

# UML with V

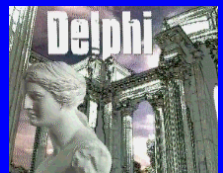
max.kleiner.com

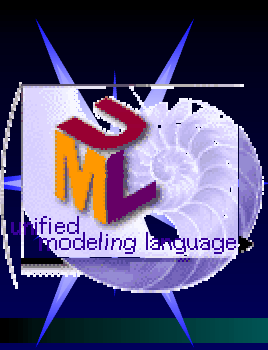
*much communication in a motion  
without conversation or a notion*





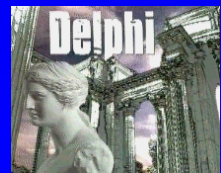
- ☯ The process of defining the requirements is iterative, which means that e.g. if a Use Case is drawn and the according Activity is not what the customer expects, then we go back to the Use Case and correct it.
  - GUI Design and DB Modeling is not part of UML !
  - But along with class diagrams we can deliver mask prototypes to define the GUI with attributes for DB tables
  - UML is a bridge so that the customer and the developer understand the same about what the system is supposed to do. Talk the same language means the same glossary.

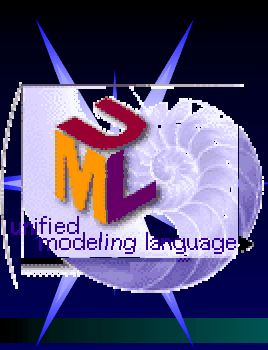




# Open or programming for change

- ☯ The Unified Modeling Language [UML95] is a third-generation object-oriented modeling language for specifying, visualizing, and documenting the artifacts of an object-oriented system under development. It fuses the concepts of different notations.
- ☯ The result is a single, common, and widely usable modeling language for users of these and other methods.
- ☯ A *model* is an abstraction of a modeled system with all diagrams, specifying the modeled system from a certain viewpoint (business or technical) and at a certain level of abstraction.



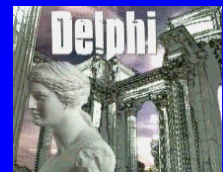


# Summary of the session

- ☯ The diagrams in the sense of a V process
- ☯ The 8 basic diagrams in traceability and notation

---

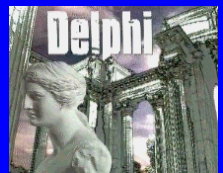
- ☯ Practical use of an UML example „Train Control System Interlock“

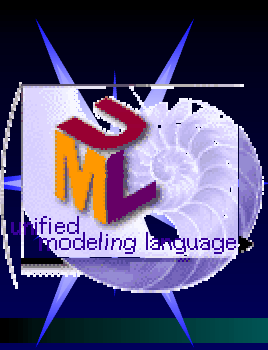




# Who's here?

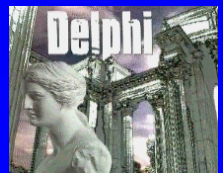
- ☯ Which one has experience of UML?
- ☯ Why is design and documentation so dreadful?
- ☯ What do we know about: OOP is iterative and incremental?
- ☯ Why is RAD (Rapid Application Development) so bad for OOP?
- ☯ Remember: Model->Code->System (MCS)





# Metamodel

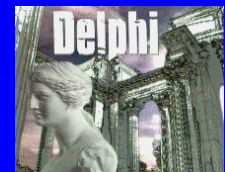
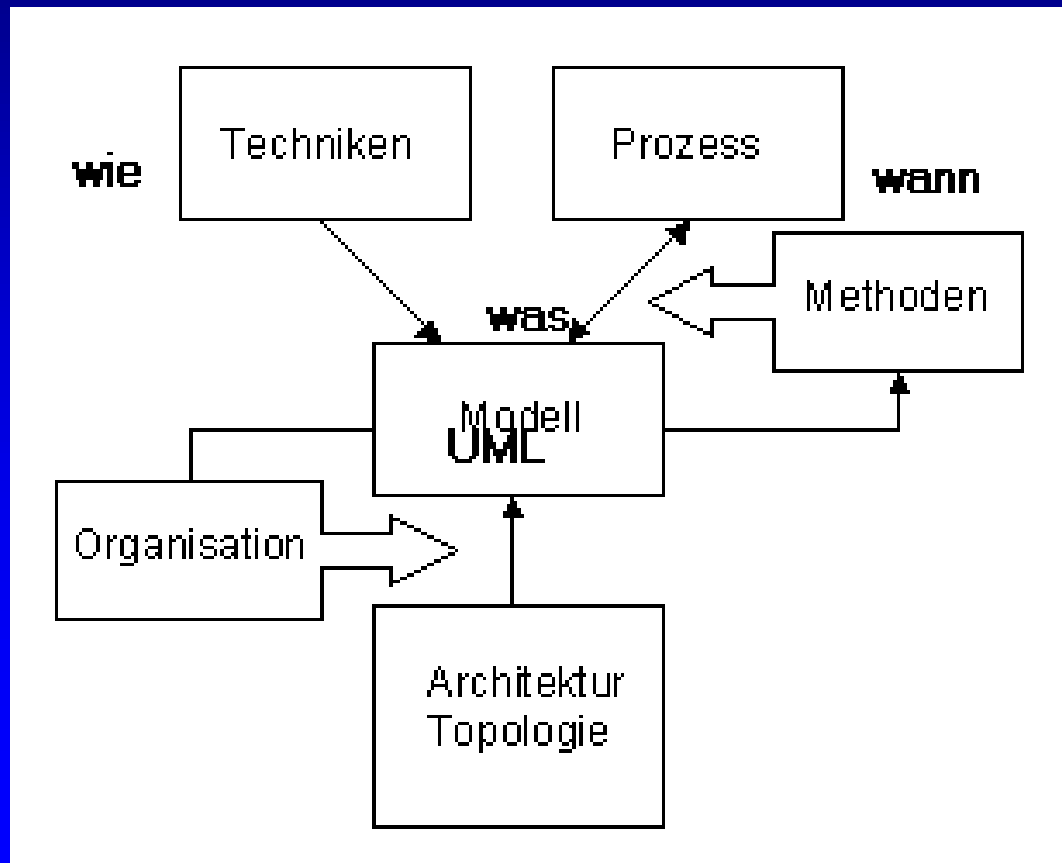
- ☯ The UML metamodel is a logical model and not a physical (or implementation) model. The advantage of a logical metamodel is that it emphasizes declarative semantics, and suppresses implementation details.
- ☯ UML does not prescribe a specific process.
- ☯ UML defines MDA (model driven architecture)

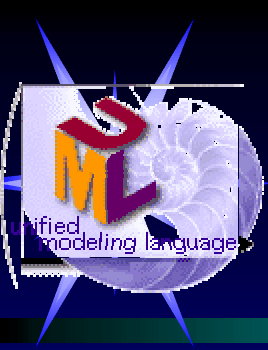




# Process with V-Model

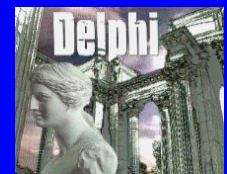
- You can integrate each diagram in a time phase of a process, see paper UML with V or next slide





# Process with V

Phase	Model	Instance
Analysis	Use Case	Requirements
Analysis	Activity	Business Units
Analysis/Design	Class Diagram	Objects and Types
Design	State Event	State Values
Implementation	Sequence	Scenarios
Implementation	Packages	Versions
Integration	Component	Files
Integration	Deployment	Devices





# UML 2.0 V Diagrams

## ☉ Use Case

- Initializing, find actors
- Requirements in context
- Discussion reviews

## ☉ Activity

- Business process
- Paralleled, events
- Workflow...

## ☉ Class Diagram /Object Diagram

- find structure
- OOP-Entities
- Relations
- Utility function...

## ☉ State Event /Protocol Automat

- dyn. Behavior in classes
- Object life cycle
- State transitions

## ☉ Deployment

- Collaboration of components
- Aspects of deployment
- Horizontal System architecture
- Net, protocols and topologies

## ☉ Component /Subsystem /Composition

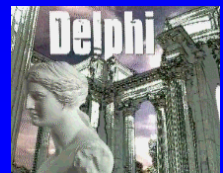
- Autonomic interfaces
- executable groups
- source, binary, executable

## ☉ Packages /Collaboration Patterns

- Impact of changes
- Vertical architecture
- Package =unit as a module
- Libraries, patterns

## ☉ Sequence Diagram /Communication /Interaction Overview /Time Diagram

- Message flow of objects
- Timeline of call stack
- Dynamic call cascades





# Aim of UML: a real model ?!

CNN, 25.09.2001



FUTURES  
S&P ▲ 0.50

WAR AGAINST TERROR  
PUTIN IN GERMANY FOR  
ANTI-TERROR TALKS

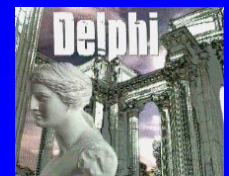


What's  
wrong here?



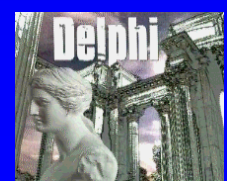
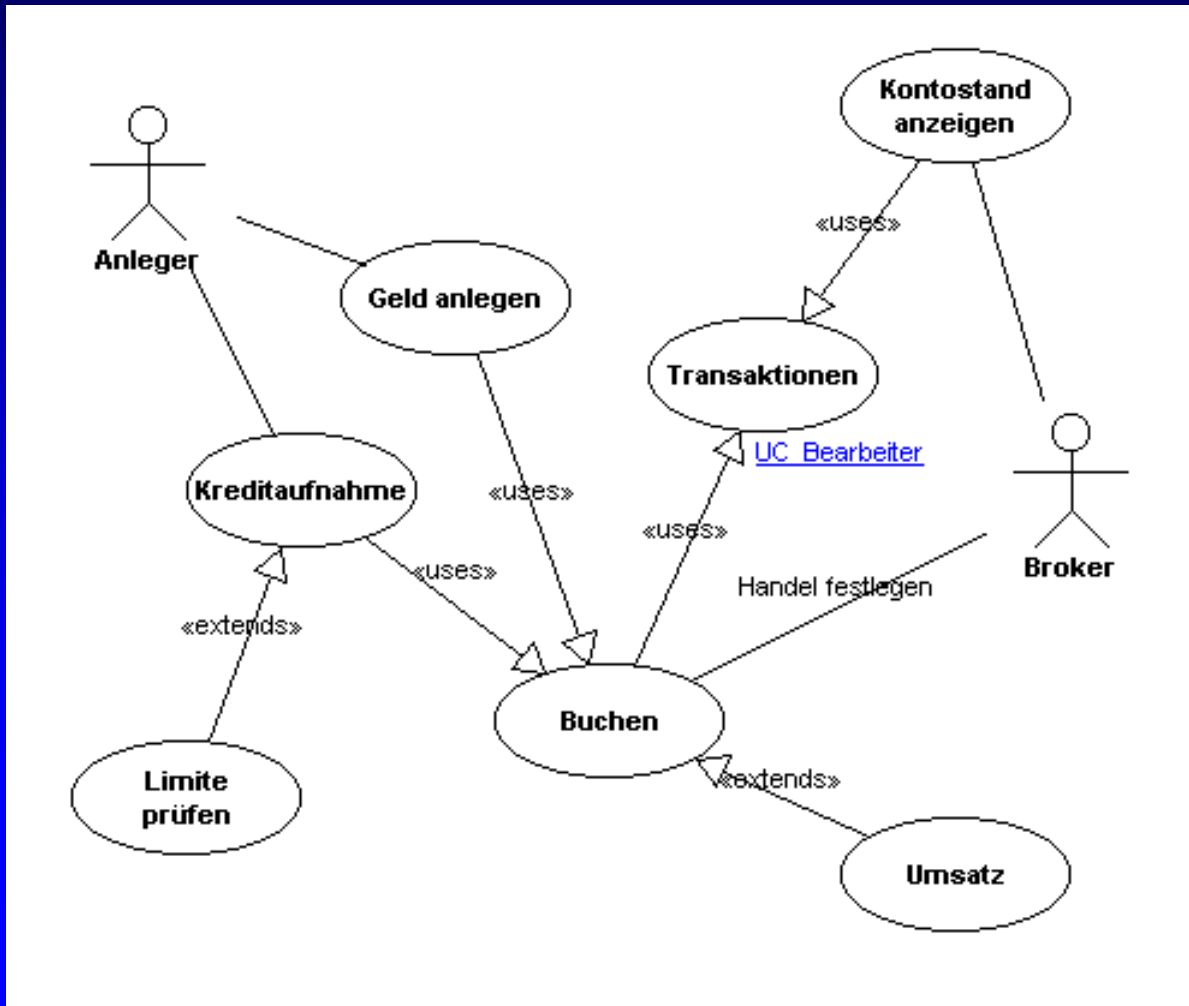
The following model deals with financial business, main actions are managing a portfolio, trading at stock markets and generate balance sheets.

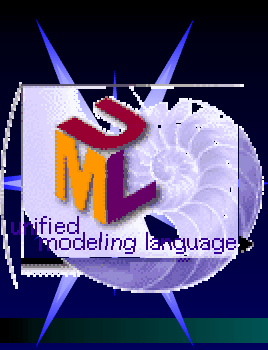
Part II is a real example of a Train Control System





# Use Case

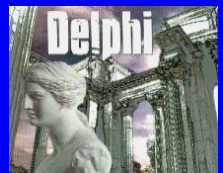


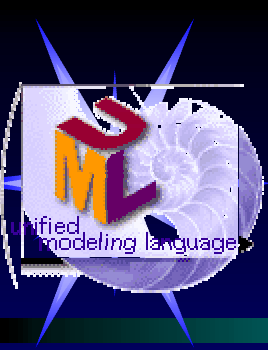


# Use Case

Use cases are used to obtain system requirements from a user's perspective. A use case can be described as a interaction that a user has with a system to achieve a goal.

It is helpful to provide a event driven template (use case script) that can be used to document use cases.





# Ex.: Use Case Script

Use Case: Create an email with no attachment

Reference: 12

Author: Air Max

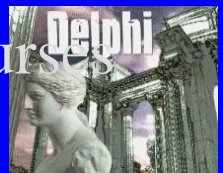
Pre-condition: User has logged into a system (the necessary conditions that have to be met before the use case can be performed)

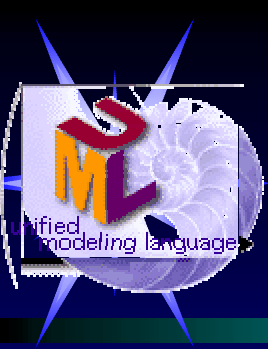
Description: User enters recipient address, subject and text message. Then, user selects to send message.

Post-condition: Mail message is sent (the state of the system after the use case is performed)

Exceptions: Recipient address not entered (different error situations that can occur)

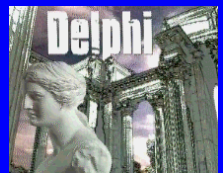
Variations: Create an email with an attachment (alternative courses that can potentially be taken)





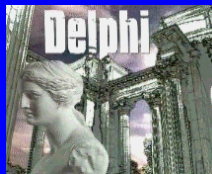
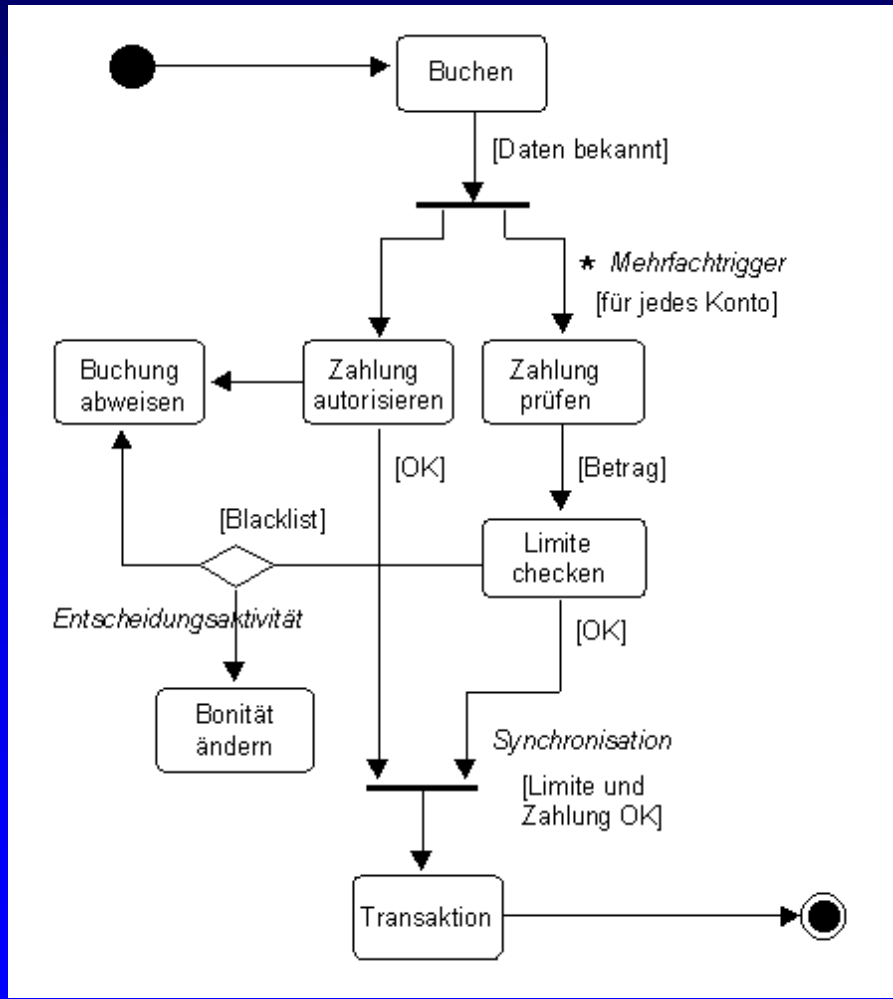
# Use Case Notation

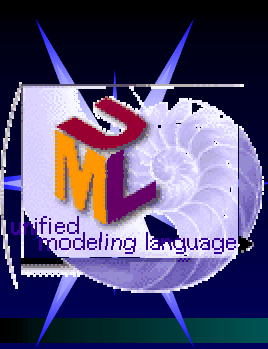
- An actor is a role that a user plays with respect to the system.
- Two types of relationships are used in use case diagrams.
- The <extend> relationship is used when one use case is similar to another use case but does a bit more or to describe optional behavior (e.g., forward a mail message).
- The <include> (uses) relationship occurs when you have common behavior that exists in multiple use cases that can be factored out into a separate use case (e.g., login use case).





# Activity

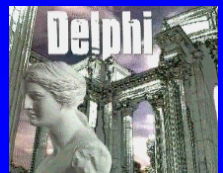




# Activity


Activity diagrams show behavior with control structure in a token manner.

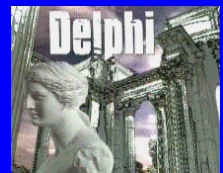
Activity diagrams can be used to show behaviors over many use case, model business workflows, or describe complicated methods in a viewpoint.





# AD Notation

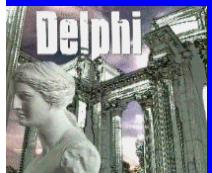
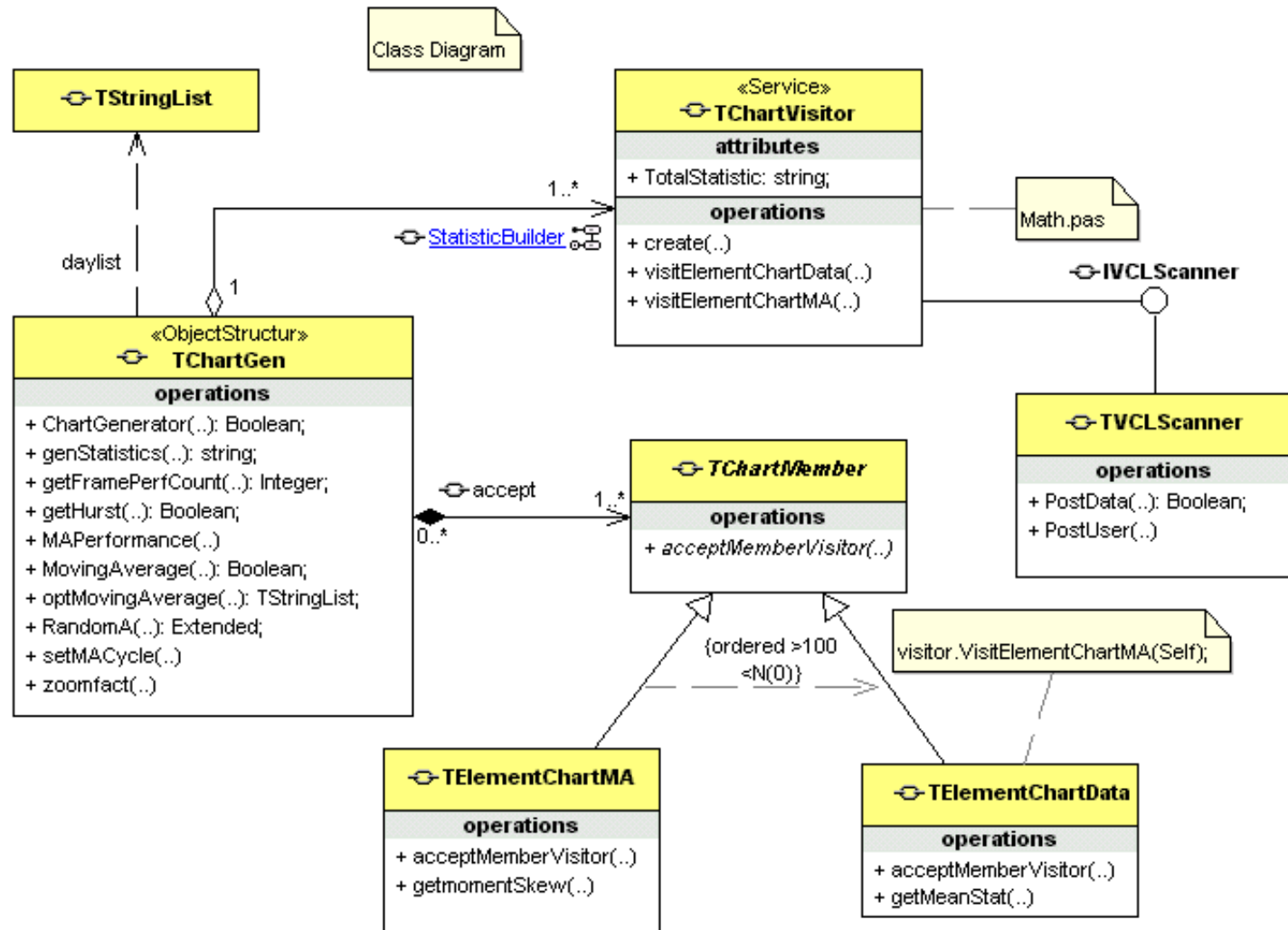
- ☯ Activities in a diagram may or may not correspond to methods.
- ☯ Specific notation found in this type of diagram includes guards which are logical expressions that evaluate to true or false. 
- ☯ A synchronization bar indicates that the outbound trigger occurs only after all inbound triggers have occurred.
- ☯ Swim lanes (using a swimming pool analogy) allow you to vertically partition an activity diagram so that the activities in each lane represent the responsibilities of a particular class or department.

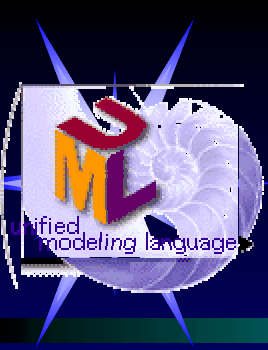






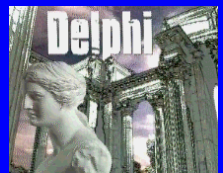
# Class Diagram





# Class Notation Part I

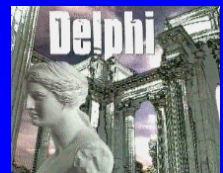
- The access modifier symbols `+`, `-`, and `#` correspond respectively to public, private, and protected access modifiers.
- Realizes relationship from a class to an interface indicates that the class implements the operations specified in the interface.
- Associations represent relationships between instances of classes. Each association has two roles; each role is a direction on the association.

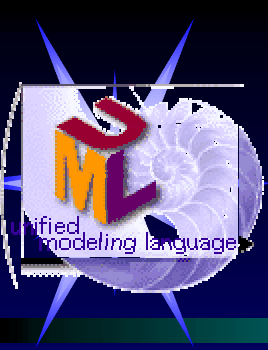




# Class Notation Part II Relationships

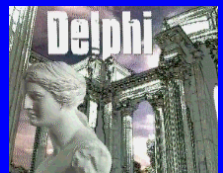
- A role has multiplicity (an indication of how many objects may participate in a given relationship). Examples of multiplicity indicators are: 1 (Exactly one), \* (0 to any positive integer), 1..\* (1 to any positive integer), 0..1 (0 or 1).
- Constraint {} specifies conditions among model elements that must be maintained as true. Constraints are shown using text enclosed in braces.
- Generalization relationship is an association with a small triangle next to the class being inherited from whereas an aggregation relationship is an association with a diamond next to the class representing the aggregate.





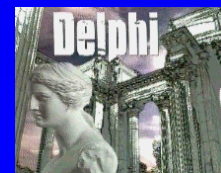
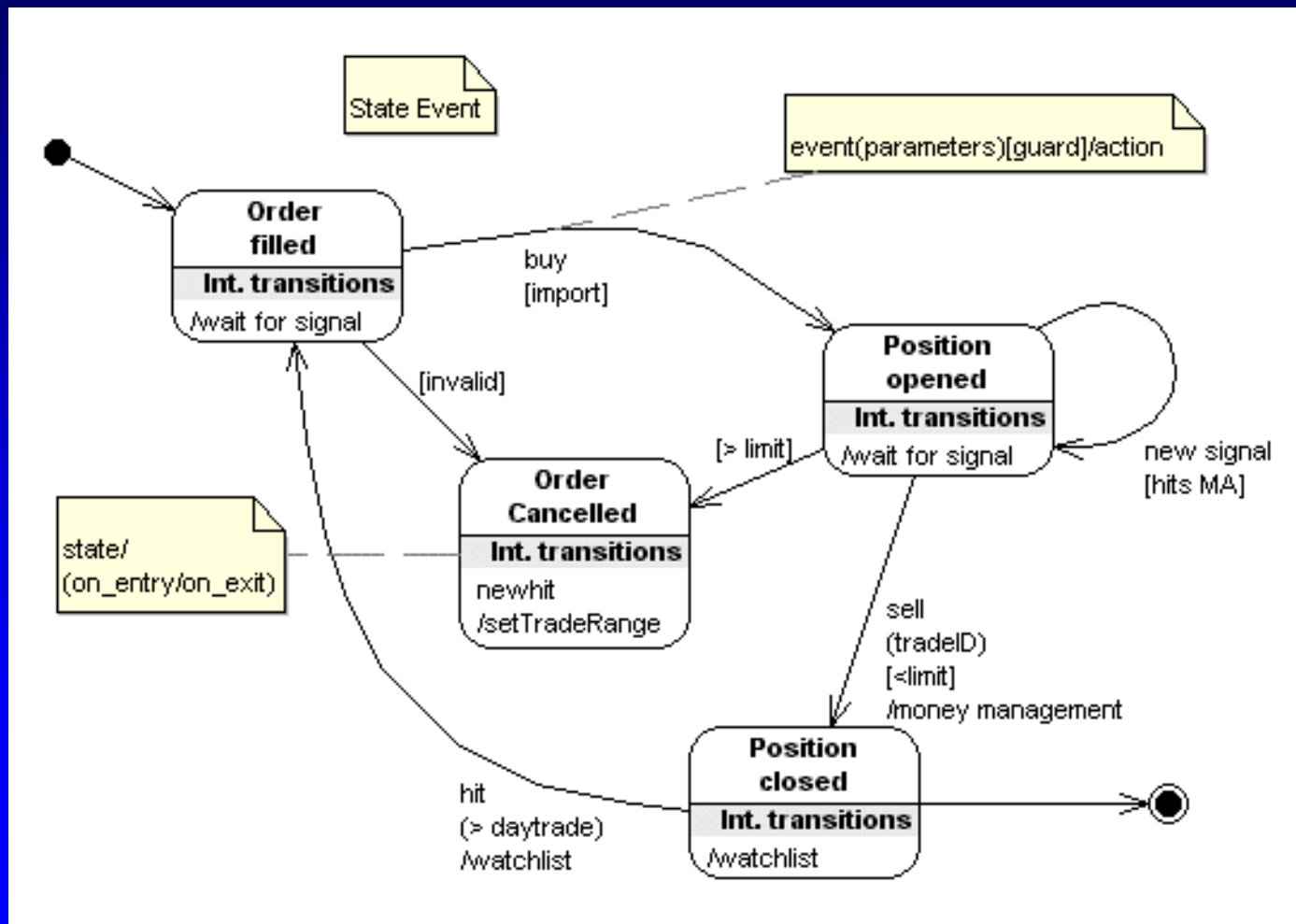
# Classbuilding

- ☯ Class diagrams can be used to document a Business Object Model and are widely used in both high-level and low-level design documents.
- ☯ To find classes you may have 3 possibilities:
  - ◀ Use of «design patterns» saves a lot of time
  - ◀ Analysis of text of the use case script
  - ◀ Team working with CRC-Cards





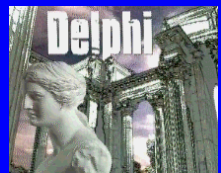
# State Event of a Class





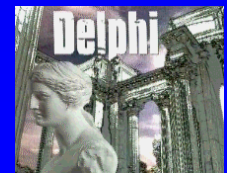
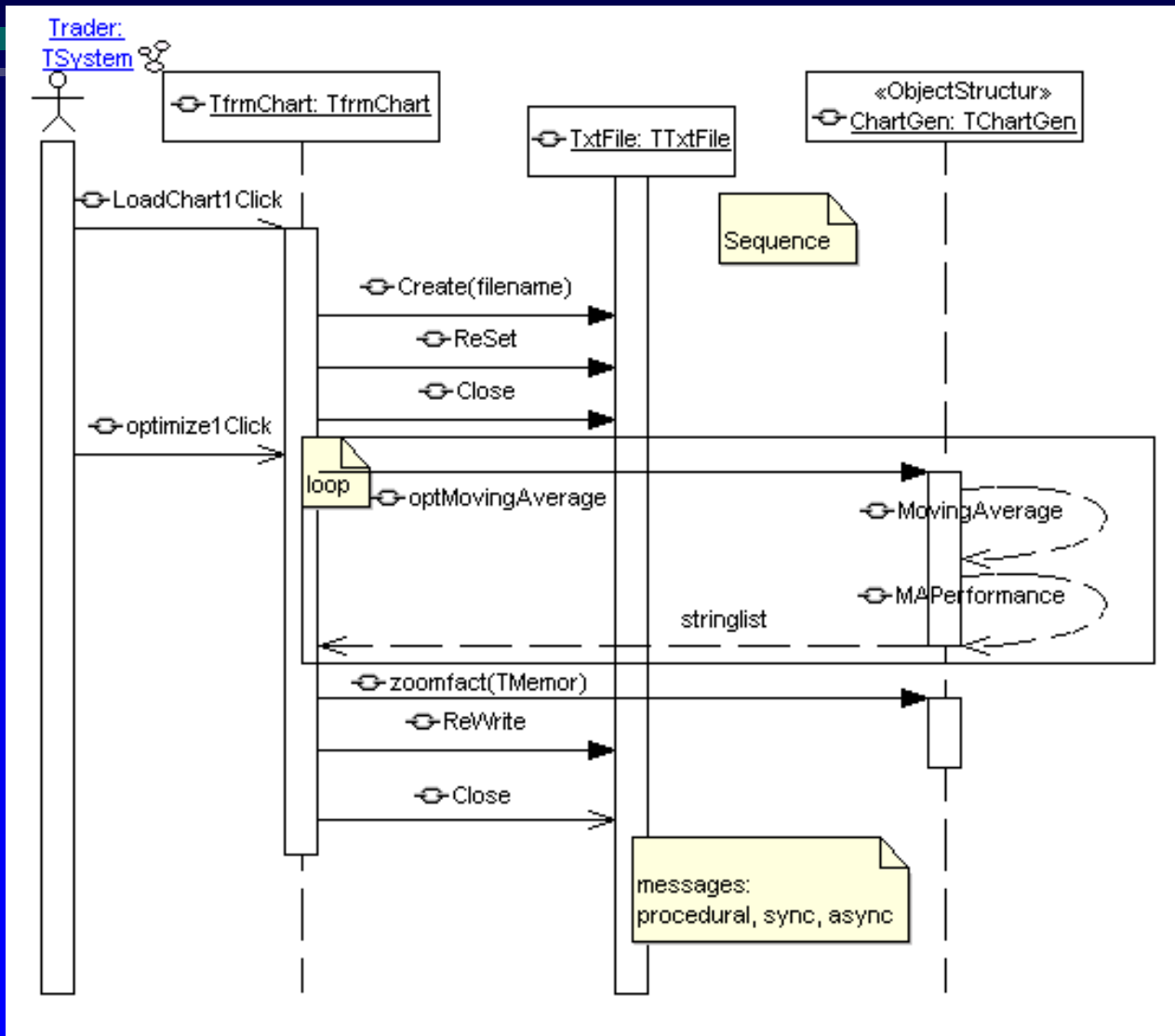
# State Event Notation

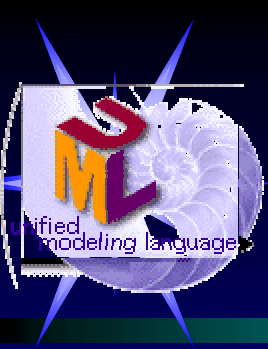
- ☯ State Diagrams explore the state transitions of a single object)
- ☯ The syntax of a <transition> between a state is:  
<event (parameter) [condition] / action
- ☯ UML is an extensible language. For example, stereotypes are one of the mechanisms that can be used to extend the UML. In general, a stereotype represents a usage distinction.
- ☯ Stereotypes can be used with any diagram to extend its meaning.





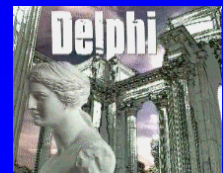
# Sequence Diagram

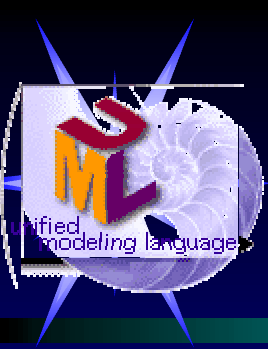




# Sequence Notation

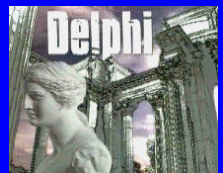
- ② Interaction diagrams show how objects interact with one another. There are two types of interaction diagrams: Collaboration and Sequence Diagram.
- ② Collaboration diagrams can be used to show how objects in a system interact over multiple use cases. Collaborations are helpful during the exploratory phases of a development process (i.e., trying to search for objects and their relationships). Since there is no explicit representation of time in collaborations, the messages are numbered to denote the sending order.





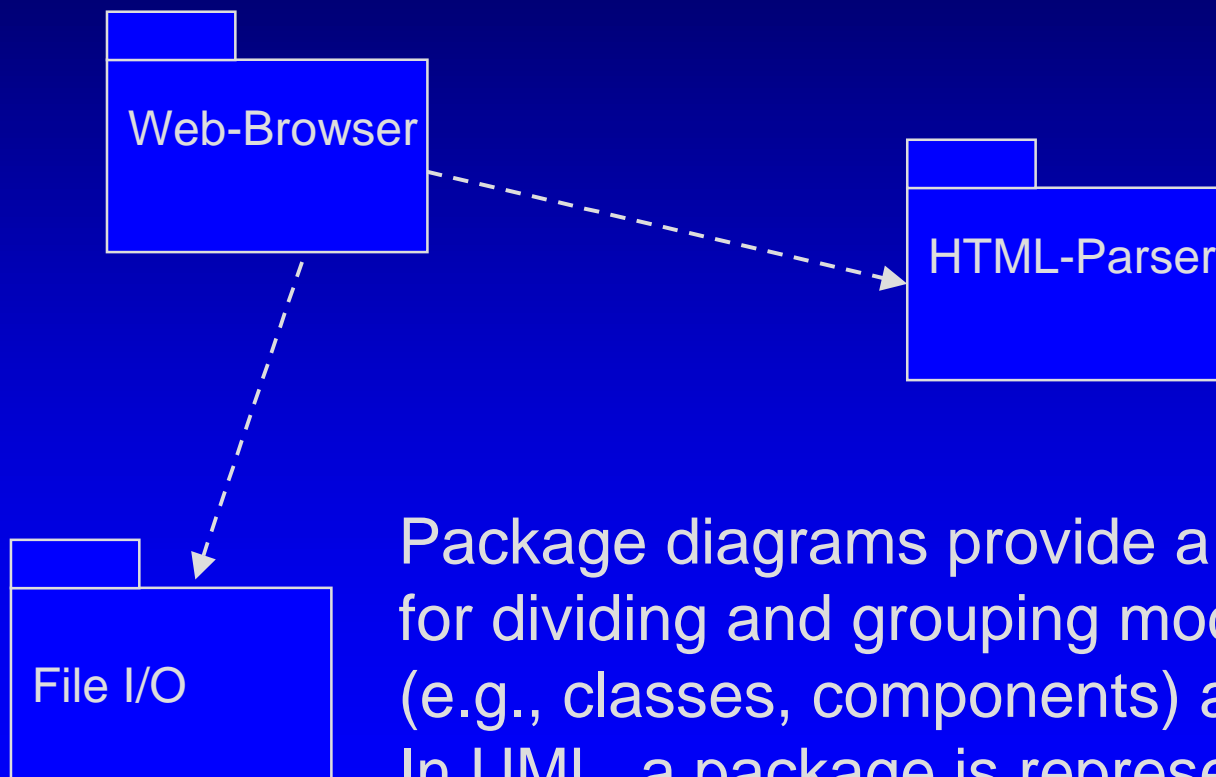
# Sequence Notation

- A Sequence diagram is typically used to show object interactions in a single use case and it is easier to see the order in which activities occur.
- The emphasis of sequence diagrams is on the order of message invocation. The vertical axis of a sequence diagram represents time whereas the horizontal axis represents objects.
- Messages are:
  - synchronic or asynchrony calls
  - iterations  $\langle * \rangle$  and conditions [ ]

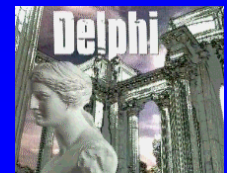




# Packages



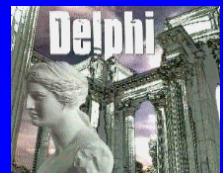
Package diagrams provide a mechanism for dividing and grouping model elements (e.g., classes, components) at design time. In UML, a package is represented as a folder with dependencies.





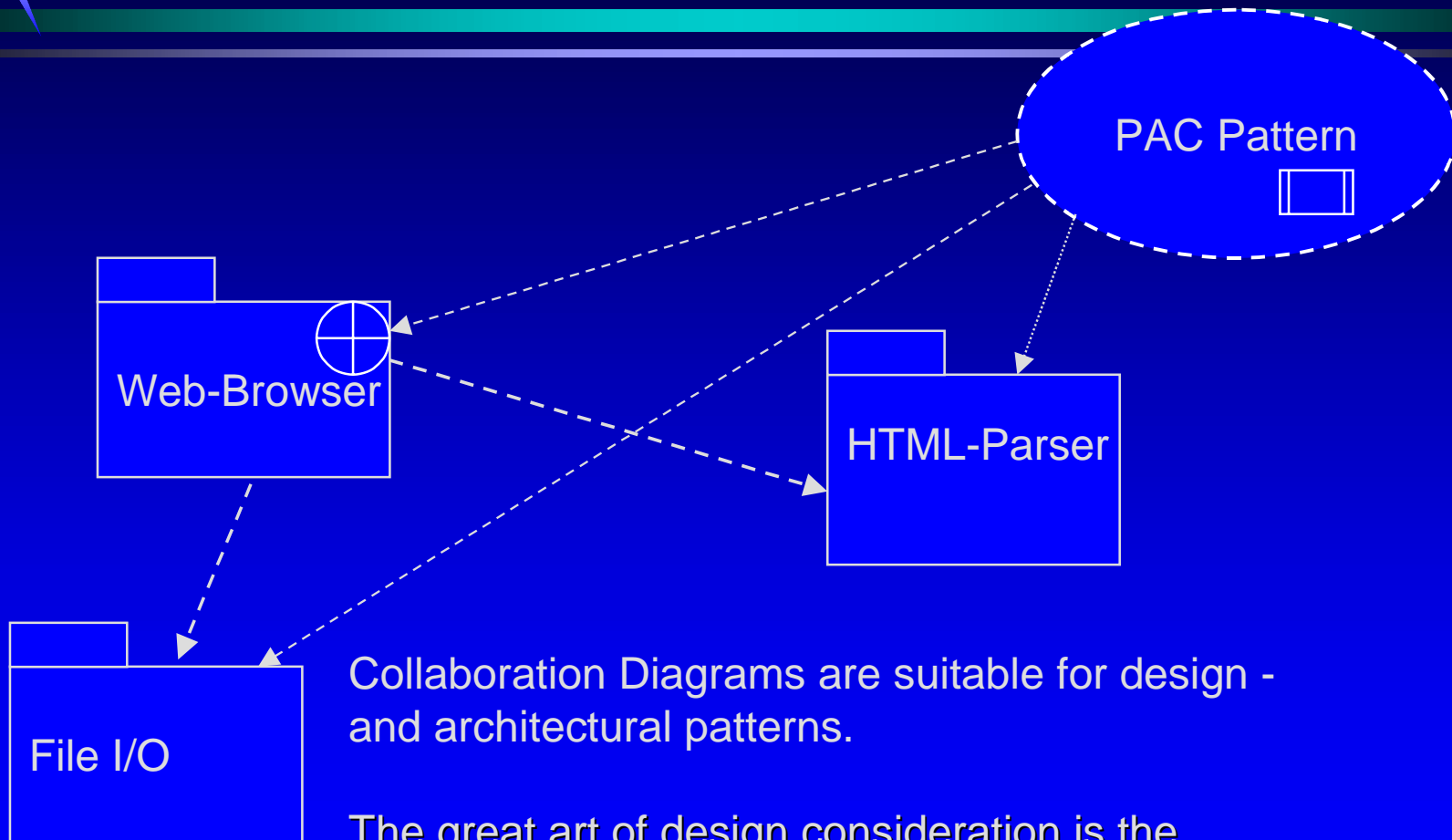
# Packages

- ☯ In effect, a package provides a namespace `//.` or a unit such that two different elements in two different packages can have the same name (qualified).
- ☯ Packages may be nested within other packages (Subsystem). e.g.: `//./system.parser.xml.header`
- ☯ Dependencies between two packages reflect dependencies between any two classes in the packages. For example, if a class in package A uses the services of a class in package B, package A is dependent on package B. An important design consideration is the minimization of dependencies between packages.



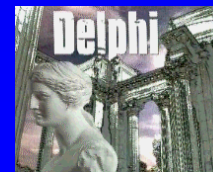


# Packages with Patterns



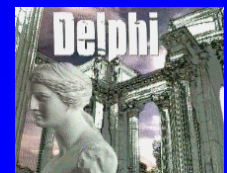
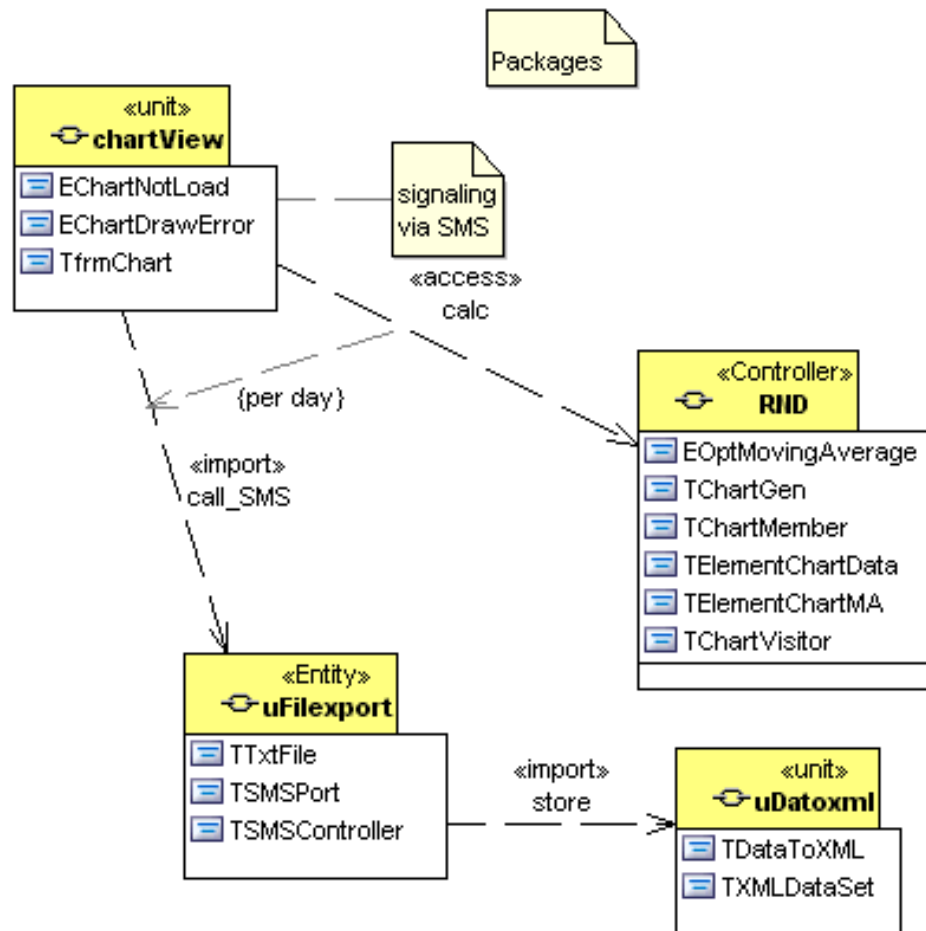
Collaboration Diagrams are suitable for design - and architectural patterns.

The great art of design consideration is the minimization of dependencies between packages which minimizes the impact of changes.





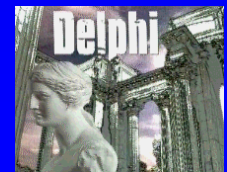
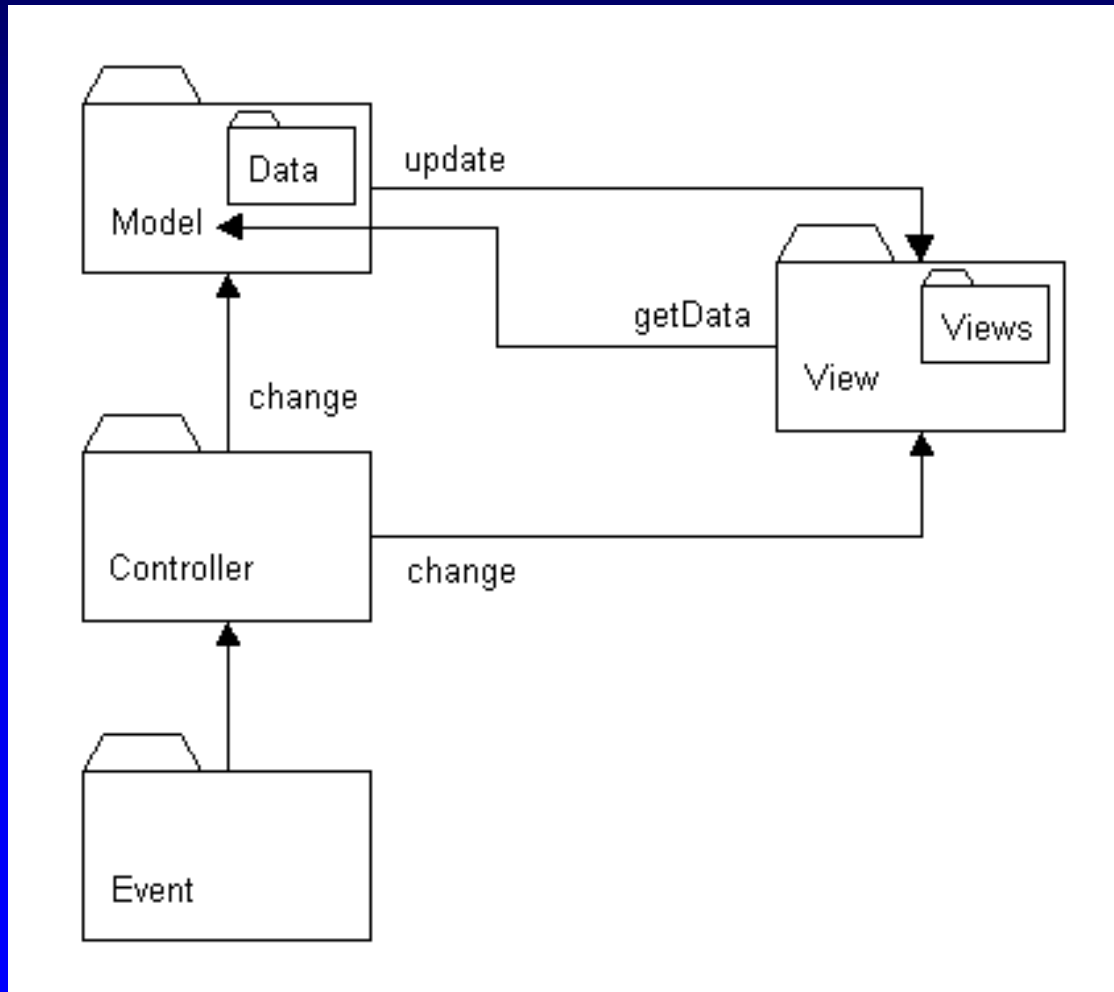
# Packages





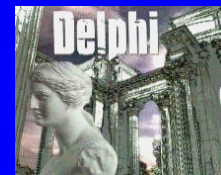
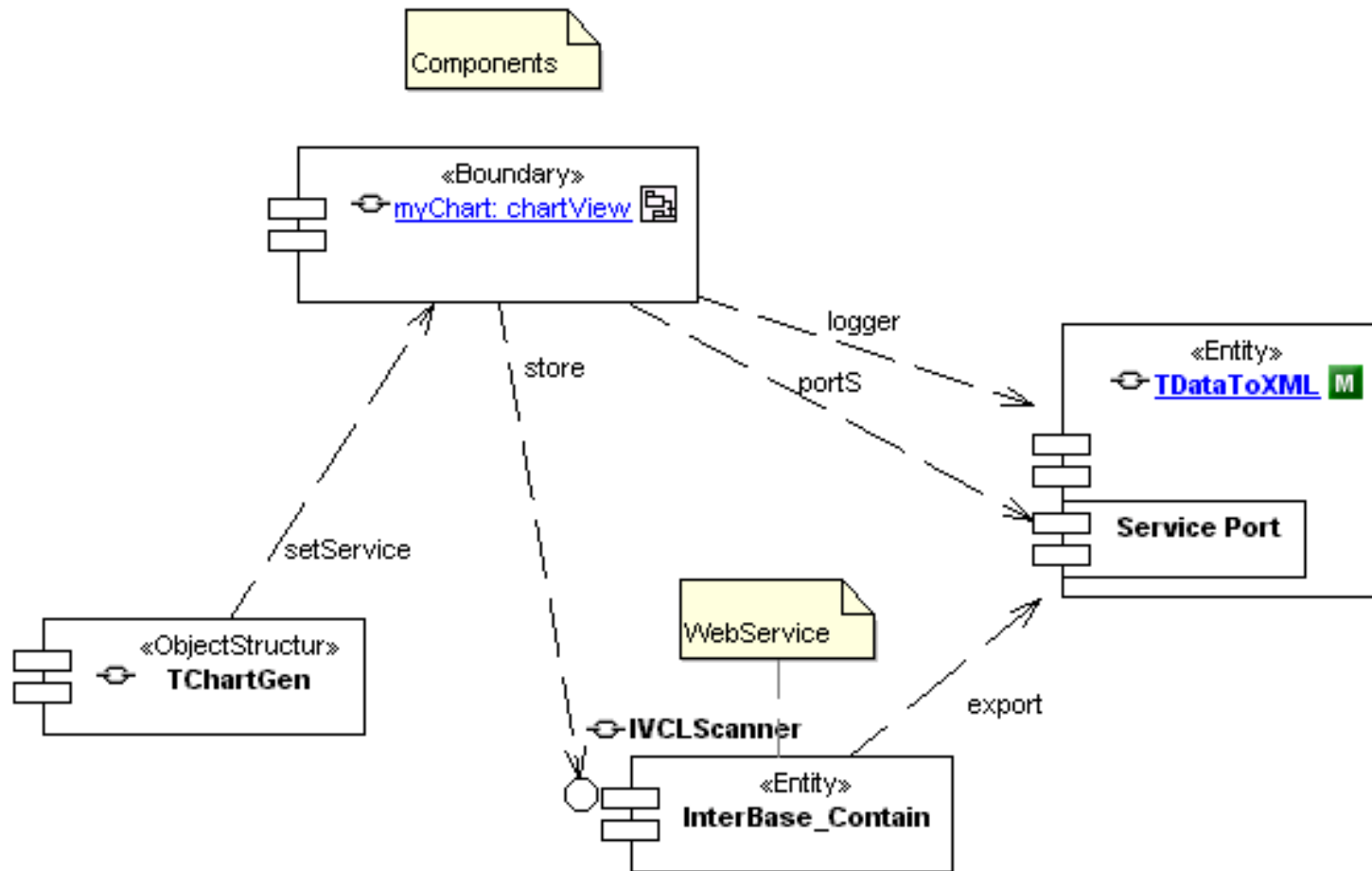
# Subsystem-Diagram

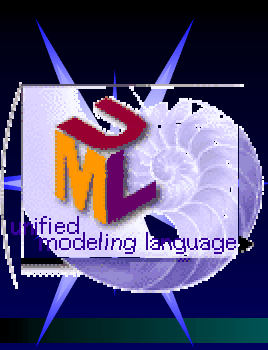
Architectural combinations and build patterns of packages





# Component

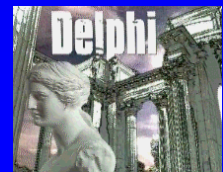




# Component

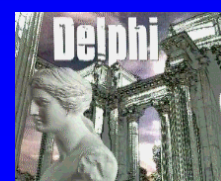
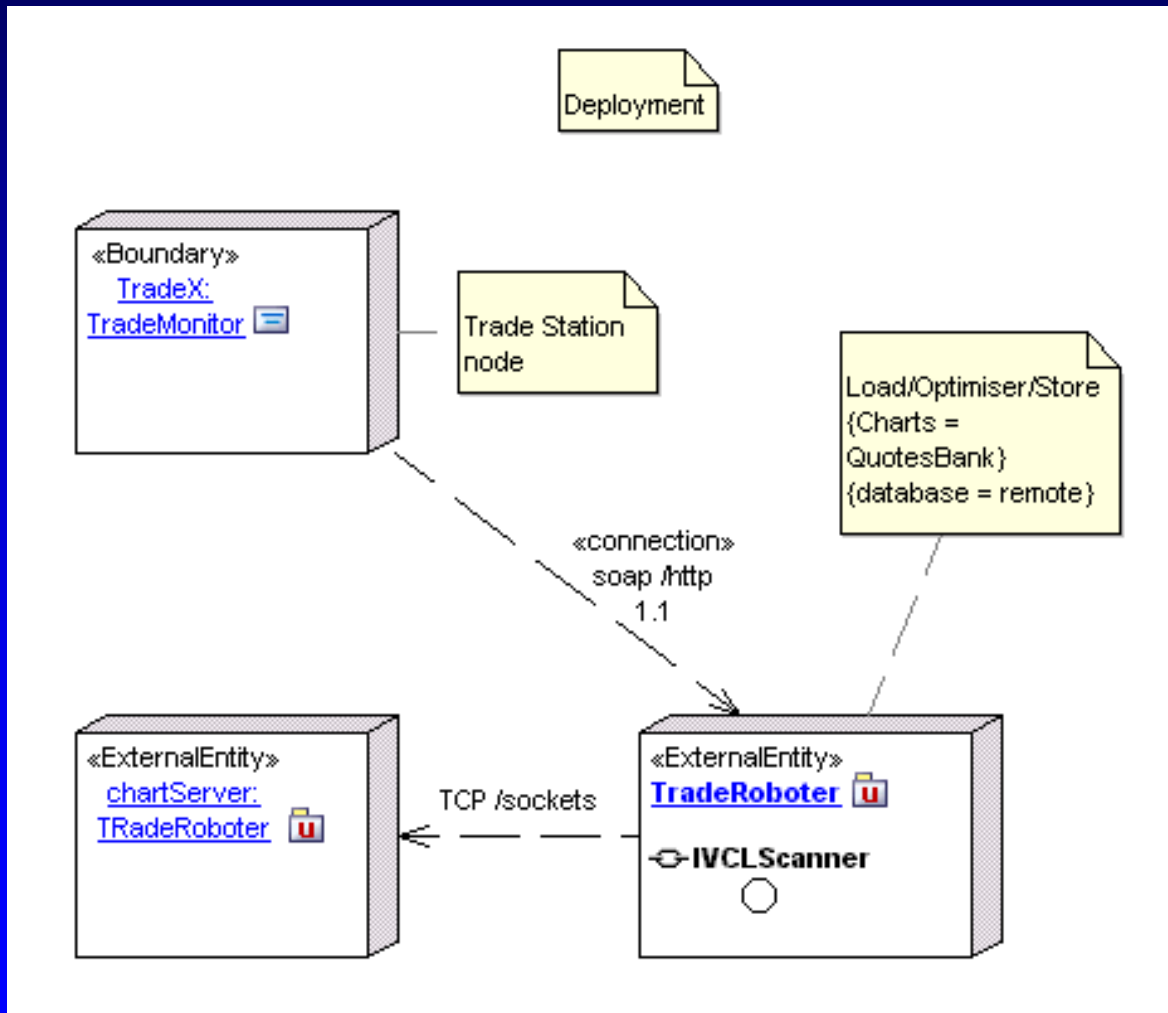
Component Diagrams show the structure of software in terms of software components and their relationship to executable components at run time

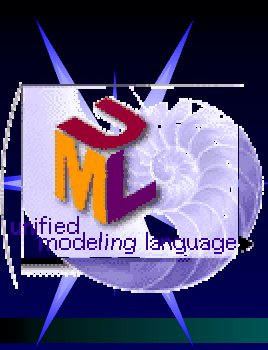
A model is complete in the sense that it fully describes the whole modeled system at the chosen level of abstraction and operational runtime





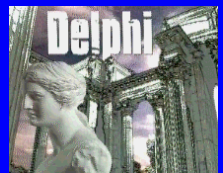
# Deployment

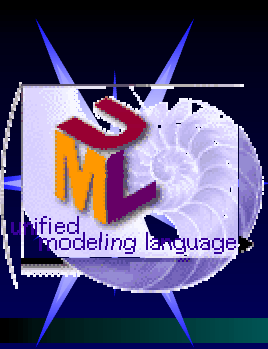




# Deployment

- ☯ Deployment Diagrams show how software components are mapped onto hardware devices and on which communication and protocol layer they work
  - ☯ Interesting for administrators and net workers in combination of a system manual
- 

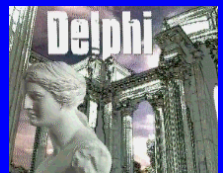




# A short test

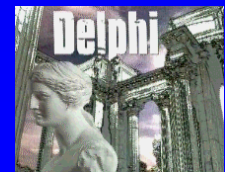
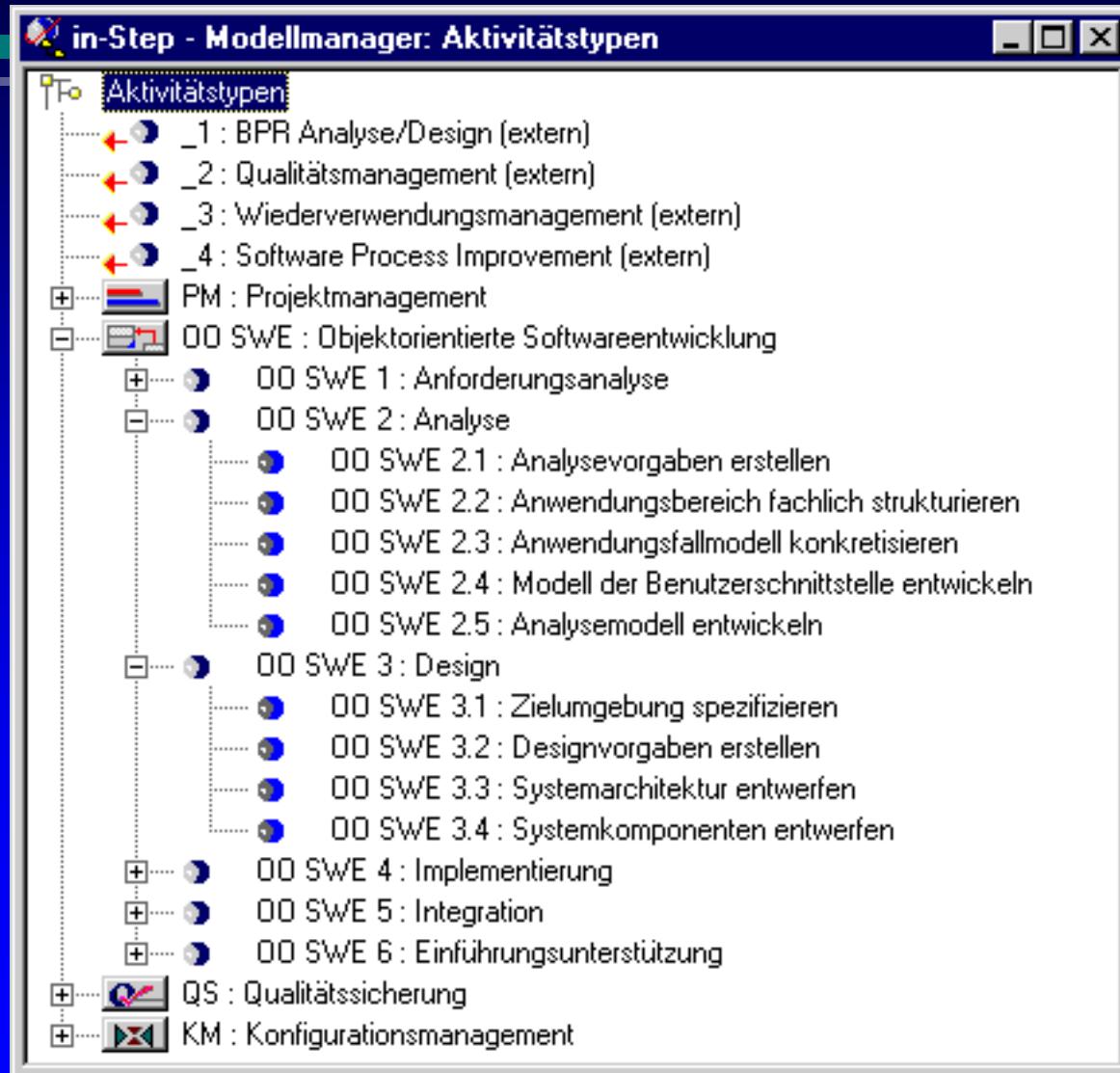
- ☹ In this overview, the UML was discussed in terms of UC, AD, CD, SE, SEQ, PAC, COM and DEP, means what?.
  - ☹ A class symbol is divided into three parts, which ones?
- 

- ☹ Where is the package ?
  - ☹ How many classes in the package?
  - ☹ Which sequence can we draw concerning the code in the package?
- 





...tools can be process oriented...



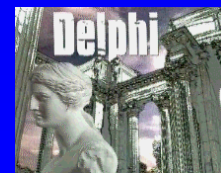


# ...or product oriented

**in-Step - Modellmanager: Produkttypen**

Produkttypen | Generalisierungen | Zustände

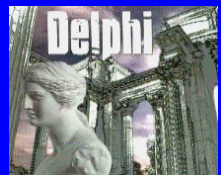
- 2.3 : Softwareentwurf
  - 2.3.1 : Designvorgaben
    - 2.3.1.1 : Ausgabedateitypen
    - 2.3.1.2 : Workbenchspezifischer Anwendungsrahmen
    - 2.3.1.3 : Stereotypen/benutzerdefinierte Eigenschaften
    - 2.3.1.4 : Code-Skripte
    - 2.3.1.5 : Packages zur Wiederverwendung
    - 2.3.1.6 : Entwurfsmuster
  - 2.3.2 : Systemarchitektur
    - 2.3.2.1 : Zielumgebung
    - 2.3.2.2 : Architekturkonzept
    - 2.3.2.3 : Design-Packages
    - 2.3.2.4 : Package-Diagramme zur Komponentengliederung
    - 2.3.2.5 : Persistenzkonzept
    - 2.3.2.6 : Transaktionskonzept
  - 2.3.3 : Designmodell
    - 2.3.3.1 : Designklassen
    - 2.3.3.2 : Klassendiagramme der Designklassen
    - 2.3.3.3 : Zustandsdiagramme zu Designklassen
    - 2.3.3.4 : Sequenzdiagramme für Designszenarien
- 2.5 : Softwareprodukte

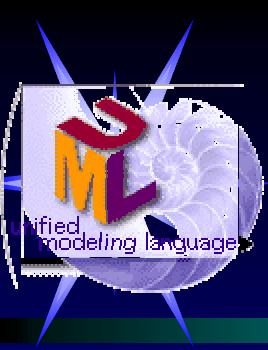




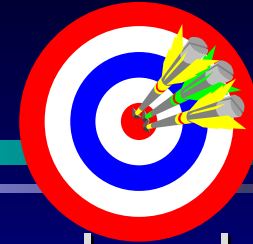
# Conclusion

- ☉ There are many different levels of UML notation. For this short presentation, the basic types of notation have been covered. For a more complete description of the UML notation, you can refer to the UML 2.1 Notation Guide. When beginning to use the UML, it is helpful to start with some basic notation (e.g., use case or class diagrams ) and then, as you become more proficient, advanced UML notation can be used.
- ☉ UML 2.0 can now be certified in a prometric test center worldwide.





# All the best!



- ☯ I'm still confused but on a much higher level
- ☯ Bubbles don't crash -->parachute approach
- ☯ If you can't see it, you can't manage it !
- ☯ [max.kleiner.com](http://max.kleiner.com)
- ☯ Book: «Patterns konkret»
- ☯ UML Script on:  
[/download/umlscript2.zip](#)
- ☯ [www.softwareschule.ch](http://www.softwareschule.ch)

