2IW80 Software specification and architecture

Software architecture: More Architecture Styles

Alexander Serebrenik

TU/e Technische Universiteit Eindhoven

University of Technology

Where innovation starts
Some Common Styles

- **Traditional, language-influenced styles**
  - Main program and subroutines
  - Object-oriented

- **Layered**
  - (Virtual machines)
  - Client-server

- **Data-flow styles**
  - Batch sequential
  - Pipe and filter

- **Shared memory**
  - Blackboard
  - Rule based

- **Interpreter**
  - Mobile code

- **Implicit invocation**
  - Event-based
  - Publish-subscribe

- **Peer-to-peer**
  - "Derived" styles
    - C2
    - CORBA
A look at different systems and their architectures

- **HZRS** aims at **automatic recognition of handwritten zip code recognition**.

- The process involves
  - hypothesizing the location of the ZIP code on the envelope
  - segmenting and recognizing ZIP code digits,
  - locating and recognizing City and State names,
  - looking-up the results in a dictionary (ZIP vs. City/State)

- Different image recognition/pattern recognition algorithms.

- What style would you use?
A look at different systems and their architectures

• **HZRS** aims at **automatic recognition of handwritten zip code recognition**.

• The process involves
  • hypothesizing the location of the ZIP code on the envelope
  • segmenting and recognizing ZIP code digits,
  • locating and recognizing City and State names,
  • looking-up the results in a dictionary (ZIP vs. City/State)

• Different image recognition/pattern recognition algorithms.
What styles are involved in e-mail communication?
What styles are involved in e-mail communication?

- Client-server

and many more...
Implicit Invocation Styles

• **Basic idea**
  - Event announcement instead of method invocation
  - “Listeners” register interest in and associate methods with events
  - System invokes all registered methods implicitly

• **Style invariants**
  - “Announcers” are unaware of their events’ effects
  - No assumption about processing in response to events
Publish-Subscribe

- **Subscribers** register/deregister to receive specific messages or specific content.
- **Publishers** broadcast messages to subscribers.

- **Analogy**: newspaper subscription
  - Subscriber chooses the newspaper
  - Publisher delivers only to subscribers
  - Ergo, publisher has to maintain a list of subscribers
  - Sometimes we’ll need proxies to manage distribution.

http://israel21c.org/israel-in-the-spotlight/going-on-vacation-dont-stop-your-newspaper-subscription-donate-it/
Players

- Subscriber 1
- Subscriber 2
- Subscriber n

- Stream
- Event
- Game Server

In: register, reg info
Out: none

In: new terrain, spacecraft
Out: none
Publish-Subscribe: Style Analysis

• **Summary:**
  - Subscribers register/deregister to receive specific messages or specific content.
  - Publishers broadcast messages to subscribers synchronously or asynchronously.

• **Design elements**
  - Components: publishers, subscribers
  - Connectors: procedure calls/network protocols
  - Data: subscriptions, notifications, published information

• **Topology:**
  - Either subscribers directly connected to publishers
  - Or via intermediaries
Publish-Subscribe: Style Analysis

• What are common examples of its use?
  • Social media “friend”
  • GUI
  • Multi-player network-based games

• What are the advantages of using the style?
  • Subscribers are independent from each other
  • Very efficient one-way information dissemination

• What are the disadvantages of using the style?
  • When a number of subscribers is very high, special protocols are needed
Event-Based Style

• In **Publish-Subscribe** the publisher is responsible for maintaining the list of subscribers.

• What if the subscribers were responsible for knowing their publishers?

We no longer need to distinguish publishers and subscribers!
Frequently called “event bus”

Commercial middleware
Event-Based Style: Style Analysis

• **Summary:**
  • Independent components asynchronously emit and receive events communicated over event buses

• **Design elements**
  • Components: concurrent event generators/consumers
  • Connectors: event bus (may be more than one)
  • Data: events

• **Topology:**
  • Communication via the event bus only
Event-Based Style: Style Analysis

• What are common **examples** of its use?
  • User interface software
  • Enterprise information systems with many independent components (financial, HR, production, …)

• What are the **advantages** of using the style?
  • Scalable
  • Easy to evolve (just add another component!)
  • Heterogeneous (as long as components can communicate with the bus they can be implemented in any possible way)

• What are the **disadvantages** of using the style?
  • No guarantee when the event will be processed
Peer-to-Peer Style

• In the Event-Based approach we no longer distinguish between publishers and subscribers
  • “Every component can act as publisher and/or subscriber”

• What if we try to do the same for “client-server”?
  • We had it in the layered (virtual machine) style
  • But it was restricted to the layered structure!
Peer-to-Peer Style

• In the Event-Based approach we no longer distinguish between publishers and subscribers
  • “Every component can act as publisher and/or subscriber”

• What if we try to do the same for “client-server”?
  • We had it in the layered (virtual machine) style
  • But it was restricted to the layered structure!

Peers:
• independent components
• can act as either clients or servers
Adapted version

- multiple landers need to communicate about the landing area to avoid collisions
- communication is possible only within a certain range
Peer-to-Peer: Style Analysis

• **Summary:**
  • State and behavior are distributed among peers which can act as either clients or servers.

• **Design elements**
  • Components: peers
  • Connectors: network protocols, often custom
  • Data: network messages

• **Topology:**
  • Network, usually dynamically and arbitrarily varying
Peer-to-Peer

- What are common **examples** of its use?
  - sources of information are distributed
  - network is ad-hoc
Peer-to-Peer Style: Style Analysis

• What are the advantages of using the style?
  • Robustness (if a node is not available the functionality is taken over)
  • Scalability
  • Decentralization

• What are the disadvantages of using the style?
  • Security (peers might be malicious or egoistic)
  • Latency (when information retrieval time is crucial)
Heterogeneous Styles

• More complex styles created through composition of simpler styles
  • Example: **Distributed objects**
    - OO + client-server network style
    - 2II45 Architecture of Distributed Systems
Some Common Styles

- **Traditional, language-influenced styles**
  - Main program and subroutines
  - Object-oriented

- **Layered**
  - (Virtual machines)
  - Client-server

- **Data-flow styles**
  - Batch sequential
  - Pipe and filter

- **Shared memory**
  - Blackboard
  - Rule based

- **Interpreter**
  - Interpreter
  - Mobile code

- **Implicit invocation**
  - Event-based
  - Publish-subscribe

- **Peer-to-peer**

- **“Derived” styles**
  - C2
  - CORBA
What style is implemented here?

BBC NEWS

What is this page?
This is an RSS feed from the BBC News - Home website. RSS feeds allow you to stay up to date with the latest news and features you want from BBC News - Home.

To subscribe to it, you will need a News Reader or other similar device. If you would like to use this feed to display BBC News - Home content on your site, please go here.

Help, I don't know what a news reader is and still don't know what this is about.

RSS Feed For: BBC News - Home
Below is the latest content available from this feed. This isn't the feed I want.

Ukraine 'planning Crimea withdrawal'
Ukraine is drawing up a plan to withdraw soldiers and their families from the Crimea region, the security chief in Kiev says.

FBI 'aids search for missing plane'
The FBI is aiding Malaysia's investigation into the disappearance of the airliner missing for more than a week, the White House says.

Pistorius 'on stumps when shooting'
A key police ballistics expert tells the Oscar Pistorius trial the South

Subscribe to this feed
You can subscribe to this RSS feed in a number of ways, including the following:

- Drag the orange RSS button into your News Reader
- Drag the URL of the RSS feed into your News Reader
What style is implemented here?

**What is this page?**

This is an RSS feed from the BBC News - Home website. RSS feeds allow you to stay up to date with the latest news and features you want from BBC News - Home.

To subscribe to it, you will need a News Reader or other similar device. If you would like to use this feed to display BBC News - Home content on your site, please go here.

> **Help, I don't know what a news reader is and still don't know what this is about.**

**RSS Feed For:** BBC News - Home

Below is the latest content available from this feed. This isn't the feed I want.

---

**Ukraine 'planning Crimea withdrawal'**

Ukraine is drawing up a plan to withdraw soldiers and their families from the Crimea region, the security chief in Kiev says.

**FBI 'aids search for missing plane'**

The FBI is aiding Malaysia's investigation into the disappearance of the airliner missing for more than a week, the White House says.

**Pistorius 'on stumps when shooting'**

A key police ballistics expert tells the Oscar Pistorius trial the South
2IW80 Software specification and architecture

Software architecture: Architecture Description Languages

Alexander Serebrenik

TU/e Technische Universität Eindhoven University of Technology

Where innovation starts
Sources for this topic

www.iso-architecture.org/ieee-1471/index.html

Welcome to the ISO/IEC/IEEE 42010 Website


SOFTWARE ARCHITECTURE
Foundations, Theory, and Practice

Richard N. Taylor • Nenad Medvidovic • Eric M. Dashofy
Architecture description language

- Architecture description elements should be somehow expressed

- An architecture description language (ADL) is any form of expression for use in architecture descriptions.
  - provides one or more model kinds to frame concerns of its stakeholders
Architecture description language

• Architecture description elements should be somehow expressed

• An architecture description language (ADL) is any form of expression for use in architecture descriptions.
  • provides one or more model kinds to frame concerns of its stakeholders:
    - narrowly focused (a single model kind)
    - widely focused (several model kinds; optionally organized in viewpoints)
  • often supported by automated tools to aid the creation, use and analysis of its models.
Examples of ADLs?

• An architecture description language (ADL) is any form of expression for use in architecture descriptions.

• You already know at least three architecture description languages

• Who can name them?
Examples of ADLs?

• An **architecture description language (ADL)** is any form of expression for use in architecture descriptions.

• You already know **at least three** architecture description languages

• Who can name them?
  • UML
  • Natural language
  • Notation we’ve used do describe architectural styles
Goal of this lecture

• Review and **compare** different ADLs

• We need **comparison criteria**!
  • Similarly to the analysis we have done for architectural styles
Evaluating ADLs: Criteria and Examples

• **Scope and purpose**
  • What does the technique help you model? What does it *not* help you model?
    – NL: capture design decisions in prose form
    – UML: design decisions in 13 diagram types

• **Basic elements**
  • What are the basic elements (the ‘atoms’) that are modeled? How are they modeled?
    – NL: any concepts required
    – UML: classes, associations, activities, nodes, use cases…
Evaluating ADLs: Criteria and Examples

• **Static and dynamic aspects**
  • What static (~structural) and dynamic (~behavioral) aspects of an architecture does the approach help you model?
    − NL: any aspect can be modelled
    − UML: some static diagrams (class, package), some dynamic (state, activity)

• **Dynamic modeling**
  • To what extent does the approach support models that change as the system executes?
    − NL: done manually, tool support: text editor
    − UML: depends on the environment but usually limited
Non-functional aspects

To what extent does the approach support (explicit) modeling of non-functional aspects of architecture?

- **NL**: expressive vocabulary available (but no way to verify)
- **UML**: almost no direct support; natural-language annotations
  - exception: time behavior – UML timing diagrams.
Evaluating ADLs: Criteria and Examples

• **Ambiguity**
  • How does the approach help you to avoid (or embrace) ambiguity?

• **NL is inherently ambiguous**
  • “John saw the man on the mountain with a telescope”.
  • Who has the telescope? John, the man on the mountain, or the mountain?
Evaluating ADLs: Criteria and Examples

• **Ambiguity**
  • How does the approach help you to avoid (or embrace) ambiguity?

• **NL is inherently ambiguous**

Possible solution

*The (name) interface on (name) component takes (list-of-elements) as input and produces (list-of-elements) as output (synchronously | asynchronously).*

• Can make data easier to read and interpret,
• However, such information is generally better represented in a more compact format…
Evaluating ADLs: Criteria and Examples

- **Ambiguity**
  - How does the approach help you to avoid (or embrace) ambiguity?

- **UML**
  - **Dependency**: the source uses the target in order to realize its functionality (but does not include an instance of it)
    - does A call B? use B? create B?...
    - solution: “profiles”

  \[ \langle \langle \text{call} \rangle \rangle \quad \langle \langle \text{use} \rangle \rangle \quad \langle \langle \text{create} \rangle \rangle \]
Evaluating ADLs: Ambiguity Summary

• **Ambiguity**
  • How does the approach help you to avoid (or embrace) ambiguity?
    - NL: tends to be ambiguous; statement templates and dictionaries help
    - UML: many symbols are interpreted differently depending on context; profiles reduce ambiguity
Evaluating Modeling Approaches (cont’d)

• **Accuracy**
  • How does the approach help you to assess the correctness of models?
    - NL: manual
    - UML: syntactic well-formedness checks, manual

• **Precision**
  • At what level of detail can various aspects of the architecture be modeled?
    - NL/UML: at any level
Evaluating Modeling Approaches

• **Viewpoints**
  • Which viewpoints are supported by the approach?
    - NL: any viewpoint (but no specific support for any particular viewpoint)
    - UML: each diagram type can be associated with at least one viewpoint
Evaluating Modeling Approaches

- **Viewpoints**
  - Which viewpoints are supported by the approach?
    - NL: any viewpoint (but no specific support for any particular viewpoint)
    - UML: each diagram type can be associated with at least one viewpoint

- **Viewpoint Consistency**
  - How does the approach help you assess or maintain consistency among different viewpoints?
    - NL: manual
    - UML: little in the languages itself; research prototypes

---

Validation

- **Recall**: Interaction diagrams should be consistent with the corresponding class diagrams and use case diagrams
- **Rule**: Objects in [sd] should be instances of classes in [cd]
- **Rule**: Name of the message [sd] should match an operation in the receiver’s class [cd]
Surveying Modeling Approaches

• Generic approaches
  • Natural language
  • UML, the Unified Modeling Language
  • PowerPoint-style modeling

• Domain- and style-specific languages
  • Koala
  • Weaves
  • AADL

• Early architecture description languages
  • Darwin
  • Rapide
  • Wright

• Extensible architecture description languages
  • Acme
  • ADML
  • xADL

and there are more!
Informal Graphical Model Example

- **User Interface Component**: Display simulator status, get new burn rate, invoke calculation.
- **Calculation Component**: Take burn-rate, calculate new height, velocity, fuel, simulator time, check for termination.
- **Data Store Component**: Store height, velocity, fuel, simulator time.
Informal Graphical Modeling

• General diagrams produced in tools like PowerPoint

• Advantages
  • Can be aesthetically pleasing
  • Size limitations (e.g., one slide, one page) generally constrain complexity of diagrams
  • Extremely flexible due to large symbolic vocabulary

• Disadvantages
  • Ambiguous, non-rigorous, non-formal
    – But often treated otherwise
  • Cannot be effectively processed or analyzed by machines/software
Related Alternatives

• Some diagram editors (e.g., Microsoft Visio) can be extended with semantics through scripts and other additional programming
  • Generally ends up somewhere in between a custom notation-specific editor and a generic diagram editor
  • Limited by extensibility of the tool

• PowerPoint Design Editor (Goldman, Balzer) was an interesting project that attempted to integrate semantics into PowerPoint
Informal Graphical Evaluation

- **Scope and purpose**
  - Arbitrary symbols + text diagrams

- **Basic elements**
  - Geometric shapes, splines, clip-art, text segments

- **Static & Dynamic Aspects**
  - Any aspect can be modeled, but no semantics behind models

- **Dynamic Models**
  - Rare, although APIs to manipulate graphics exist

- **Non-Functional Aspects**
  - With natural language annotations

- **Ambiguity**
  - Symbolic vocabulary/dictionaries

- **Accuracy**
  - Manual reviews and inspection

- **Precision**
  - Up to modeler; generally canvas is limited in size (e.g., one ‘slide’)

- **Viewpoints**
  - Any viewpoint

- **Viewpoint consistency**
  - Manual reviews and inspection
Darwin: an early ADL [Magee, Kramer 1991]

- General purpose architecture description language
  - graphical and textual visualizations
  - focused on structural modeling of systems

- Advantages
  - Simple mechanism for modeling structural dependencies
  - Repeated elements through programmatic constructs
  - Can be modeled in pi-calculus for formal analysis
  - Can specify hierarchical (i.e., composite) structures

- Disadvantages
  - Limited usefulness beyond simple structural modeling
  - No notion of explicit connectors
    - Although components can act as connectors
Darwin Example

```java
component DataStore{
    provide landerValues;
}

component Calculation{
    require landerValues;
    provide calculationService;
}

component UserInterface{
    require calculationService;
    require landerValues;
}

component LunarLander{
    inst
        U: UserInterface;
        C: Calculation;
        D: DataStore;
    bind
        C.landerValues -- D.landerValues;
        U.landerValues -- D.landerValues;
        U.calculationService -- C.calculationService;
}
```

Canonical Textual Visualization

Graphical Visualization
component WebServer{
    provide httpService;
}

component WebClient{
    require httpService;
}

component WebApplication(int numClients){
    inst S: WebServer;
    array C[numClients]: WebClient;
    forall k:0..numClients-1{
        inst C[k] @ k;
        bind C[k].httpService -- S.httpService;
    }
}
Darwin Evaluation

- Scope and purpose
  - Modeling software structure

- Basic elements
  - Components, interfaces, configurations, hierarchy

- Static & Dynamic Aspects
  - Mostly static structure; some additional support for dynamic aspects through lazy and dynamic instantiation/binding

- Dynamic Models
  - N/A

- Non-Functional Aspects
  - N/A

- Ambiguity
  - Rigorous, but structural elements can be interpreted in many ways

- Accuracy
  - Pi-calculus analysis

- Precision
  - Modelers choose appropriate level of detail through hierarchy

- Viewpoints
  - Structural viewpoints

- Viewpoint consistency
  - N/A
More advanced ADLs

- **Wright** [Allen 1997]
  - Syntax similar to Darwin
  - Based on communicating sequential processes
  - Descriptions can be model-checked (deadlock, anybody?)
  - High learning curve
  - Addresses a small number of system properties relative to cost

- **Koala**
  - inspired by Darwin
  - Domain-specific: product lines of embedded consumer-electronics devices
Koala Example

interface IDataStore{
    void setAltitude(int altitudeInMeters);
    int getAltitude();
    void setBurnRate(int newBurnRate);
    int getBurnRate();
    ...
}

configuration parameters

switch
More advanced ADLs

• **ArchJava** [Aldrich, Chambers, Notkin 2002]
  • extension of Java
    – components, ports, connections
  • enforces *communication integrity*

• **Fractal** [Bruneton, Coupaye, Leclercq, Quema, Stefani 2004]
  • components: *primitive* (code) and *composite* (group)
  • code generation
Extensible ADLs

• **Trade-off**
  - The expressiveness of general-purpose ADLs and
  - The optimization and customization of more specialized ADLs

• **Best of both worlds?**
  • **Use multiple notations in tandem**
    - (Difficult to keep consistent, often means excessive redundancy)
  • **Overload an existing notation or ADL (e.g., UML profiles)**
    - Increases confusion, doesn’t work well if the custom features don’t map naturally onto existing features
  • **Add additional features we want to an existing ADL**
    - But existing ADLs provide little or no guidance for this
  • **Extensible ADLs attempt to provide such guidance**
xADL

- Modular XML-based ADL intended to maximize extensibility both in notation and tools

**Advantages**
- Growing set of generically useful modules available already
- Tool support in ArchStudio environment
- Users can add their own modules via well-defined extensibility mechanisms

**Disadvantages**
- Extensibility mechanisms can be complex and increase learning curve
- Heavy reliance on tools
<types:component xsi:type="types:Component"
    types:id="myComp">
  <types:description xsi:type="instance:Description">
    MyComponent
  </types:description>
  <types:interface xsi:type="types:Interface"
      types:id="ifacel">
    <types:description xsi:type="instance:Description">
      Interface1
    </types:description>
    <types:direction xsi:type="instance:Direction">
      inout
    </types:direction>
  </types:interface>
</types:component>
xADL Example

```
<types:component xsi:type="types:Component"
    types:id="myComp">
    <types:description xsi:type="instance:Description">
        MyComponent
    </types:description>
    <types:interface xsi:type="types:Interface"
        types:id="ifacel">
        <types:description xsi:type="instance:Description">
            Interface1
        </types:description>
        <types:direction xsi:type="instance:Direction">
            inout
        </types:direction>
    </types:interface>
</types:component>
```

```
component{
    id = "myComp";
    description = "MyComponent";
    interface{
        id = "ifacel";
        description = "Interface1";
        direction = "inout";
    }
}
```
<types:component xsi:type="types:Component"
    types:id="myComp">
    <types:description xsi:type="instance:Description">
        MyComponent
    </types:description>
    <types:interface xsi:type="types:Interface"
        types:id="iface1">
        <types:description xsi:type="instance:Description">
            Interface1
        </types:description>
        <types:direction xsi:type="instance:Direction">
            inout
        </types:direction>
    </types:interface>
</types:component>
ArchStudio Environment
Scope and purpose
- Modeling various architectural concerns with explicit focus on extensibility

Basic elements
- Components, connectors, interfaces, links, options, variants, versions, ..., plus extensions

Style
- Limited, through type system

Static & Dynamic Aspects
- Mostly static views with behavior and dynamic aspects provided through extensions

Dynamic Models
- Models can be manipulated programmatically

Non-Functional Aspects
- Through extensions

Ambiguity
- Base schemas are permissive; extensions add rigor or formality if needed

Accuracy
- Correctness checkers included in ArchStudio and users can add additional tools through well-defined mechanisms

Precision
- Base schemas are abstract, precision added in extensions

Viewpoints
- Several viewpoints provided natively, new viewpoints through extensions

Viewpoint consistency
- Checkable through external tools and additional consistency rules
2IW80 Software specification and architecture

Software architecture: Architecture Evolution (a brief primer)

Alexander Serebrenik
How do architectures change with time?

Recall...

• **Architectural drift** is introduction of principal design decisions into a system’s descriptive architecture that
  - are not included in, encompassed by, or implied by the prescriptive architecture
  - but which *do not violate* any of the prescriptive architecture’s design decisions

• **Architectural erosion** is the introduction of architectural design decisions into a system’s descriptive architecture that *violate* its prescriptive architecture

But what happens *in practice*?
- F1 concerns system architecture and is stable.
- Conjecture: system architecture is stable
  - No requirements being added
Architecture grows but the growth is limited

Number of internal dependencies in Eclipse: added, kept, kept from r1, deleted.

- Wermelinger, Yu, Lozano observed **linear growth** for architecture
  - Lehman predicts linear growth of the system **size** [conservation of familiarity]
  - Godfrey, Tu observe **superlinear** growth of system size in Linux
Architecture as implemented

• Another way to study architecture evolution

Code ➔ Architecture

Architecture Reconstruction
import java.util.*;

class User {
    int userCode;
    String fullName;
    static int nextCodeAv = 0;
    ...
    public User(String name) {
        fullName = Name;
        userCode = User.nextCodeAv++;
    }

    public int getCode() {
        return userCode;
    }
}

User

userCode: int
fullName: String
nextCodeAv: int

User(String name)  
getCode(): int

Do not draw the library classes
What about the relationships?

<table>
<thead>
<tr>
<th>Relationships</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Association / aggregation</strong></td>
<td>class A { B b; }</td>
</tr>
<tr>
<td><img src="A" alt="A" /> ➔ <img src="B" alt="B" /></td>
<td></td>
</tr>
<tr>
<td><strong>Dependency</strong></td>
<td>class A { void f(B b) {b.g(); } }</td>
</tr>
<tr>
<td><img src="A" alt="A" /> ➔ <img src="B" alt="B" /></td>
<td>class A { void f() {B b; ... b.g();} }</td>
</tr>
<tr>
<td><strong>Generalization</strong></td>
<td>class A extends B {...}</td>
</tr>
<tr>
<td><img src="A" alt="A" /> ➔ <img src="B" alt="B" /></td>
<td></td>
</tr>
<tr>
<td><strong>Realization</strong></td>
<td>class A implements B {...}</td>
</tr>
<tr>
<td><img src="A" alt="A" /> ➔ <img src="B" alt="B" /></td>
<td></td>
</tr>
</tbody>
</table>
More about architecture evolution?

- Reconstruction is *much* more difficult
  - weakly typed containers, generics, …
  - behavioral diagrams (state machines, sequence diagrams)
  - advanced programming techniques (EJB interceptors, …)

- Model comparison
  - Manual – visualization challenges
  - UMLDiff, EMFDiff
  - Semantic differences

- 2IS55 Software evolution
Major problems with current ADLs

• No support for evolution during the execution
  • Architecture can change during the execution:
    – client connects to a different server
    – Check Müller, Villegas on Runtime evolution in
  • Snapshots easily become obsolete

• No support for evolution due to change in requirements
  • Wright
    – model checking: no incremental approach
    – minor modification ⇒ everything should be rechecked
  • ArchJava
    – no real separation of architecture/implementations
    – overtly complex code
Some Common Styles

- **Traditional, language-influenced styles**
  - Main program and subroutines
  - Object-oriented
- **Layered**
  - (Virtual machines)
  - Client-server
- **Data-flow styles**
  - Batch sequential
  - Pipe and filter
- **Shared memory**
  - Blackboard
  - Rule based

- **Interpreter**
  - Interpreter
  - Mobile code
- **Implicit invocation**
  - Event-based
  - Publish-subscribe
- **Peer-to-peer**
- **“Derived” styles**
  - C2
  - CORBA

Architecture description language

- Architecture description elements should be somehow expressed

- An architecture description language (ADL) is any form of expression for use in architecture descriptions.
  - provides **one or more model kinds** to frame **concerns** of its stakeholders

![Architecture Diagram]

*University of Technology*
Some Common Styles

- **Traditional, language-influenced styles**
  - Main program and subroutines
  - Object-oriented
- **Layered**
  - (Virtual machines)
  - Client-server
- **Data-flow styles**
  - Batch sequential
  - Pipe and filter
- **Shared memory**
  - Blackboard
  - Rule based
- **Interpreter**
  - Interpreter
  - Mobile code
- **Implicit invocation**
  - Event-based
  - Publish-subscribe
- **Peer-to-peer**
- **“Derived” styles**
  - C2
  - CORBA