

---

## Distributed Component Technologies & their Software Engineering Implications

Wolfgang Emmerich

Zuhlke Engineering Ltd  
49 Great Cumberland Place  
London W1H 7TH, UK  
wje@zuhlke.com

Dept. of Computer Science  
University College London  
Gower St, London WC1E 6BT, UK  
w.emmerich@cs.ucl.ac.uk

- 
- ◆ Distributed component technologies:
    - ◆ COM
    - ◆ CORBA
    - ◆ J2EEare successfully used in industrial practice
  - ◆ What techniques that we have developed can assist in their use?
  - ◆ Do they imply any new research questions that we should be addressing?

- ◆ The Road to Distributed Components
- ◆ Use of Distributed Components in Industrial Practice
  - ◆ Layered N-Tier Architectures
  - ◆ Model Driven Architecture
- ◆ SE Research supporting Use of Distributed Components
  - ◆ Reasoning about Component-based Architectures
    - Static Properties
    - QoS
  - ◆ Component Deployment
- ◆ Future Research Agenda
- ◆ Summary and Conclusions

## The Road to Distributed Components

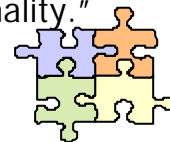
- ◆ Pre-Java:
  - ◆ limited support for distributed computing
  - ◆ violation of information hiding
- ◆ Programming required to combine existing classes
- ◆ Difficulties to combine classes in different programming languages or even different implementations of the same language
- ◆ Object-orientation only delivered reuse on a small scale (reuse foundation classes)
- ◆ Idea: Devise component models that address the problems by grouping related classes into components with language independent interfaces.

"A *component* denotes a self-contained entity (black-box) that exports functionality to its environment and may also import functionality from its environment using well-defined and open interfaces.

In this context, an interface defines the syntax and semantics of the functionality it comprises.

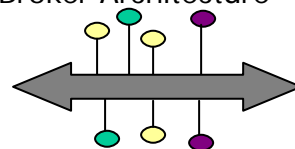
Components may support their integration into the surrounding environment by providing mechanics such as introspection or configuration functionality."

(Michael Stal, Concepts&Tools 19(1998)).



- ◆ How can we support application assembly (e.g. GUIs)?
- ◆ Local component models that support
  - ◆ Interface Definition
  - ◆ Visual Configuration and Assembly
  - ◆ Introspection / Reflection
  - ◆ Customisation through scripting
- ◆ Some influence from ADLs? (Darwin, Wright, Rapide,...)
- ◆ Examples:
  - ◆ Microsoft Object Linking and Embedding – OLE (1992)
  - ◆ Common Object Model – COM (1995)
  - ◆ Java Beans (1997)

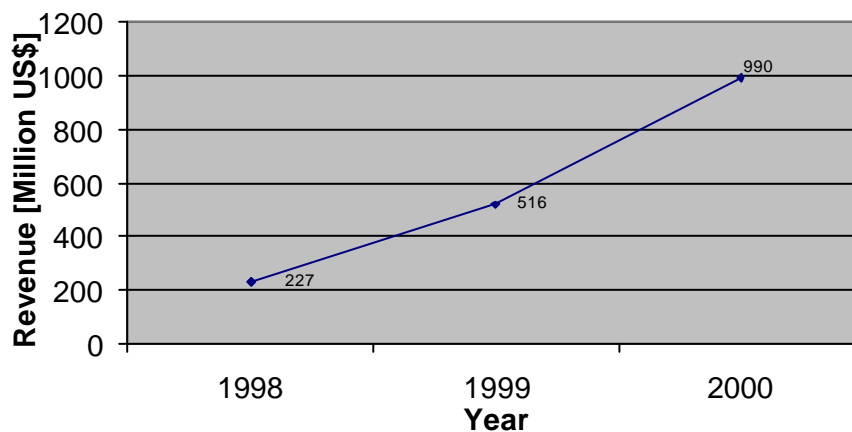
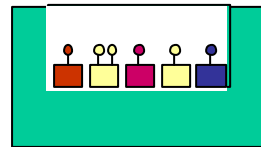
- ◆ Main programming languages do not support distributed system construction well
- ◆ Local component models do not support interaction across machine boundaries
- ◆ Programming language heterogeneity
- ◆ Solutions provided by distributed objects
  - ◆ OMG's Common Object Request Broker Architecture – CORBA (1992)
  - ◆ Microsoft's (D)COM
  - ◆ Java Remote Method Invocation



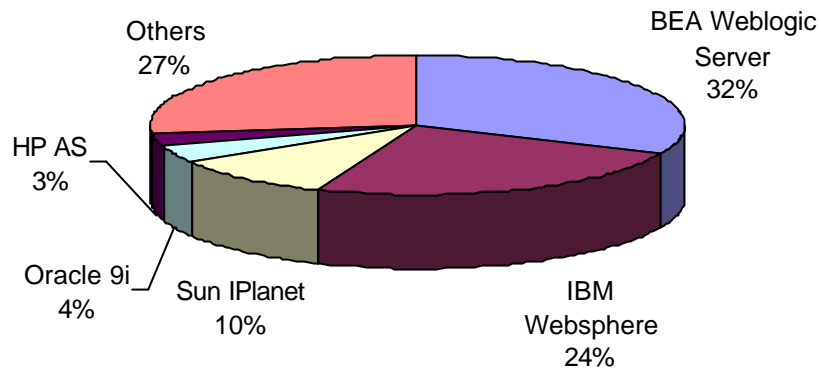
- ◆ Interface Definition Language (IDL)
- ◆ IDL compiler that generates proxies or stubs
  - ◆ Perform marshalling and un-marshalling
  - ◆ Resolve data heterogeneity introduced by platforms
- ◆ Reflection Mechanisms
  - ◆ Interface Repositories / Type Library
  - ◆ Dynamic Invocation Interface / Automation
- ◆ Higher-level Services
  - ◆ Naming, Event communication, Transactions, Persistence, Trading

- ◆ Distributed object programmers need to have distributed systems know-how to achieve
  - ◆ Security
  - ◆ Scalability
  - ◆ Persistence
  - ◆ Failure resilience
- ◆ Leads to:
  - ◆ Distraction from business objectives
  - ◆ Skills shortage
- ◆ Lack of portability of server implementations
- ◆ No standardization / support for deployment

- ◆ Observation: Most of the time, distributed objects were used in particular patterns [Watson, 2002]
- ◆ Distributed components capture these and provide pre-canned support
- ◆ Idea: Build *application servers* that provide containers to control component execution in order to achieve:
  - ◆ Scalability
  - ◆ Security
  - ◆ Persistence
  - ◆ Transactions
- ◆ Examples
  - ◆ Microsoft Transaction Server (1997)
  - ◆ EJB (1998)
  - ◆ CORBA Component Model (2002)



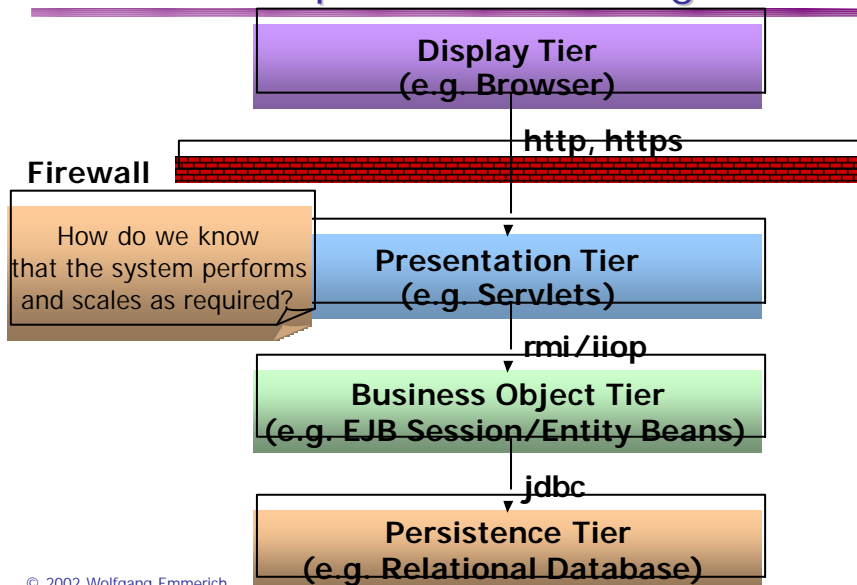
Source: Gartner 2001



Source: Gartner 2001

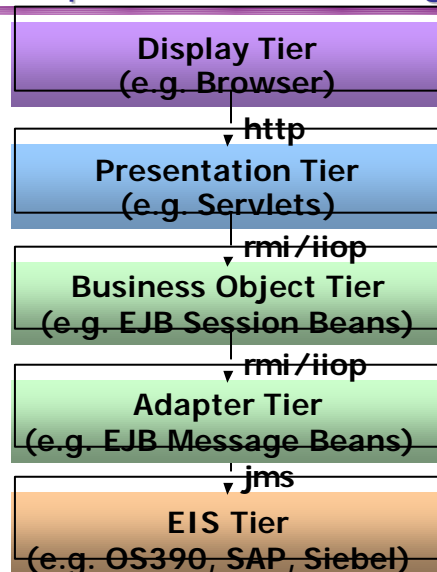
## Use of Distributed Components in Industrial Practice

- ◆ How can we build scalable applications that can be distributed across wide-area networks?
- ◆ Layered architectures (influence from C2?)
  - ◆ Each layer serves particular purpose
  - ◆ Layers implemented in appropriate technologies
- ◆ How can we achieve security?
  - ◆ Separate domains using firewalls
  - ◆ But firewalls block component communication
  - ◆ Use protocols that are typically not obstructed by firewall (e.g. http, https)
- ◆ How can we keep deployment costs low?
  - ◆ User interface in a browser
  - ◆ Problem: Portability of Java Applets
  - ◆ Use dynamic HTML pages generated by server pages

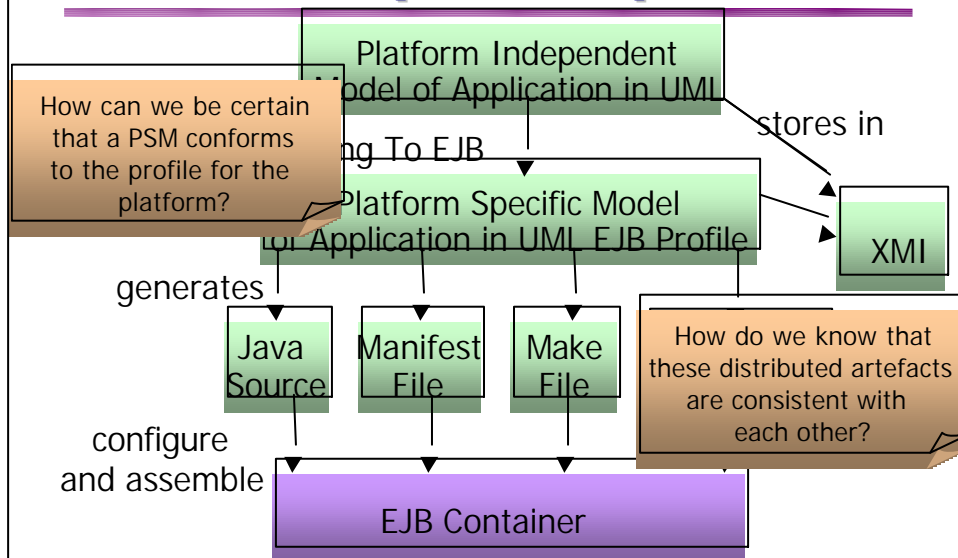


- ◆ Companies can typically not afford to throw away investment into existing systems
  - ◆ Example: Airline reservation
  - ◆ Example: Bank's Global Positioning System
- ◆ How can we build wide-scale distributed applications that interface with legacy applications?
- ◆ Idea: use components to wrap legacy applications [Mowbray 1995]

Distributed Systems are often developed by Distributed Teams!



- ◆ There are (too?) many different distributed component platforms:
  - ◆ Java 2 Enterprise Edition
  - ◆ CORBA Component Model
  - ◆ .NET
- ◆ Uncertainty about which platform will prevail
- ◆ Danger of mixing business logic with platform specific code
- ◆ Leads to non-portable components
- ◆ On server side: Business logic outlive platforms
- ◆ Derive components from platform-independent design models



---

## Software Engineering Research on Use of Distributed Components

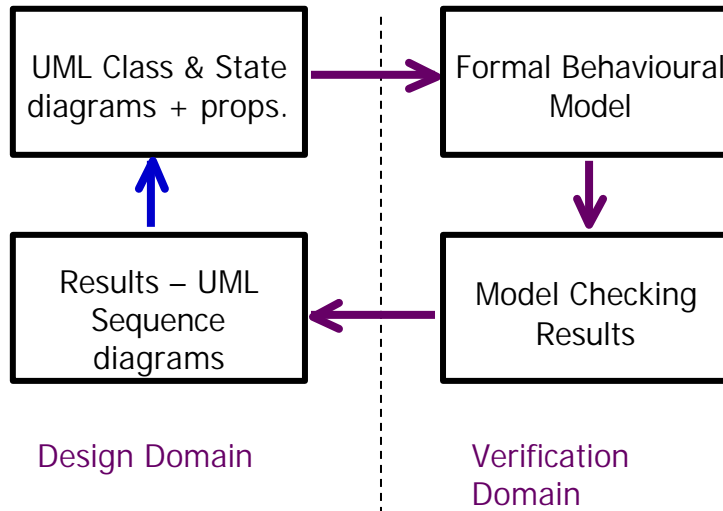
---

- ◆ How meaningful are UML models?
  - ◆ Do they comply with the platform profile?
  - ◆ Consistency with models produced by teams elsewhere?
- ◆ Distributed components execute in parallel
  - ◆ Correctness of behavioural models of distributed components?
    - Deadlock freedom
    - Safety
    - Liveness
  - ◆ Can we quantify their
    - Performance,
    - Scalability / Throughput and
    - Reliability?

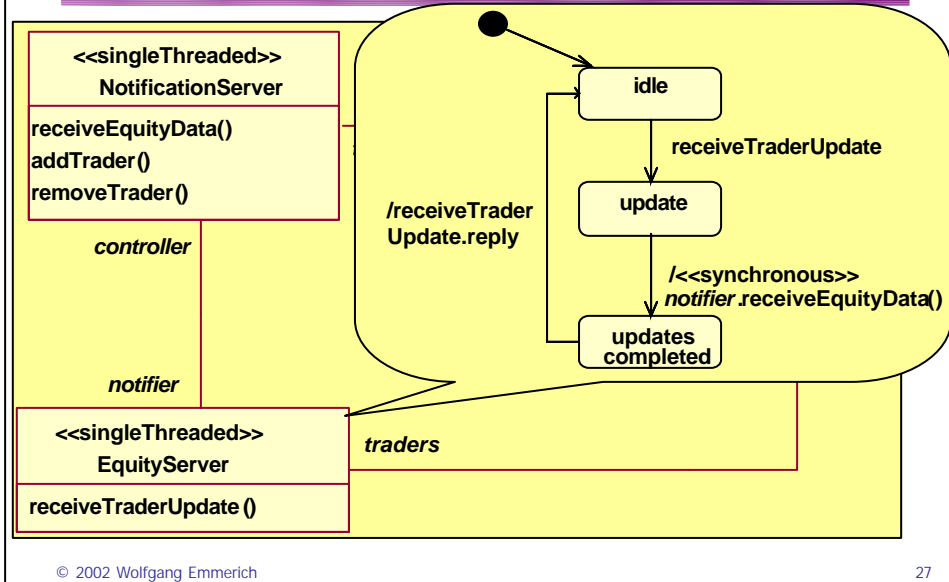
- ◆ Profiles (and UML meta model) specified informally. Precise constraint definition?
- ◆ Most UML CASE tools do not support checks for profile compliance
- ◆ Large projects may use different UML CASE tools at different sites
- ◆ De-couple constraint specification and checks from CASE tools.
  - ◆ Vaziri & Jackson, 1999: Alloy
  - ◆ Nentwich et al, 2001: xlinkit

```

<description>
  Each attribute listed as a 'cmp-field' in the
  deployment descriptor      entity bean must be
  an attribute of the        implementation class
</description>
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>Catalogue</ejb-name> ...
      <ejb-class>CatBean</ejb-class>
      <cmp-field>
        <field-name>OrderNumber</field-name>
      ...
    </cmp-field>
  </entity>
</enterprise-beans>
</ejb-jar>
  
```



- ◆ Translation of UML state diagrams into Promela
  - ◆ [Lilius & Paltor, 1999],
  - ◆ [McUmbler & Cheung, 2001]
 with focus on formal state diagram semantics
- ◆ Behavioural compliance of sequence diagrams with state diagrams [Muccini & Inverardi, 2001]
- ◆ Use of CCS to prove deadlock freedom of DCOM architectures in [Inverardi & Tivoli, 2001]
- ◆ Detecting deadlocks, safety and liveness properties of dist. objects & components in [Kaveh & Emmerich, 2001]



```

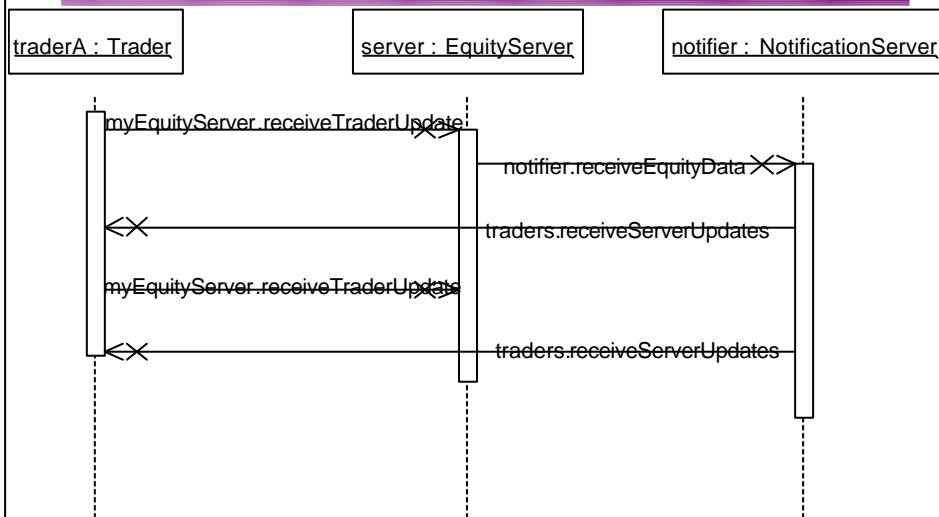
EQUITYSERVER=( startserver->Idle),
Idle=(receivetraderupdate->Update),
Update=(notifier.receiveequitydata->receiveInvocationReply ->Updatescompleted),
Updatescompleted=(reply->Idle).

NOTIFICATIONSERVER=(startnotificationserver->Idle),
Idle=(receiveequitydata->Preparingdata | addtrader->Idle | removetrader->Idle),
Preparingdata=(reply->Sending),
Sending=(traders.receiveserverupdates->receiveInvocationReply ->Sending |
finishedsendout->Idle).

NOTIFIER_OA =(receiveRequest ->relayRequest ->receiveReply ->relayReply ->NOTIFIER_OA).

||TRADING_SYSTEM = (notifier1:NOTIFICATIONSERVER || equityserver1:EQUITYSERVER
|| notificationserverOA:NOTIFIER_OA ) /{
equityserver1.notifier.receiveequitydata / notificationserverOA.receiveRequest,
equityserver1.receiveInvocationReply / notificationserverOA.relayReply,
notifier1.receiveequitydata / notificationserverOA.relayRequest,
notifier1.reply / notificationserverOA.receiveReply }.
    
```

- ◆ Generate a Labelled Transition System from the input process algebra
- ◆ Carry out an exhaustive search in state space of the underlying LTS for action traces leading to property violations
- ◆ In case of violation, produce an action trace
- ◆ Trace is used to construct a UML sequence diagram to show scenario leading to the property violation
- ◆ For liveness property violation the sequence diagram depicts the trace to the terminal set



- ◆ How can we predict the performance/scalability of a distributed component architecture?
- ◆ Use of stochastic process algebras (SPAs) for software architectures [Bernardo et al, 2002]
  - ◆ Compositionality
  - ◆ Quantitative analysis based on mappings to
    - Markov Chains
    - Queuing networksto calculate response times and throughputs
- ◆ SPAs well supported by model checking tools
  - ◆ Tipp [Götz et al, 1993]
  - ◆ PEPA [Hillston, 1994]
  - ◆ Two Towers [Bernardo, 2001]

- ◆ How can we evolve a distributed component-based architecture to improve performance, scalability, fault-tolerance and security of distributed components without changing the source code?
- ◆ Application servers support component deployment using flexible deployment descriptors
- ◆ These descriptors have an important impact on how well the software system meets non-functional requirements!
- ◆ Which methods are there to engineer deployment information?

- ◆ Emerging discipline [Bishop & Herrmann, 2002]
- ◆ For example [Rutherford et al, 2002]:
  - ◆ Bean Automated Reconfiguration framework (BARK)
  - ◆ Defines scripting language for describing actions required for EJB deployment, such as:
    - Installation
    - Activation
    - Deactivation
    - Modification
  - ◆ Provides interpreter with different reliability guarantees for executing scripts (e.g. atomic execution of a script on a number of hosts)

## Research Agenda

- ◆ Components will be increasingly used to implement Web-Services
- ◆ Web services will be hosted and composed across different domains
- ◆ Service level agreements (SLAs) will need to determine quality of service provision and use
- ◆ How do we systematically engineer SLAs?
- ◆ How can containers enforce service level agreements?

- ◆ Components and application servers may be executing services on behalf of competing organizations, e.g. All major European airlines use Amadeus for reservations and ticketing
- ◆ Organizations may render services to competing organizations, e.g. JP Morgan uses BoNY for back-office processing
- ◆ How can we define trust?
- ◆ How can we prove that required trust-levels are achieved by a distributed architecture?

- ◆ How well does a SW architecture accommodate changes in requirements and its environment?
- ◆ Aka self-healing or self-organizing systems
- ◆ Explicit deployment support accelerates component technologies into pole position
- ◆ But deployment support varies among application server
- ◆ How do we select the most appropriate application server?

- ◆ In 2001: Integration of asynchronous messaging with component technologies:
  - ◆ Message Beans in EJB 2.0
  - ◆ Microsoft Message Queue in .NET
- ◆ Used with components that control workflow
  - ◆ MQSeries Workflow in IBM Websphere
  - ◆ Biztalk Server in .NET
- ◆ **Engineers need**
  - ◆ Graphical modelling notations and tools
  - ◆ Simulation and analysis support
  - ◆ Compliance of actual message flow with prescribed workflow
- ◆ Many of the problems faced have been addressed in software process literature
- ◆ Can these solutions be rejuvenated?

- ◆ Various distributed component technologies
  - ◆ Java 2 Enterprise Edition
  - ◆ Microsoft COM, MTS, and .NET
- ◆ Well-supported and adopted in industry
  - ◆ Used in well-understood N-Tier Architectures
  - ◆ Technology independence with OMG's MDA
- ◆ Research results on use of components
  - ◆ Reasoning about models of components
  - ◆ Deployment
- ◆ Many open & worthwhile SE research questions

Slides available from  
<http://www.distributed-objects.com>