward invocations to the remote stub



15

Naming and Discovery

- A special “Naming” service component (e.g., the RMIregistry in Java) of the middleware takes care of connecting the invoker and the invoked
  - The invoked must communicate its public interface to the Naming Service and must make known itself via a “public name” (binding of name)
  - The Naming service act as a “Yellow Pages” service
  - The invoker asks a reference to an object with a specific name to the Naming service, which will provide a proxy to it in return (this operation is called “discovery” or “lookup”)



16

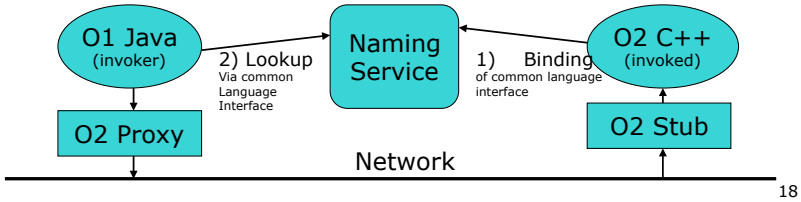
Attribute-based discovery

- The classical Naming service (as in RMI)
  - Enables two objects, previously-unknown to each other, to interact
  - BUT there must be a priori agreement on the name
  - So, it is not very adaptive
- Solution: attributed-based discovery
  - An object does not bind itself to a simple name, but to a set of attributes, and publish these attributes on the Naming service
  - *("printer", "laser", "color", "8ppm")*
  - *("trains", "timetable", "Italy")*
- When another object needs specific services, it can ask to the Naming services for objects with attributes of interest
  - *("printer", "laser", *, *)* → I need a laser printer, no matter if it is color or slow
  - *("train", "timetable", *)* → I need a service for train timetables
- And it obtains in return proxies to all services "matching" the requested attributed
  - This clearly makes the system more adaptive to dynamic changes and more suitable to open systems
  - There is little to know in advance, most information can be obtained on-the-fly
  - It requires the naming service to do the "pattern matching" work
- Implemented in the so called JINI middleware (now part of J2EE), to support more dynamic "plug&play" distributed object applications

17

Heterogeneous Interactions

- The stub and the proxy
  - Decouple the invoker from the invoking objects
  - They interact with the mediation of these components, provided by the middleware
- Therefore, one can think at the two objects being different in terms of
  - Programming language, Technologies, Type systems, Interface specification
- Provided that
  - There is a ***common interface language*** to publish interfaces on Naming service
  - The stub and the skeletons to the necessary "translation" work
- Any two heterogeneous object can interact
- The pioneer system in this area is CORBA (IDL, Interface description language). Now, there are standard way of publishing interfaces and systems descriptions
  - Exploiting XML and according to the SOAP standard
  - Implemented by most middleware systems



18

Other Services

- Transactions
  - The middleware can make "critical" operations be part of a transaction, All-or-nothing semantics
    - A set of operation (method invocations) are executed as an atomic unit
    - If one operation fails the system "roll back" – i.e., recovery – to its previous state
  - E.g., transferring money from a bank to another
- Synchronization
  - Critical operations/methods on an object or on a set of objects should be performed without having other objects act concurrently
  - The middleware can provide at maing specific actions "mutually exclusive", this synchronizing the accesses to shared resources/object
- Load Balancing
  - Among a set of equivalent services/objects, the middleware can take care of automatically providing a proxy to the less loaded one
  - Many popular Web sites (e.g., google) exploit a middleware doing this, to distributed request among several machines
- Replication & Caching
  - Several intensively accessed services/objects can be automatically replicated by the middleware in several locations, so as to Improve overall performances and tolerate local failures
  - The Web, actually has a caching system…
- Quality of Service
  - Ensure that operations are served within a specific time, i.e., with a guaranteed quality
- All these additional services may clearly also be present (in sligthly different forms adapted to the model) for event-based and shared dataspace middleware systems…

19

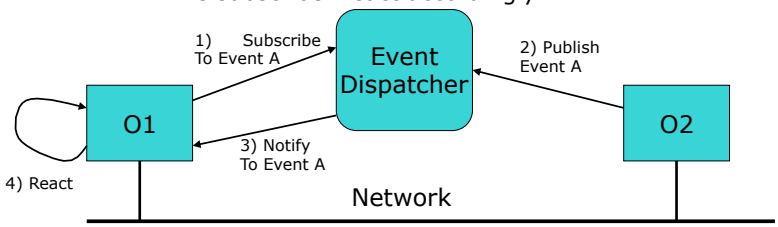
Event-based Models

- Interactions are based on the key concept of "events"
  - "Some that happened"
- Publish-Subscribe Model
  - Generated event can be "published", i.e., made known to the audience
  - Interested components can "susbscribe" (i.e., declare interest) to specific types/classes of event
  - A susbscriber gets notified of relevant published events
  - And react to it by executing a proper "event handling" method
- Various types of events can be conceived
  - System level events (new hardware, clock, system error, etc.)
  - OS Level events (i.e., file open, file unlocked, device ready, etc.)
  - Network level events (new node connected, node disconnected, transmission error)
  - Application level events (new object created, new message arrived, data changed, etc.)
  - User level events (i.e., interactions with the program)

20

Publish-Subscribe Interactions

- A functionality of the middleware to subscribe to events
  - Tells the middleware what events a component is interest in
  - Associate a reaction function/method to events (event handler)
- A functionality of the middleware to generate events
  - Application level components can decide how and what events to generate
- Upon generation of an event
  - The middleware catch the event
  - It “delivers” (“notifies”) the event to the subscriber
  - The subscriber react accordingly



Models of Subscriptions

- By Name
  - Events are published with a name
  - Subscribers declare interest in specific names
  - Has the same problems of Naming in RMI (not dynamic, not adaptive)
- By Type
  - Events can be Typed (i.e., belong to a specific class, as in Java)
  - Subscriber declare interest in specific types (e.g., as in Java GUI, ActionEvents or DocumentEvents)
  - Has the problem that either the types are many or a subscriber receives too many notifications in which it may not be interested
  - But if the types are too many, this resembles a naming scheme
- By Attribute
  - As in attributed-based naming of services
  - Enables adaptive subscription with little a priori knowledge
  - Clearly requires the Event Dispatcher to analyze events and subscription and make the publish-subscribe work
- Events with a “Content”
  - Event can also “contain” things, i.e., additional descriptions, data, attachments
  - So, they can be used to pass data from one component to another, as a message

Models of Notification

- Synchronous
  - As soon as an event arrives, the event dispatcher
    - Check the matching subscriptions
    - Notify the interested components
    - Forget the event
- Asynchronous
  - When an event arrives, the event dispatcher
    - Check the matching subscriptions
    - Notify the interested components
    - Stores the event (possibly with a lease time)
    - Future subscribers can be notified of past events too
- From the subscribed viewpoint
  - The subscriber can decide to react immediately to an event
  - Or it can delay event handling to any later appropriate moment
- In most systems
  - The subscribers can specify on which events of the past (how far in the past) they are interested
  - Events can specify a time-to-live (lease time)

Space and Time Decoupling

- Event-based models have a very important characteristics to promote adaptivity (other than providing context-awareness)
- When considering the asynchronous notification mode
- Space decoupling:
  - The components do not need to coexist in space (they can be temporarily on different networks)
  - The components do not need to share a common “name space” (if based on attributed matching)
- Time decoupling:
  - The components do not need to coexist in time
  - An event can be generated at time T
  - The generator can die or go away
  - The subscriber can arrive/born at time T+T1
  - Subscribe to the event
  - And be notified about the past event
- On the other hand, in the synchronous mode
  - Event-based interaction can be used to synchronize activities

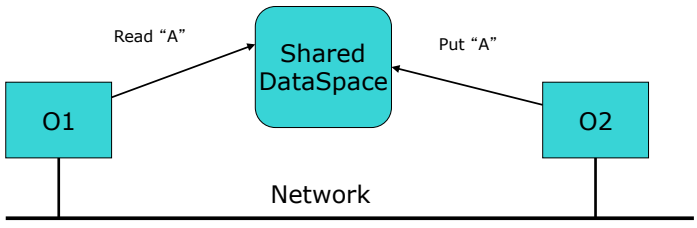
Event-based Models vs. RMI

- Advantages of Event-based Models
  - Context-awareness
    - Components can be easily made aware of what is happening in the system
  - Decoupling
    - Remote method invocation require the invoking and the invoker to coexist in space (i.e., in the same network) and time (i.e., must be on execution simultaneously)
    - Event-based models enable space and time decoupling, better suited to a dynamic world
  - Message-passing
    - Since event can have a content
    - And can solicit the execution of a method
    - Event-based models can "mimic" RMI ones
- Advantages of RMI Models
  - Well-known and used interaction models
- Get the best of the two
  - Enable RMI interactions and at the same time make components be able to handle/generate events
  - Also to discover the arrival/dismissing of objects and services

25

Shared Dataspace Models

- Components interacts via a sort of "shared memory"
  - Where they can **put** data
  - Where they can **get** data (extracting or simply reading it)
- Provided by a component acting as a shared dataspace service



26

Data Models

- Simple variables
  - The same as “normal programs” access the RAM memory
  - Put(v); Get(v);
- Tuples (or Records), i.e., ordered set of typed fields
  - (int 5, float 3.14, String “Hello”);
  - Put(Tuple); Read(Tuple);
- Object
  - E.g., Serialized Java Objects
- Structured Files
  - XML, RDF, etc.
  - In that case, accessing the shared memory may implies accessing portions of the files, or modifying (e.g., via XSL) already stored files
  - Remember the Servlet Context and the WEB-INF.xml file?
- Any File
  - Mp2, DivX, etc.
  - Accessing the dataspace means depositing and retrieving complete files
- The DataSpace model provides internal policies to
  - Organizes the data internally
  - Maximize efficiency in retrieving

27

Models for Data Access: By Name

- By Name
  - Any piece of data has an associated “name”
  - E.g., “Var V”, “Tuple Franco”, “Object John”, “File MyData.XML”
- Data is stored with associated name
  - Put(v, 5)
  - Put(Tuple, “Franco”);
- Data is retrieved by asking for a specific name
  - Int value = Read(v);
  - Tuple = Read(“Franco”);
- **Advantage**: Conceptually very simple
- **Disadvantage**: Not Adaptive, require a priori agreement on names
- Sometimes, wild cards can be used to specify names
  - Vector Tuple[] = ReadAll(“F*”);
  - Better solution, still not enough for open and dynamic systems…

28

Models for Data Access: By Content

- By Content, applicable to structured data, similar to attribute-based naming
  - Any piece of data has content (i.e., values for its fields)
  - (int 5, String "Bye Bye", char "c", float 4.56); (Tuple A)
  - (int 7, String "Hello", char 'c', float 4.76) (Tuple B)
  - (int 5, float 3.14, String "Hello") (Tuple C)
- Data is stored without having necessarily a name
- Data is retrieved by asking for specific characteristics of its content, as in DBMS access
  - Tuple = Read(int 5, String null, char 'c', float null);
  - This request "match" Tuple A (corresponding structure and corresponding content), not march Tuple B (non corresponding content) and not Tuple B (non corresponding structure)
- **Advantage**: more dynamic and adaptive, requires only knowing the structure of data
- **Disadvantage**: more complex to implement
- Sometimes, many data items "match" a request, for which:
  - Either one data item is get at random
  - Or collective operations retrieving all data items can be provided

29

DataSpace Models vs. Event-based

- Advantages of DataSpace models
  - Space and Time uncoupling, as in event-based models → Although some models may require explicit synchronization to access to data
  - Nice representation of common context for interactions, which event-based models miss
  - Physical contextual information can be stored in the data space, as a useful virtual reflection of a real-world environment
- Disadvantages
  - Components may have to actively interrogate ("poll") the dataspace to understand what's happening
  - Relevant events or data may be lost
- Solution
  - DataSpace services can provide for event notification
  - Whenever some new data enters the dataspace
  - Whenever some component access the dataspace
  - In that way. Components can be made aware of what's happening
  - The DataSpace service becomes also an event service

30

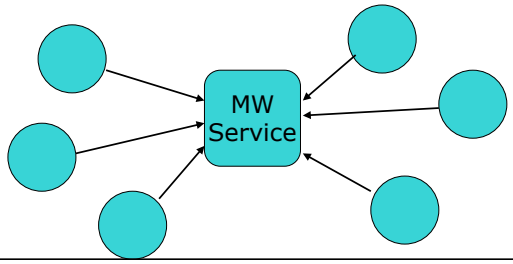
Implementation Models

- All the analyzed middleware services (discovery service, event dispatcher, dataspace) can be implemented in different ways
  - Centralized
  - Locally Distributed
  - Distributed
  - Totally Distributed (Peer-to-Peer)
  - With different types of “task partitioning” in case of distributed implementations
- Where each solution has advantages and disadvantages

31

Centralized Implementation

- A Single component on a single node to implement the middleware service
  - All components access to it for any requests
    - All RMI names are registered there OR
    - All events are dispatched by it and all subscriptions managed by it
    - All data is stored and accessed by it
- E.g., The DHCP service of the university



32

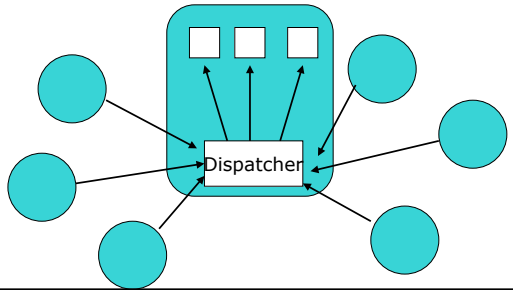
Centralized Implementation: Pros and Cons

- Advantages
  - Very simple implementation
  - No problems of “consistency”, there is a single version of the “world” situation
- Disadvantages
  - Not scalable to many large-size systems
    - Computational bottleneck
    - Communication bottleneck
    - Memory bottleneck
  - Do not exploit locality
    - The service may be far away from the component that needs it
  - Single point of failure

33

Locally Distributed Implementation

- The service is still a single one, but it is implemented on a “cluster” of local computers
  - Each in charge of providing the same service
  - With a “dispatcher” component that act as access point and forward the request to the one of the nodes in the cluster
- Examples: the Google cluster (15.000 Workstations), the Italian Vodafone cluster (for mobile phone bill accounting)



34

Locally Distributed Implementation: Mechanisms and Policies

- Mechanisms
  - Each node is able to autonomously executed the service
  - Typically, they all can access the same data, i.e., have a uniform view of the world (i.e., of the objects, events, data)
  - The Dispatcher receives request for services and "command" one node to provide the service
  - Then, the commanded node interact directly with the client
- Policies
  - How the dispatcher select a node? Typically, with the aim of load balancing
  - Randomly OR Cyclically: high-probability to have uniform load distribution if all requests are similar
  - To Less Loaded node: takes into account how many services each node is currently serving, and forward the service to the less loaded node

35

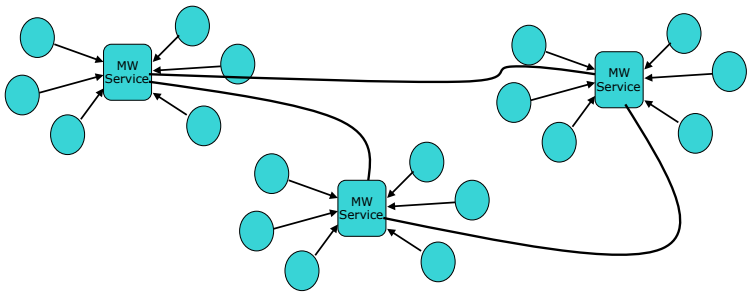
Locally Distributed Implementation: Pros and Cons

- Advantages
  - Very effective for intensively accessed services → high computational load
  - Very effective for management (all in a single room)
  - Very effective to protect data
- Disadvantages
  - The Dispatcher is a Communication bottleneck
  - The Dispatcher is a Single Point of Failure (but there are solutions to deal with this)
  - Do not exploit locality in accesses

36

Distributed Implementation

- A set of services centers
  - Distributed across different sites
  - Coordinating with each other
  - Each capable of servicing requests
- E.g. The DNS System, The Usenet News



37

Distributed Implementation: Pros and Cons

- Advantages
  - Scalability
  - Not computation or communication bottleneck
  - Locality in access to services
  - No single point of failure
- Disadvantages
  - More complex to implement
  - Problems of mechanisms and policies required to coordinate the distributed service centers

38

Distributed Implementation: Mechanisms and Policies

- Mechanisms
  - The various service centers take care of a specific portion of the work
  - They coordinate with each other (exchange data, information, and synchronize) to ensure that they share a common vision of the world
  - They can forward a request to other service centers, or they can cooperatively fulfill requests
- Policies
  - According to which strategy they can partition the work?

39

Distributed Implementation: Policies (1)

- Global replication – Local Service
  - All data and info is replicated to all service centers
  - i.e., all RMI services registers all events, all tuples
  - You can then request a service to any service center, and it will be fulfill it autonomously
- Advantages
  - Requests can be effectively served
  - Very resilient to faults
  - Exploit locality in requests very well
- Disadvantage
  - Very high costs for global replication
  - High communication costs for preserving consistency among the various distributed replicas
  - Consider for example when a client changes a variable in the dataspace….such change on a single replica must consistently reflect everywhere…

40

Distributed Implementation: Policies (2)

- No Replication – Global Forwarding of Requests
  - Data is stored on a single copy on a single node
    - This is typically the node where such data/event/RMIservice was produced
    - But there could be other choices, e.g., group relative data/event/services together on a node based on its characteristics or content (Cfr. Distributed Hash Tables)
  - e.g., the name of an RMU service or an occurred event
  - When you do a request to a service center, it is forwarded to all service centers. The one which has the data necessary to fulfill the request, will then contact directly the client
- Advantages
  - No problems of replication consistency
  - Low costs for maintaining a single vision of the world
  - Exploits locality very well : if multiple answer to a request are possible (e.g., there are several RMI services matching the requested attributes, the most "local" one is typically obtained)
- Disadvantage
  - Requests may take a long time to be fulfilled
  - There are points of failure, that however affect only a portion of the requests

41

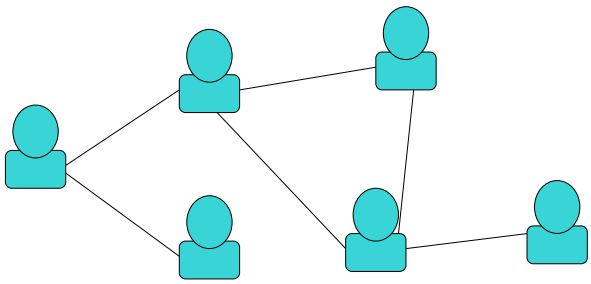
Distributed Implementation: Policies (3)

- Mixed Policies
  - Do some partial replication of data (not on all service centers)
  - Forward requests to a partial sub-set of service centers
  - Ensuring that any request will be forwarded to at least one node containing a replica of a specific data
- In general
  - Such a solution tries to get the advantages of both previous solutions
  - Minimizing the advantages of both
- It is a matter of "**goals**" to be reached to decide what is the best policy to adopt for a specific middleware service

42

Fully Distributed Implementation

- At the extreme, one can imagine that each component that wish to exploit middleware services
  - Volunteer itself to also act as “service center”
  - Taking care of some portion of the data and of some portion of requests
  - In coordination with the other service centers
- This is “Peer-to-Peer”
- E.g., Kazaa, Gnutella, etc.



43

Fully Distributed Implementation

- Policies
  - Today, typically based on a “no replication, full forwarding of requests” policy
  - But replication indeed is promoted!!
- Advantages
  - Really open and decentralized
  - No single point of failure
- Disadvantages
  - Dramatic communication overload due to requests
  - Very complex inter-connection network for “peers”
- Requires specific solutions
  - To be analyzed separately

44

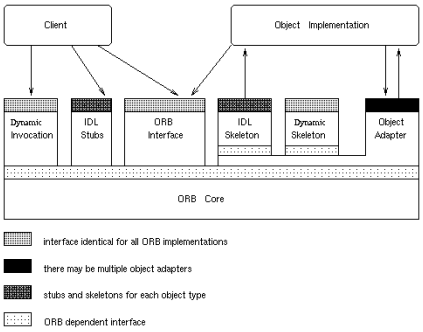
Overview of Existing Middleware

- CORBA
- J2EE
- .NET

45

CORBA

- One of the first working middleware
- Goal: supporting distribute and heterogeneous object-based applications
- Not service-oriented
- Based on a Distributed Object Model
  - Object and services advertise themselvev
  - Client request CORBA (i.e. the so called ORB Object Request Broker) which services are available
  - The ORB receives service invocations and forwards them to the interest objec "implementations"
- The ORB is the basic service of the middleware



46

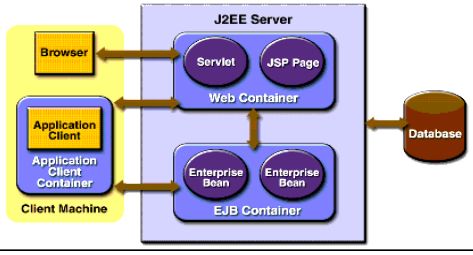
CORBA Components

- ORB
  - Is the basic engines
  - Takes care of services naming (also attribute-based naming) for objects and their services
  - Handles proxies and stubs
  - Receives “discovery” requests
  - Forwards service requests to objects and “translate” requests in case of heterogeneous objects
  - In some cases, replication services
- IDL: Interface Definition Language
  - The standard language with which objects advertise to ORB their public (invokable) interfaces
  - Enables heterogeneous interactions
- Event channel
  - A component of the ORB that handles events and subscriptions
- IIPO
  - An ORB is typically executing over a LAN
  - IIPO is a protocol (over IP) to enable different ORB to interact with each other and to define Inter-ORB systems, that is, a CORBA system which works in a distributed implementation
  - Local replication, global forwarding of requests

47

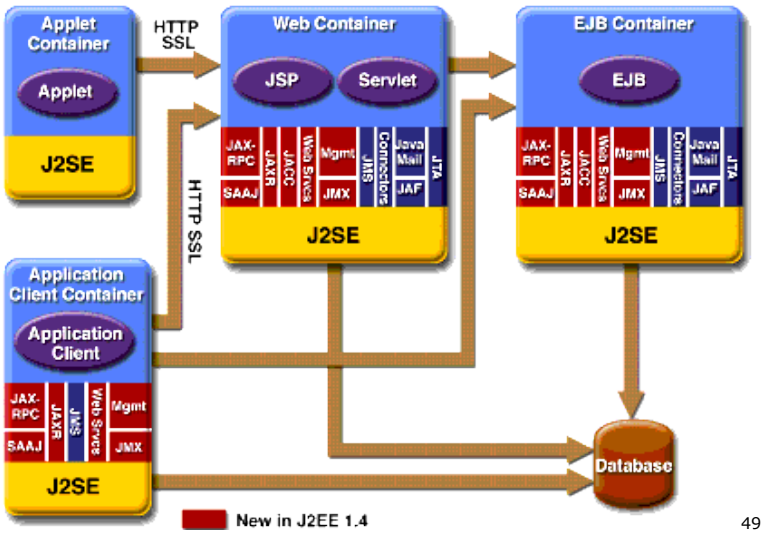
J2EE

- Based on Java Technology and Web-based
  - All Java features
  - Web-based middleware features (Servlet, JSP, Servlet Context)
  - Enterprise JavaBeans (shared self-contained objects)
- Plus a number of “distributed middleware” features
- Goal: supporting distributed Java Web-based applications
  - Also support for service-oriented architectures



48

J2EE Architecture



49

J2EE Components

- Web-based (JPS, Servlet, JavaBeans)
  - XML classes
  - Specific classes to handle and manipulate XML files
  - Various classes to manipulate Web-based graphical interfaces
- SOAP Interface Components
  - To publish services according to the SOAP standard (which includes XML descriptions of services, attributes of services, possible code attachments)
  - An Emerging standard for Web-based distributed object applications
- Plus:
  - An XML HTTP-based Messaging Service
  - JavaSpaces as a Shared DataSpace service
  - JINI, as an attribute-based “Service Discovery” service
  - Security services

50

Microsoft .NET

- Goals similar to that of J2EE
  - And support for service-oriented architectures
- But relying on proprietary product rather than on free ones
- Examples
  - C# vs. Java
  - ASP vs. JSP
  - ActiveX vs. JavaBeans
  - VisualBasic vs. JavaScript
  - XML is XML (it's a standard!!)
  - SOAP is SOAP (it's a standard!!)
- However, if you know what Java Technologies are, you will not problems at all in getting .NET at hand….

51

Open Issues in Middleware

- Support for personalized "user-aware" services
  - User profiling
  - Understanding and adapting to users need and context
- Support for Pervasive Computing
  - Integrating in an easy and seamless way sensors, location systems, Tags, etc. (see lecture on pervasive computing technologies)
- Semantic Services
  - Services that enable to understand what is happening in a "cognitive" way
  - E.g., exploiting in a more intense way XML, RDF, Common Ontologies
- Autonomic Services
  - Services that can self-configure, self-repair
  - E.g., exploiting self-inspection and naturally-inspired self-organization
- Multiagent Systems Middleware
  - Supporting the multiagent systems paradigm
  - See lecture on multiagent systems

52

Readings

- T. Eugster et. Al, “The Many Faces of Publish Subscribe”, ACM Computing Surveys, Vol. 35, No. 2, June 2003
- D. Schmidt, R. Schantz, “Middleware for Distributed Systems”, chapter in the book “Evolving the Common Structure for Network Centric Applications, Wiley & Sons (2001)
- K. Geihs, “Middleware Challenges Ahead”, IEEE Computer, Vol. 24, No. 6, June 2001.
- J. Waldo et al., “A Note on Distributed Computing”, Lecture Notes in Computer Science, No. 1222, June 1999.

See Also

The Various Resources on Middleware at http://dsonline.computer.org

53