Computer Science for Engineers

Lecture 5
Software Engineering – part 2

Prof. Dr. Dr.-Ing. Jivka Ovtcharova
Dipl. Wi.-Ing. Dan Gutu
5th of December 2008
Rational Unified Process (RUP) is an object oriented process model for software development. It is also a commercial product of the Rational Software company - which since 2002 has been a part of IBM.

RUP uses Unified Modeling Language (UML) as the notation language.

The life-cycle of RUP is divided into 4 principle phases:

- **Inception** (conceptualization: determine project extent)
- **Elaboration** (Design: develop convertible architecture)
- **Construction** (Implementation: fill the architectural skeleton with functionality)
- **Transition** (Product transfer: transfer application to the user environment)
Rational Unified Process: Dynamic Structure

3.3. Methods of OO Analysis and Design

3.3.1 Introduction

Distribution of tasks in small steps (*Iterations*)

Source: IBM - Rational
Phases of the Rational Unified Process: *Inception* and *Elaboration*

- **Inception** (Conceptualization)
  - Specification of the vision for the end product
  - Specification of the essential transactions
  - Definition of the project extent
  - Project costs and risks
  - Finish: **Life Cycle Objective Milestone**

- **Elaboration** (Design)
  - Specification of product characteristics
  - Design of architecture
  - Design of the necessary activities and resources
  - Finish: **Life Cycle Architecture Milestone**
Phases of the Rational Unified Process: **Construction** and **Transition**

- **Construction** (Implementation)
  - Generation of the product
  - Development of the architecture
  - Result: manufactured product
  - Finish: **Initial Operational Capability Milestone**

- **Transition** (Product transfer)
  - Release of the product to the user
  - Test of the quality level
  - Delivery, Training, Customer Support, Maintenance
  - Finish: **Release Milestone**
Phases of the Rational Unified Process

- The results of the phases are the so called **Milestones**:
  - **Lifecycle objectives milestone**: Vision including a rudimentary use case model, (essential functionalities), tentative architecture, identification of important risks, design of the engineering phase
  - **Lifecycle architecture milestone**: Architecture prototype, detailed use case model, design of the construction phase
  - **Initial operational capability milestone**: Design models and Beta-Release of the Software
  - **Product release milestone**: Release in production quality
Disciplines of RUP: Static Structure

- The disciplines (interchangeably used with working steps) orient themselves to **special roles within the developing team**, and are administered by certain people or groups.

- In detail, these disciplines are:
  - **Central work steps**
    - Business Modeling
    - Requirements
    - Analysis & Design
    - Implementation
    - Test
    - Deployment
  - **Supported work steps**
    - Configuration & Change Management
    - Project Management
    - Environment
**Rational Unified Process**

3.3. Methods of OO Analysis and Design

3.3.1 Introduction

- Business Modeling
- Planning
- Initial Project Plan
- Evaluation
- Deployment
- Test
- Analysis & Design
- Implementation
- Configuration & Change Management
- Environment

Source: IBM - Rational
Business Modelling

• Define the **core problem**:  
  - Which functions should the system offer?  
  - How should the operating surface behave?  
  - How efficient, safe, … must the system be?  

• Define the **relevant environment**:  
  - type, number of users  
  - available hardware, previous software  

• Estimate **feasibility**  
  - Technical, personal capacity  
  - Costs (and uses)
Requirements

- Purpose of the system
- Desired functions
- Correct / false input, corresponding reactions
- Configuration of the operating surface
- Efficiency - goals
- Document requirements
- Protection / Safety aspects
- Standards to use
- Time plan, expenditure estimate, risk estimate (approx.)
Analysis and Design

- Rough system structure
  - Analyze the components
  - Define teamwork
  - Test design vs. requirements

- Describing the components
  - Purpose and role of a component
  - Service offered from one component

- Relationships between components
  - Which components can use the other ones
  - Forming of design decisions
Implementation and tests

- **Implement the components**
  - Choose data structures
  - Choose algorithms
  - Formulate in the programming language

- **Document the components**
  - How do the components accomplish their tasks?
  - Establish implementation alternatives

- **Test components** (vs. design)
  - Arrange the test environment, gather test data
  - Perform a run through of the test
  - Verification

- **Test externally programmed components**
- Application
  - Install the system for the customer
  - accommodate
  - Inspection by the customer
  - Transfer of the system
  - Train the User

- Maintenance
  - Correct errors
  - Modify: functionalities adapt/ better
<table>
<thead>
<tr>
<th>Diagram type</th>
<th>Diagram</th>
<th>Disciplines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case</td>
<td>Use case diagram</td>
<td>Business modeling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requirements</td>
</tr>
<tr>
<td>Static model</td>
<td>Class diagram</td>
<td>Analysis &amp; Design</td>
</tr>
<tr>
<td></td>
<td>Instance diagram</td>
<td>Implementation</td>
</tr>
<tr>
<td>Dynamic model</td>
<td>Collaboration diagrams</td>
<td>Analysis &amp; Design</td>
</tr>
<tr>
<td></td>
<td>Sequence diagrams</td>
<td>Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Analysis &amp; Design</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Implementation</td>
</tr>
<tr>
<td></td>
<td>State diagrams</td>
<td>Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Analysis &amp; Design</td>
</tr>
<tr>
<td></td>
<td>Activity diagrams</td>
<td>Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Analysis &amp; Design</td>
</tr>
</tbody>
</table>
3.3.2. Example: Modeling a M&S System
The Rational Unified Process is very complex, since it deals with a broad band of systems.

For small and middle-sized projects, simple process models are designed.

The rough course of the corresponding visualized process models is equivalent to RUP.

In order to distinguish these process models from the company designed product „Rational Unified Process“, the model is called „Irrational Separated Process“ (ISP).
To further explain ISP, **UML diagrams** are used (activity and class diagrams).

ISP is an **iterative process**.

Iterations can be of different types:

- Return to an earlier phase to recover an error
- Handling of a **complementary functionality** that was previously not recognized or purposely not considered in order to „not be overwhelmed with complexity“

UML is used mainly in the first 3 phases; requirements, analysis, and design.

To keep framework intact, nothing more will be said about the implementation, test, start-up, and application phases in this example.
The process model ISP (2)

3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

- Requirements description
- Analyse
  - [else]
  - [Requirements model complete]
- Concept
  - [else]
  - [Concept model complete]
- Implementation
  - [else]
  - [Implementation model complete]
- Test
  - [else]
  - [Test successful]
- Placing into operation
  - [else]
- Usage
  - [else]
  - [Supplementary functions desired]

[Requirements model complete]
[Concept model complete]
[incremental functional enhancement complete]
[Maintenance required]
• **Sub activities of the requirements description:**
  - Create model of the problem domain
  - Create use case model
  - Create interfaces

• **Components of the requirements model**
  - Problem domain model: [class diagram]
  - Use case model
    - Activity diagram
    - Use case diagram
    - Use case description
  - Interfaces
    - User interface definition
    - System interface definition
Requirements: Partial Activities

3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

- Create problem domain model
- Create use case model
- Create interface model

- Problem domain model
- Use case model
- Interface model

[else]

[no other use cases left]
Elements of the Requirements Model

3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

- Problem domain model
- Use case model
- Interface model

<<UML>>

- Class diagram
- Activity diagram
- Use case diagram
- Use case description
- UI - Definition
- System interface definition
The **domain problem model** exhibits a **conceptional model** of each real segment that should be embedded in the application to be developed. It describes the **world of the future system user**, without going into technical details that are connected with the latter system.

- This **conceptional model** should above all synchronize the **understanding** that the developers have with each customer. Most importantly it serves as a **communication medium**.

- The **most important element** of the problem area model is the **class diagram** that reflects the **static structure of the observed system**.
• A M&S program should be developed to support the mechatronic development process.

• The program should support a library with physical components (engine, gear box, axel, electric resistor, hydraulic pump)
  - The behavior of each physical component will be described through an equation.
  - The combined possibilities of each component will be defined through an interface.
  - A component is comprised either from other components, or will be described through an equation

• Physical interconnections (cable, wire, therm. heat flow) connect the components to a mechatronic system.

• The mechatronic system is modeled through components and their interconnections.
Components that are composed of other components (engine, gear box, axes)

Components that are described through equations (centrifugal mass, gear, …)

Component equation:

```plaintext
w = der(phi);
ma = der(w);
J*ma = flange_a.tau + flange_b.tau;
```
Example - Problem Domain Model

3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

- Mechatronic system
  - Interface
    - Equation: String
      - Equation: String
        - isCompatible():Boolean
  - Physical assembly
    - 0..*
  - 1
  - 1

  - Equation: String
  - Interface
    - 1
    - 2
    - isCompatible():Boolean
    - 2
    - 1

  - Inductor
    - Inductivity: Double
  - Resistor
    - Resistance: Double
  - Body
    - Mass: Double
  - Spring
    - Spring constant: Double
  - Damper
    - Damping constant: Double
  - Condensator
    - Capacity: Double
  - Cable
    - Length: Double
    - Diameter: Double
    - Max. potential: Double
    - Mx. Intensity: Double
  - Arbor
    - Length: Double
    - Diameter: Double
    - Max. torque: Double
    - Mx. revolution speed: Double
  - Flange
    - Position: Double
    - Force: Double
  - Electric clamp
    - Electric potential: Double
    - Intensity: Double
The **use case model** describes the **desired system functionality**, in which it defines **single use cases from the system, or supported use cases**.

- **Main elements of the use case model:**
  - **Use case diagram:**
    should give an overlook over the entire functionality
  - **Use case description:**
    - **Verbal**: specifies each use case in its natural language (German, English, …)
    - Whenever such a verbal specification is considered to be not precise enough, the specification can be **formalized through 3 forms of UML-diagrams**:
      - Activity diagrams
      - Sequence diagrams
      - Collaboration diagrams
3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

Use Case Model – Use Case Diagram

M&S System

- Create model
- Modify model
- Store model
- Open model
- Optimize model parameter
- Simulate
- Plot diagrams
- Visualize simulation in VR

- Constructor
- Employe
- Planer
• A simulation can be performed by a qualified user at that point in time (constructor or designer). For this, the following steps must be performed:

1. The user selects the model which should be simulated.
2. The user types in the simulation parameters.
3. The user starts the simulation.
4. The user selects the form of output for the simulation results (i.e. plot, animation, or VR).
5. When the simulation results are not satisfactory enough, steps 2-4 can be repeated.
3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

Activity Diagram „Perform Simulation“
3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

**Activity Diagram (1)**

- **User**
  - Select model
  - Enter simulation parameters
  - Other parameters: All parameters OK
  - Start simulation

- **M&S System**
  - Load model
  - Model
  - Simulation parameters
  - Translate model

Flow:
- Select model
- Load model
- Enter simulation parameters
- Other parameters: All parameters OK
- Start simulation
- Translate model

Decision Points:
- [OK] for Load model
- [invalid model]
- [Model defective]
- [Model OK]
3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

Activity Diagram (2)

Simulation results

Results not satisfactory

Choose output form

[Plot]

Plot results

[else]

Animate results

[else]

Represent results in VR

Evaluate simulation results

Results ok

Compute simulation

Simulation parameters
The finished use case model is an **integrated element of the contract between the customer and the employer**, as it **standardizes the functionality of future usage**.

The **inherent iterative character of the process model** naturally includes **frequent modifications of the use case model**, in which most of these modifications exhibit **refining and corrections**, and do not tarnish the contract.

Aside from requirements contained in the use case model, so called **non-functional requirements** can be included in the contract. I.e., side conditions in reference to the performance, reliability, ergonometry, etc.

Since UML offers no notation elements in order to formalize such requirements, this topic will be waived until further discussion.
A phase of the **detailed analysis** of the **domain problem model** on one side and of the **use case** on the other side follows the preceding determination of the requirements of the system.

**Objective:** to obtain **complete and contradiction free models** that describe the exact structure of the object in the problem area, its behavior, and its wavering interactions for solving the given tasks.

The resulting **analysis model** reflects a **system still independent from a concrete implementation**.

In contrast to the requirement model, the analysis contains **objects** that do not come from the considered reality segments, but rather from the **solution area that is to be assigned**.
The objects of the analysis model can be divided into 3 categories:

- **Entity objects**, that correspond with the real objects of the problem area
- **Boundary objects**, that serve as the interaction with the actors
- **Control objects**, that encapsulates the overlapping object behavior—many cases deal with objects that coordinate with the execution of single use cases.

The purpose of this structuring is to obtain a system structure as stable as possible, that should guarantee the possibility of a **maximum of change locality**, before the real design phase. That means that expected changes of the system requirements should affect as few objects as possible.
Analysis (3)

• Partial activity of the analysis
  - Refine the domain problem model: generate a structure model
  - Partition the structure model: refine the structure model
  - Generate a dynamic model

• Elements of the analysis model
  - Structure model: class diagram
  - Dynamic model
    - Collaboration diagram
    - Sequence diagram
    - State diagram
    - Activity diagram

3.3. Methods of OO Analysis and Design
3.3.2. Example: modelling of M&S Systems
3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

Refine domain problem model

Structure model [created]

Partition structure model

Structure model [refined]

Create dynamic model

Dynamic model [created]

[no other use cases left]
3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

- Analysis model
- Structure model
- Dynamic model
  - Collaboration diagram
  - Sequence diagram
  - State diagram
  - Activity diagram
Structure Model

- The result of refining the problem domain model is the **structure model**.
- The structure model does not replace the domain problem model. It serves, as before, to be a simple communication vehicle between the contractor and the developer.
- The next partial activity of the analysis is the **partitioning of the possibly too complex structure models in manageable, cohesive packets**. This step will not be further discussed due to space and time reasons.
3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

Structure Model - Example
Dynamic Model

- The second level of the analysis phase consists of generating the **dynamic model**.
- It describes the **dynamic behavior of the system in dynamic diagram form**.
- For more complex use cases **activity, sequence, and collaboration diagrams** can be applied.
- **State diagrams** can be formatted for classes whose objects feature non-trivial life cycles. Particular user and system interface classes belong to these diagrams.
3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

Dynamic Model: Sequence Diagram „Start simulation“

- **StartSimulation**
- **TranslateModel()**
- **DetermineComponentEquation**
- **TransmitEquation**
- **UniversalMathematicEquation**
- **ListMathematicEquations**
- **DetermineNestingStructure**
- **TransmitConnectedComponents**
- **ListConnectedComponents**
- **TransformComponentEquations**
- **SetupEquationSystem()**
- **SimplifyEquationSystem()**
- **NormedEquationSystem()**
- **SolveEquationSystem()**
- **EquationSystemSolution**

- **SimulationComputation()**
- **TranslatedModel**
• During the concept phase, all **technical decisions** that are **essential for the implementation** must be made.

• The more general analysis model must be specialized so that it fulfills any resulting side conditions from these decisions.

• While in both previous phases the focus of modeling lay in **WHAT** the future system should do, the question now becomes **HOW**.

• The concept phase is roughly outlined in 2 parts
  - System concept
  - Detail concept
• Partial activity of the concept
  - System concept: Architecture model
  - Detail concept: refining of the structure model and the dynamic model

• Elements of the concept model
  - Structure model
  - Architecture model
    - Component diagram
    - Distribution diagram
    - Architecture description
  - Dynamic model
Concept: Partial Activity

3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

System concept

Detailed concept

Architecture model

Structure model [detailed]

Dynamic model [detailed]

[no further refinement necessary]
3.3. Methods of OO Analysis and Design

3.3.2. Example: modelling of M&S Systems

- Concept model
  - Structure model
    - <<UML>> Component diagram
    - <<UML>> Distribution diagram
    - Architecture description
  - Architecture model
    - <<UML>> Component diagram
    - <<UML>> Distribution diagram
    - Architecture description
  - Dynamic model
In the course of the concept, the „big decisions“ are made. They deal with, among others:

- **the implementation environment**: system software, programming language, included foreign products (i.e. class libraries)
- **the persistence mechanisms**: the form and way that information will be memorized, how the single program running will survive
  - Example: differentiate between the **data system**, or relational or object oriented **data base system** options.
- **The possible distribution** to multiple computers (i.e. Java-Applets) and the belonging **communication mechanisms**: which software components will replace which hardware nodes and how successful is the communication between the nodes?
- **The communication structure** between the components **within** a node.
Example: Architecture Description

- Implementation environment:
  - Programming language: Java
  - Developing environment: Eclipse
  - Platform: platform independence (Java)
- Persistence mechanisms: Data system
- Concurrency: Single-Threaded System
  - Synchronous execution
  - News will be implemented as a method call
Detailed Concept

- The architecture model (result of the system design) will be used as input of the corresponding **detail concept**.

- In detail concept the refining of the structure model and of the dynamic model of the analysis is carried out in order to convert the decisions of system design. Hence the following work:
  - **Complementing the class diagrams with relevant interface classes**, i.e. in order to implement the interfaces postulated in the analysis with the persistence mechanisms.
  - **Integration of the class descriptions**, such as visibility or operations for the direct access to attributes, as well as programming language dependent information, as in virtual Java-Methods
  - **Modification** of certain model elements in respect to future re-use.
Refined Structure Model - Sequence Diagram

3.3.2. Example: modelling of M&S Systems

loadModel → aSimulator

New ModelCreator → XMLReader

createModel

XMLParser
buildMass
buildDamper
buildSpring

returnCreatedModel

SimulationModel

Source: after [GHJV01]
<table>
<thead>
<tr>
<th>Reference</th>
<th>Title and Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>[HiKa99]</td>
<td>M. Hitz, G. Kappel: „UML@Work“, dpunkt.verlag, 1999</td>
</tr>
</tbody>
</table>
Computer Science for Engineers

Lecture 5
Data structures – part 1

Prof. Dr. Dr.-Ing. Jivka Ovtcharova
Dipl. Wi.-Ing. Dan Gutu
5th of December 2008
Outline

Lecture Content

1. Preface
2. Basics
3. Object orientation
4. Data Structures
  4.1. Introduction
  4.2. Graphs
  4.3. Use of Graphs
  4.4. Trees
  4.5. Use of Trees
  4.6. Linked Lists
  4.7. Queues and Stacks
5. Algorithms
4. Data Structures

4.1 Introduction

- According to definition 1.1 from lecture 1, computer science is the science that concerns itself with the structure, effectiveness, construction principles, and applications of information processing systems as well as their usage.

- In order to complete these tasks, computer science developed a formalism, which allows verified characteristics and exceptions to structures, effectiveness, and principles.

- This formalism and the possibility for their verification are prerequisites for a systematic data processing, which can be automatized through a computer.
Data Structures - Examples

• A **data structure** in computer science is a **specific form to manage data and link the structures** with one another, so that it’s possible to access the data and especially manipulate it in an applicable manner.

• Data structures are always linked with certain operations to allow these manipulations.

• Application examples are
  - Data pool systems
  - Networks
  - Lists of assembly parts
  - Telephone book
  - Pictures
• The different types of data structures are applicable for different specific tasks according to their characteristics, linked operations, and implementation.

• Decision criteria are, for example
  - Amount of data to be managed
  - Data fluctuation
  - Necessary access possibilities
  - Run time behavior of the data access
  - Alterability of the data structure
A graph is a formation of vertices that are connected through edges.

Graphs are visually described through images and through mathematical structures. This general definition allows the description of complex associations and characteristics.
Definition 3.1: \( G = (V,E) \) is a graph if and only if

1. \( V \) is heap \((\text{vertices})\)
2. \( E \subseteq \{ \{v_1,v_2\} \mid v_1,v_2 \in V \} \) \((\text{edges})\)

example

\[ G_1: \]

\[ V = \{v_1,v_2,v_3\} \]
\[ E = \{ \{v_1,v_2\}, \{v_2,v_3\} \} \]

Nomenclature

In the graph \( G_2 = (\{v_1\},\{v_1\}) \)
the edge \( e_1 = \{v_1\} \) is a loop.

\[ G_2: \]

\[ V = \{v_1\} \]
\[ E = \{ \{v_1\} \} \]
**Definition 3.2:** Let $G = (V,E)$ and $G' = (V',E')$.

$G'$ is a **subgraph** of $G$ if and only if 
\[ E' \subseteq E \text{ and } V' \subseteq V \quad (\iff G' \subseteq G) \]

**example**

**$G_3$:**

\[
V = \{v_1, v_2, v_3, v_4\} \\
E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_2\}\}
\]

**$G_3'$:**

\[
V' = \{v_1, v_2\} \\
E' = \{\{v_1, v_2\}, \{v_2\}\}
\]
Definition 3.3: A path \( \pi \) in \( G \) is a sequence of \( \pi = (v_0, v_1, \ldots, v_n) \) with \( \forall 0 \leq i < n : \{ v_i, v_{i+1} \} \in E \).

Example

\( G_4 \):

\[ \pi = (v_1, v_4, v_3) \]

Nomenclature

\( n = |\pi| = \text{Length of } \pi \)
\( \text{Start}(\pi) = v_0 \)
\( \text{End}(\pi) = v_n \)

The total of all paths \( \pi \) in \( G \) is \( \text{Path}(G) \).
Definition 3.4: The path $\pi$ in $G$ is a **circle** if and only if

$$\text{Start}(\pi) = \text{End}(\pi)$$

**Example**

$G_4$:

$$\pi = (v_2, v_4, v_3, v_2) \text{ is a circle}$$

$$\pi = (v_1, v_4, v_2) \text{ is NOT a circle}$$

**Nomenclature**

A path $\pi$ is a **simple edge** circle, when each edge on the path $\pi$ is contained only once in the path.
**Definition 3.5:** the branching factor $d$ for a graph $G$ and a vertex $v$ is the number of edges connected with the vertex $v$.

$$d(G,v) = \#(v' \in V \mid \{v,v'\} \in E)$$

The branching factor $d$ for a graph $G$ gives the maximal branching factor for all vertices.

$$d(G) = \text{Max}(d(G,v))$$

**example**

$G_5$:

- $d(G_5,v_3) = 2$
- $d(G_5,v_4) = 1$
- $d(G_5) = 5$
A graph $G$ was previously described by the amounts $E$ and $V$ and can also be described through the input of a weight function $g$.

Let $G = (V, E)$ and $g : V \times V \to \mathbb{R}$ be the weight function.

Therefore:
\[
g(v_1, v_2) = \infty \Rightarrow \{v_1, v_2\} \notin E
\]
\[
g(v_1, v_2) \neq \infty : \text{weight of the edge } \{v_1, v_2\} \in E
\]

That means that an edge is an element of the graph when the weight is smaller than $\infty$.

**example**

\[
G_6:
\]

\[
g(v_1, v_2) = 2
\]
\[
g(v_2, v_3) = \infty
\]
\[
g(v_1, v_3) = 5
\]
Definition 3.6: A **directed graph** $G$ is a pair $G = (V,R)$ with

1. $V$ is a heap (**vertices**)
2. $R \subseteq V \times V$ (**directed edges**)

(The order of the vertices in an edge gives the direction)

**example**

$G_6$:

$V = \{v_1, v_2, v_3, v_4\}$

$R = \{\{v_2, v_1\}, \{v_3, v_1\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_2, v_2\}\}$
Example of a Graph (1)

KVV – bus and street car map
Example of a Graph (2)

How do I get from the main train station to the university (path)?

Start(π) = Hauptbahnhof

End(π) = Kronenplatz / University

path π = {Hauptbahnhof, Poststraße, Augartenstraße, Kongresszentrum, Ettlinger Tor, Marktplatz, University}

How much time do I need for this path (edge quantifier)?

\[ G_π = (V,E) \subseteq KVV \quad (G_π \text{ is the subgraph of the line graph}) \]

\[ \text{weight}(G_π) = \sum_{e \in E} g(e) = 12 \quad (\text{Sum of all edge weights}) \]

\[ G_π : \]

```
H  P  2
  |   |
  1  A
  |   |
  2  K
  |   |
  3  E
  |   |
  3  M
  |   |
  1  U
```
• General dynamic system: mechanical, electrical, hydraulic, thermal, chemical systems, among others

• Example of a dynamic system: traveling vehicle, micro-electronic switch, satellite positioning system

• Classic modeling approach
  - Creation of an ideal model of a real system
  - Transformation of the ideal model into mathematical equations, esp. „block diagram“. 

• Disadvantages to the classic modeling approach
  - Elaborate process
  - With new changes to the ideal model, new equations must then be derived
  - Discipline specific approach
    (mechanic, electric, hydraulic …)
The Bond Graph Approach

• Founded by H. Paynter, MIT, MA, 1959
• Recursive top-down disassembly into sub systems that exchange energy with one another
• Interdisciplinary representation of the dynamic behavior of physical systems
• System models are described with a uniform notation for all types of physical systems
• A powerful tool for modeling technical systems, esp. when more than one discipline is involved.
• Contains the information about the physical structure: the structure of a bond graph results from the topologic structure of the schematic representation of the system.
• With changes to the ideal model, only the concerned model parts must be updated: very good for model driven design and „What-If?“ situations
• The Bond graph-approach is an object oriented approach
Where can bond graphs be used?

- Thermodynamic
- Mechanic
- Hydraulic
- Electric
- Magnetism and chemical
Usage Areas of Bond Graphs

- Electric
- Mechanic Translation
- Mechanic Rotation
- Hydraulic/Pneumatic
- Chemical engineering
- Thermodynamic
- Magnetism

\[ I_s = L_{11} \frac{\Delta T}{T} + L_{12} \frac{\Delta S}{T} \]
Types of Sizes in a Bond Graph

- Postulate: for each discipline (mechanic, electric, hydraulic, …) there are 2 variables, whose product is equal to the performance of:
  - Effort and flow
- Hence a uniform, interdisciplinary representation: exploitation of the so-called „classical“ analogy
  - Effort: strength (mech.), tension (el.), pressure (hydr.), entropy flow (therm.)
  - Flow: speed (mech.), current (el.), volume flow (hydr.), temperature (therm.)
- Other analogies are also possible
- Sub-models should satisfy the energy conservation equation.
Types of Junctions in a Bond Graph

- **1-Port junction**
  - C (capacitor): spring (mech.), condensator (el.)
  - I (Inertia): mass (mech.), bobbin (el.)
  - R (Resistor): attenuator (mech.), resistance (el.)
  - $S_e$ (Effort Source): strength (mech.), tension (el.)
  - $S_f$ (Flow Source): source speed (mech.), power (el.)

- **2-Port junction**
  - TF (transformer): lever, gear (mech.), transformer (el.). Transformations within the discipline but also between different disciplines.
  - GY (gyrator): turbine, pump, electric motor. Transformations mostly between different disciplines.

- **3-Port junction**
  - 1-linkage: similar to a serial, electric switch. All flow-variables are equal. The sum of the effort-variables is equal to 0.
  - 0-linkage: similar to a parallel, electric switch. All effort-variables are equal. The sum of the flow-variables is 0.

---

### Data Structures

#### 4. Use of Graphs

**Element** | **Causality**
---|---
R-Element | ![R-Element](image)
I-Element | ![I-Element](image)
C-Element | ![C-Element](image)
Ideal Flow Source | ![Ideal Flow Source](image)
Ideal Effort Source | ![Ideal Effort Source](image)
Transformer | ![Transformer](image)
Gyration | ![Gyration](image)
0-Junction | ![0-Junction](image)
1-Junction | ![1-Junction](image)
Example 1: formation of an electric system in a bond graph

Electric system

Bond graph
Example 2: formation of a mechanic system in a bond graph

Mechanic system (frictionless)

\[ \text{Bond graph} \]

\[ V_3 = V_1 - V_2 \]
Example 3: formation of an interdisciplinary system in a bond graph

**Electric motor**

```
Se → 1 →GY → 1 →TF: D/2
```

- **Se**: Voltage source
- **1**: Constant current source
- **GY**: Gain
- **TF**: Transfer function

**Network**

```
Network
```

**Motor**

```
Motor
```

**Load**

```
Load
```

**Graphical Representation**

- **u_1**, **u_2**, **u_3**: Input signals
- **v_0**: Reference linear velocity
- **v_1**: Output signal
- **w_0**: Reference angular velocity
- **w_1**: Output signal

---

4. Data Structures
4.3 Use of Graphs

**Diagram with Symbols**

- **R**: Resistance
- **I**: Current
- **L**: Inductance
- Bond graph: graph with directed bond
  - vertex: idealized description of subsystems, components, elements of a system
  - Edges (bonds): ideal energy exchange between power ports
  - Half arrow: reference direction of the energy flow

\[ V_3 = V_1 - V_2 \]
Bond Graphs: Summary

- Mapping of a physical system from a data structure graph
- Uniform description of all energy areas, multi-disciplinary systems
- Compact, concise description of complex multi-body systems
- Modeling assumptions, inconsistencies, structure of the mathematical model readable from the bond graph
- The Bond graph (BG) is a generic model form
  - Transferable to other graphical representations (block diagrams, iconic schematics)
  - Deviated from different forms of mathematical model from BG (DAE-system, state room models, Lagrange's-equation, transfer functions)
  - Can be described in object oriented modeling languages, i.e. Modelica