INF5120 – Model-based System Development

Lecture 12: System Architecture and Information/Ontology Modeling

April 15th, 2013
Arne J. Berre, SINTEF ICT
Content

- Lecture plan 2013
- Oblig 2 and 3
- System Architecture
- Cloud computing
- Java 6, JEE
- Conceptual Modeling
- Ontologies
- Information Modeling
- UML Class Models
- ISO 19103
- UML Profiles
# INF5120 - Lecture plan - 2013

1 (14/1): Introduction – overview Enterprise Architecture with UML and BPMN and DSLs
2 (21/1): Service Innovation and Design, AT ONE method/workshop – myServiceFellow (Marika Lüders)
3: (28/1): Value Networks/VDML BPMN, vs. UML Activity diagrams - Oryx
5 (11/2): UML and Req.Modeling –Agile User stories versus Use cases
6 (18/2): Business-SoaML, Requirements Modeling, Goal Modeling, BMM, and Non Functional requirements
7 (25/2): Model driven engineering – Metamodels, DSL, UML Profiles etc. NAV on NFR

9 (11/3): Method Engineering, SW Process frameworks, SPEM/EPF, ISO 24744, FACESEM/ESSENCE (Brian Elvesæter)
10(18/3): Model driven engineering, transformation technologies,
11(8/4): MDE and DSL in practice, with ThingML example – Franck Fleurey
12(15/4): System Architecture and Information/Ontology modeling, UML, ISO 19103
13(22/4): System Architecture and Service models, semantic models
14(29/4): BPMN and Business Process Management and UI models
15(6/5): Platform models for the Cloud, with CloudML, Alessandro Rossini
16(13/5): Conclusion and Summary for INF5120 - Preparation for Exam

Exam: Monday June 3rd, 2013, (4 hours)
INF5120 – Oblig plan - 2013

1 (14/1): Introduction
2 (21/1): myServiceFellow/Service Design
3: (28/1): Oryx
4 (4/2): Balsamiq
5 (11/2): Use cases 2.0
6 (18/2): Oblig 1 – Group work
7 (25/2): EMF and Eclipse – Group (3) presentation – Business Model, Business-SoaML
8 (4/3): Oryx– Group (3) presentation - User stories/use cases, UI, NFR
9 (11/3): Group presentation(s) on Oblig 1: User stories/use cases, UI/Balsamiq
10(18/3): Group presentation on Oblig 1 – NFR 1 – Parallell Start on Oblig 2 (individual) and Oblig 3 (group)

- 22/3 : Delivery of Oblig 1
EASTER

11(8/4): Walk through of Oblig 1 – Questions on Oblig 2 and 3
13(22/4): Group work, Oblig 2/3 – Group presentation/discussion
14(29/4): Group work, Oblig 2/3 – Group presentation /discussion – delivery Oblig 2
11/5: Delivery of Oblig 3
16(13/5): Walk trough of Oblig 3 - Preparation for Exam
Exam: Monday June 3rd, 2013, (4 hours)
Oblig 3 – Concierge Model-based System Architecture

- System Architecture for Concierge
- Basis for Cloud realisation
- Use of Java, JEE 6.x
- MagicDraw Cameo Enterprise Architecture 17.0.3

- Information Model – UML Class model
- Service Model – SoaML model
- Process/Behaviour Model – BPMN/Seq.diagrams, …
- UI Model – (from Balsamiq)
3-tier System Reference architecture

Data Layer:

Application Layer:

Business Layer:

User Service

Service Infrastructure

Legacy

UserService

DataService

UIC

Service

Entity

Business Service Infrastructure

Telecom and Informatics
Work products

Business Architecture
Inception & Elaboration phase focus
Modeling artefacts at different levels of abstraction and from different viewpoints

System Architecture
Elaboration & Construction phase focus
Construction & Transition phase focus

- Requirements Model: System Boundary, Use case scenario, NFR and other requirements
- Service Model: Services Architecture, Service Contract, Service design, Service Detailed design
- Platform Realization Model: Cloud, Web Services, JEE, MAS, P2P/Grid, SWS
Key concepts

- **Process Phases:**
  - **Four phases** are defined (related to UP phases). The key concept is iterations in each phase.

- **Disciplines:**
  - **Five core disciplines** are defined where each has a specific goal and includes some practices to achieve the goal.
    - **Business modelling:** understanding the business and providing information for developing a system that satisfies business goals.
    - **Requirement modelling:** eliciting stakeholders’ requirements on what the system should do and the constraints.
    - **Service modelling:** Developing a service-oriented architecture and related components that fulfil the requirements.
    - **Platform realization:** realization on the specific platform.
    - **Testing:** validation and verification of the system.

- Each discipline includes a set of practices that are performed in one or several tasks. The output are work products (models and others).
Use case driven

- All process components, from requirements capture to test, are driven by use cases/scenarios

- Analyze
- Design and Implementation
- Test

Capture, Clarify and Validate the Use Cases
Realize the Use Cases
Verify that the Use Cases are fulfilled

Use Cases bind these work steps together
Architecture centric (SOA elements)

Code Modules Offered as service with description for
- Discovery
- Utilisation (composition)

(Dirk Krafzig, Karl Banke, and Dirk Slama. Enterprise SOA. Prentice Hall, 2005)
Different architectural views

- **Application architecture**
  - This is the architecture of an application that consumes services and integrates them into business processes in order to provide some service of value to the customers.

- **Services architecture**
  - This view shows services that are available for use through their interfaces. This view hides the implementation from the consumers.

- **Component architecture**
  - This view shows the components implementing the services.
SOA and Cloud Computing

- One can consider cloud computing the extension of SOA out to cloud-delivered resources, such as storage-as-a-service, data-as-a-service, platform-as-a-service -- you get the idea.
- Provide on demand services
- Platform/infrastructure services for running/outsourcing application services
Cloud vs SOA

Some (more and more) include SOA and SaaS as part of the cloud computing concept (the cloud include it all)

Cloud is the enabler
- Platform a service
- Storage as a service
- Computation as a service
- ...
Everything as a service

- Testing-as-a-Service
- Management/Governance-as-a-Service
- Security-as-a-Service
- Integration-as-a-Service
- Software/Application-as-a-Service
- Process-as-a-Service
- Information-as-a-Service
- Database-as-a-Service
- Storage-as-a-Service
- Infrastructure-as-a-Service
- Computation-as-a-Service
- Platform-as-a-Service
- Information-as-a-Service
- Testing-as-a-Service
Different cloud vendors different perspectives

“Servers as a Service”

- Virtual Servers
- Storage/DB as a Service
- Infrastructure as a Service
- Elastic Computing as a service

~Familiar Developer Model
Rapid Scalability

PaaS for the Inquiring Developer

- Google apps
- Gmail, Docs…
- Python App Server
- Storage/DB as a Service
- Infrastructure as a Service
- Elastic Computing as a service

β Offering Innovative Technology

PaaS as an Application Framework

- UI as a Service
- Logic as a Service
- Integration as a Service
- Storage/DB as a Service
- Infrastructure as a Service
- Elastic Computing as a service

Supports Large-Scale SaaS
Deep-Dyed
Java Enterprise Edition (JEE)
References

- **SiSaS Methodology Wiki**
  - [http://sisas.modelbased.net/](http://sisas.modelbased.net/)

- **JEE 6.0**
  - First cup of JEE 6
    - [http://docs.oracle.com/javaee/6/firstcup/doc/](http://docs.oracle.com/javaee/6/firstcup/doc/)
  - Official JEE 6 tutorial
    - [http://docs.oracle.com/javaee/6/tutorial/doc/](http://docs.oracle.com/javaee/6/tutorial/doc/)
JEE: Java Enterprise Edition

- Java for 3 tiers architectures

- Includes
  - Java Libraries (Mail, Protocols, etc.)
  - Tools, compilers, code generators,
  - Application Server

- JEE 6.x
Java Server Faces

- **Presentation (XHTML):**

```html
<html>
<body>
<f:view>
<h:form formName="loginForm">
<h:panelGrid columns="3">
<h:outputText value="Login"/>
<h:outputLabel value="Login"/>
<h:inputText value="#{login.login}"/>
<h:outputLabel value="Password:"/>
<h:inputSecret value="#{login.password}"/>
<h:commandButton type="reset" value="Reset"/>
<h:commandButton type="submit" action="#{login.doLogin}"/>
</h:panelGrid>
</h:form>
</f:view>
</body>
</html>
```

- **Controller (Java):**

```java
@ManagedBean(name=login)
@SessionScoped
public class LoginBean {

@EJB
private LoginService delegate;

private String login;
private String password;

public String getLogin() { … }
public void setLogin(String login){ … }

public String getPassword() { … }
public void setPassword(String password) { … }

public boolean doLogin(String login, String password) {
    delegate.login(login, password);
}
}
```
Web Services (With JEE 3.x)

- Expose methods on a network
  - XML/SOAP
  - HTTP

- WSDL descriptors
  - Generated by A.S.
  - From annotations

```java
import javax.jws.WebService;
import javax.jws.WebMethod;

@WebService
public class Hello {
    private String message = new String("Hello, ");

    public void Hello() {
    }

    @WebMethod
    public String sayHello(String name) {
        return message + name + ".";
    }
}
```
Persistence (With JEE 6.x)

- Store persistent data
- How to transform objects into tables?
- Mapping generated from annotation
- What attribute are ID, Associations realizations

```sql
create table Customers (  
id integer,  
firstname varchar(100),  
lastname varchar(100),  
email varchar(100) unique,  
constraint primary key (id) );

create table Accounts (  
id integer,  
owner integer,  
constraint primary key (id),  
constraint foreign key owner references Customer(id) );

create table Transactions (  
id integer,  
description varchar(50),  
amount double,  
constraint primary key (id),  
constraint foreign key source references Account(id),  
constraint foreign key target references Account(id) );
```
Model based system development

- What does MBSD brings?
  - Capitalize on key development processes
  - Automation
    - Model transformations

![Diagram showing MDE Tools, Engine, Model Transformations, Java Code, and Cloud with deployed text.]
Internet Bank Example

- 3-Tiers Architecture
  - Presentation
    - JSF + web server
  - Business
    - WS + Application Server
  - Persistence
    - EJB3.0 + Application Server

Diagram:
- Client-side:
  - Browser
  - Bank
    - Customer
  - Bank
    - Agent

- Server-side:
  - Java Server Faces (JSF)
  - Presentation
  - Business Services
  - Persistence
  - MySQL
  - EJB3.0 + Application Server
IaaS: Deploying a Glassfish server

- AWS-EC2: Amazon Infrastructure Service
  - Provides virtual Machine
  - With SSH connection
- Turn-key VM
- Nothing installed!
  - JRE
  - Database
  - Glassfish
IaaS: Deploying a Glassfish server

- Glassfish
  - Provide services to deploy services/apps remotely
Cordys Business Operations Platform

Cordys Cloud™
Platform-as-a-Service for enterprises, service providers and ISVs

- Complete BPM & middleware platform in the cloud
- Enables enterprise mobility & multi-channel delivery
- Managed as a cloud service by Cordys

Read More...
OMG Modeling languages and Zachman Framework

Scope (Contexts)
Business (Concepts)
System (Logic)
Technology (Physics)
Component (Assemblies)
Operation (Instances)

PIM-K

Technology/Realisation-K

Information   Process   PIM-K   Services   Rules   NFA   UI

Data   Wflow/Comp   PSM-K   Interfaces   Rules   NFA   UI

XML, OWL, BPEL/XPDL, WSDL, SWRL, Security, AJAX

Legacy and New systems/services, ERPs/ESAs

BPDM, SBVR, EDOC, UPMS, PIM4SOA, ODM

UML profiles and metamodels for BPEL, WSDL, XML, XPDL, OWL-S, WSML, WSDL-S

BPMN, POP*, ARIS, ArchiMate, GERAM, GRAI, Zachman, UEMI, B.Rules..
Information modeling and Ontology Engineering – Semantic technologies

- Conceptual Modeling – ISO TR 9001
- ER Modeling, NIAM Modeling, EXPRESS Modeling
- UML Modeling
- ISO 19103 – Conceptual Schema with UML
- Semantic Web Modeling
- Ontology Engineering
- Logic Formalisms – First order logic
- XML Schema, RDF, Topic Maps, OWL
- ODM - Ontology Definition Metamodel
- Open Linked Data, Open Linked Services
Conceptual Modeling

- Concepts
  - Conceptual model
  - Conceptual schema language
  - Conceptual schema

- Why conceptual modeling in 19100?
  - to provide a rigorous definition of geographic information and geographic information services
  - to standardize the definition of conceptual schemas
From reality to conceptual schema

- Real World
  - Universe of Discourse
    - defined in
- Conceptual formalism
  - provides concepts for describing
  - basis for one or more
- Conceptual Schema Language(s)
  -Lexical Languages
  - Graphical Languages
- Conceptual Model
  - formally represented in
- Conceptual Schema
  - provides formal language for representing
Principles

1) ISO 100% Principle
   - static aspects
   - dynamic aspects

2) Conceptualisation principle
   - exclude physical representation

3) Helsinki principle
   - agree set of rules

4) Concrete syntax
   - UML

5) Self description

Conceptualisation principle
- exclude physical representation

Basis for

Information System

Real World
- Problem domain

Universe of Discourse

Conceptual Model

Information modeling

Data modeling

Solution domain
What is relevant?

Information, Rules, constraints

Behaviour/Dynamics – *services, process*

ISO 100% Principle
- static aspects
- dynamic aspects

Universe of Discourse

Real World - Problem domain

Conceptual Model

Basis for Information System

Information modeling

Data modeling

Solution domain
Different kind of models

- Conceptual models
- Specification models
- Implementation models
Information modeling

Semantic Models

UML Activity model (or BPMN)

UML component diagram (enhanced in UML 2.0), SoaML

UML Class diagram
Parts of UML Metamodel

- GeneralizableElement
  - Association
    - AssociationEnd
      - 0..* (with 2..n label)
  - Classifier
    - +features
      - 0..* (with 1 label)
      - 0..* (with 1 label)
      - 0..* (with 1 label)
      - 0..* (with 1 label)
    - Data Type
    - Class
    - Structural Feature
    - Behavioural Feature
      - Attribute
        - initialValue : Expression
      - Operation
        - Specification
          - isPolymorphic : Boolean
          - concurrency : CallConcurrencyKind
Different kind of models

Idea
- High-level models: guide the thought process
- Abstract specifications of essential structure: focus on semantic intent
- Full implementation specifications: semantics, algorithms, data structures, performance measures, ...

System
Information Modeling Guidelines

- Process steps
- Modeling phases
- Modeling harmonization
Process Steps

A) Problem definition - Enterprise (Analysis)
   1) Identify Area of Concern/Problem definition
   2) Goals (Use cases)
   3) Interaction scenarios (sequence / collaboration diagrams)
   4) High level domain model / concepts (Class diagrams)

B) Domain model - Information

C) Architectural Model - Services

D) Implementation models

E) Implementation

F) Maintenance
Information Modeling

Phase 0: Identify scope and context

Phase 1: Identify basic entities

Phase 1b: Is the modeling approach consistent with the approach described in the rules for application schema document

Phase 2: Specify relationships, attributes and possibly operations

Phase 3: Completion of constraints

Phase 4: Model definition harmonisation – with sub-models and other work items
P1 Identify basic entities

- Refine objects/classes from enterprise viewpoint

Tasks
- 1) Identify generic and specific concepts
- 2) describe each of the entities
- 3) Watch out for synonyms and homonyms
- 4) Create a data dictionary (glossary of terms)

Questions
- Are entities according to scope?
- Can we categorize entities’
- What are the attributes and constraints?
- 1b) Consistent with rules for application schemas?
P2 Specify relationships, attributes and operations

- The entities of interest - corresponding to units of information
- Any generalization relationships between entities
- Other relationships between entities
- Check if the existence of an entity depends on the existence of another entity
- What consistency rules need to be satisfied in order for an instance of the entity, or model as a whole, to be valid.
- The semantics, if any, of the types used for attributes
- Which attributes uniquely identify an entity
- The attributes of the entities and whether any of them can be derived from other attributes
- Homonyms and synonyms - create a glossary of terms
Before harmonization

- Review your work critically
  - RATIONALIZE
- Remove redundancy:
  - Ensure that each relationship is only specified once.
  - Ensure that each constraint is captured only once, and that multiple constraints are separated.
  - Ensure that if one entity is dependent on another, then the relationship is specified from the dependent to the independent.

---

**Checklist**

1) Readability
2) Scope
3) nym principle
4) context independence
5) implementation independence
6) invariance
7) constraints
8) reality
9) redundancy
10) concepts
11) hierarchies
12) simple types
Model harmonization

- Are there redundancies or conflicts?
- Are there any ambiguities?
- Are the models complete?
- Do the abstraction levels match?
More semantics using ontologies

- Models can be annotated
  - Creating a connection between the models and a formal ontology
- The annotations can be used in order to "automate" mapping definitions
  - Not complete automation
  - Can be used for aiding mapping designer through tool support
- The reasoning power from the world of ontologies comes into play here
  - Query the ontology system for things like
    - "Give me a list of elements in Model B that are annotated with similar ontology concepts as element X in Model A"
Ontologies

- Ontologies provide a shared understanding of a domain
- They provide background knowledge to systems to automatize certain tasks
- By the process of annotation, knowledge can be linked to ontologies
  - Example: “Angelina Jolie” (Text) linked to concept Actress
  - In our ontology we also know that an actress always is female and a person
- Ontologies allow the creation of annotations → machine-readable and machine-understandable content
- If machines can understand content, they can also perform more meaningful and intelligent queries
  - Distinction of Jaguar the animal and the car
  - Combination of information that is distributed on the Web
Definitions

- An ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations to define extensions to the vocabulary.
- An ontology is a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base.
- An ontology provides the means for describing explicitly the conceptualization behind the knowledge represented in a knowledge base.
- An ontology is a formal, explicit specification of a shared conceptualization.
Examples

- OpenCyc.org
- eClass
- OpenGALEN
- W3C
- WordNet
- UNSPSC
- OWL
- SUMO
- Creative Commons
- OBO
- National Library of Medicine
- United States National Institutes of Health
- Wonder
- Web
Features of an ontology

- Modelled knowledge about a specific domain
- Defines
  - A common vocabulary
  - The meaning of terms
  - How terms are interrelated
- Consists of
  - Conceptualization and implementation
- Contains
  - Ontological primitives
Classifications of ontologies

Languages for building ontologies

- Ontologies can be built using various languages with various degrees of formality
  - Natural language
  - UML
  - ER
  - OWL/RDFS
  - WSML
  - FOL
  - ...
- The formalism and the language limit the kind of knowledge that can be represented
- A domain model is not necessarily a formal ontology only because it is written in OWL
Applications of ontologies

- Knowledge representation
  - Ontology models domain knowledge

- Semantic annotation
  - Ontology is used as a vocabulary, classification or indexing schema for a collection of items

- Semantic search
  - Ontology is used as a query vocabulary or for query rewriting purposes

- Configuration
  - Ontology defines correct configuration templates
Principles for the design of ontologies

- Clarity
  - To communicate the intended meaning of defined terms
- Coherence
  - To sanction inferences that are consistent with definitions
- Extendibility
  - To anticipate the use of the shared vocabulary
- Minimal Encoding Bias
  - To be independent of the symbolic level
- Minimal Ontological Commitments
  - To make as few claims as possible about the world

Ontology engineering

The set of activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies.

Ontology life cycle

- Domain analysis: motivating scenarios, competency questions, existing solutions
- Conceptualization: conceptualization of the model, integration and extension of existing solutions
- Implementation: implementation of the formal model in a representation language
- Usage/Maintenance: adaptation of the ontology according to new requirements

Project management: controlling, planning, quality assurance etc.
Ontology engineering activities

**Management**
- Scheduling
- Control
- Quality assurance

**Development oriented**
- Pre-development
- Environment study
- Feasibility study
- Development
  - Specification
  - Conceptualization
  - Formalization
  - Implementation
- Post-development
  - Maintenance
  - Use

**Support**
- Knowledge acquisition
- Evaluation
- Integration
- Documentation
- Merging
- Configuration management
- Alignment
Methodologies for building ontologies

- Validated guidelines on how the ontology building process should be structured
- Well-organized process instead of „ontology development driven by inspiration and intuition only“
- Methodologies do not support all of the aforementioned activities
- They implicitly assume a particular development paradigm for the ontology engineering process
- Some of them also provide supporting methods and tools
- There are many different methodologies for building ontologies
How to build an ontology?


- **Step 1: Determine the domain and scope of the ontology**
  - What is the domain that the ontology will cover?
  - For what we are going to use the ontology?
  - For what types of questions the information in the ontology should provide answers?
  - Who will use and maintain the ontology?

- **Competency Questions**
  - A set of queries which place demands on the underlying ontology
  - Ontology must be able to represent the questions using its terminology and the answers based on the axioms
  - Ideally, in a staged manner, where consequent questions require the input from the preceding ones
  - A rationale for each competency question should be given
How to build an ontology? (cont’)


- Step 2: Consider reusing existing ontologies
  - Reuse ensures interoperability and reduces costs
  - Ontology libraries and tools for customization are required for this step
  - Sub-steps
    - Discover potential reuse candidates
    - Evaluate their usability
    - Customize ontologies to be reused
    - Integrate and merge to the target ontology

- Step 3: Enumerate important terms in the ontology
  - What are the terms we would like to talk about?
  - What properties do those terms have?
  - What would we like to say about those terms?
How to build an ontology? (cont’)


Step 4: Define classes and class hierarchy

- From the list created in Step 3, select the terms that describe objects having independent existence rather than terms that describe these objects
  - These terms will be classes in the ontology
- Organize the classes into a hierarchical taxonomy by asking if by being an instance of one class, the object will necessarily (i.e., by definition) be an instance of some other class
  - If a class A is a superclass of class B, then every instance of B is also an instance of A
- Classes as unary predicates—questions that have one argument. For example, “Is this object a wine?”
- Later: binary predicates (or slots)—questions that have two arguments. For example, “Is the flavor of this object strong?” “What is the flavor of this object?”
How to build an ontology? (cont’)

Step 5: Define attributes and relationships

- Step 4 selected classes from the list of terms we created in Step 3
  - Most of the remaining terms are likely to be properties of these classes
  - For each property in the list, we must determine which class it describes

- Types of properties
  - “intrinsic” properties
  - “extrinsic” properties
  - parts, if the object is structured (physical or abstract)
  - relationships to other individuals

- Properties are inherited and should be attached to the most general class in the hierarchy
MagicDraw – Models with code generation sets
UML models
Quick guided tour of the language

- General mechanisms
  - Packages
  - Classes
- UML profiles
  - Stereotypes
  - Tagged values
  - Notes
  - Constraints

- Basic diagrams
  - Class diagrams
  - Object diagrams
  - Use case diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Statechart diagrams
  - Activity diagrams
  - Component diagrams
  - Deployment diagrams
UML Information Modeling

- Ref also ISO 19103 Standard for Conceptual Modeling
- The following material is for reference ….
Packages

- Models structuring
Class diagrams

- Class diagrams represent a set of classes, interfaces, collaborations, and their relationships.
- They are the most frequent diagrams in the objects systems modeling.
- They offer the static conceptual view of a system.
- Class diagrams, (including active classes), offer the static process view of a system.
A class is an abstract description of a set of objects.

A class can be seen as a factorization of common elements to a set of objects.

A class describes the definition domain of a set of objects.

Description conceptually split in two parts:

- A class specification which defines the definition domain and the properties of the class instances (datatype).
- A realization defines how a specification is realized.

Classes features:

- Attributes
- Operations
- Réceptions
- Relations
- Multiplicity
- Persistence
- Component
Objects

Can represents

- One instance
  - Ola : Person

- One type, interface
  - <<Interface>>
    - Person

- One class
  - Person
# Object and classes
- notation

<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
</table>
| navn : string  
personnr. : integer  
adresse : string  
gasje : money  
stillingstittel : string |

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
| endre-stilling()  
endre-adresse() |

**Example - object class**
## Objektklasser - notasjon

<table>
<thead>
<tr>
<th>Klassenavn</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributtnavn-1 : datatype-1 = defaultverdi-1</td>
</tr>
<tr>
<td>attributtnavn-2 : datatype-2 = defaultverdi-2</td>
</tr>
<tr>
<td>........</td>
</tr>
<tr>
<td>operasjonsnavn-1 (argumentliste-1) : resultattype-1</td>
</tr>
<tr>
<td>operasjonsnavn-2 (argumentliste-2) : resultattype-2</td>
</tr>
<tr>
<td>........</td>
</tr>
</tbody>
</table>
Class diagram

```
GM_Object
+ mbRegion : GM_Object
+ representativePoint : DirectPosition

+ SRS() : SpatialReferenceSystem
+ transform(SRS : SpatialReferenceSystem) : GM_Object
+ equals(object : GM_Object) : Boolean
+ distance(object : GM_Object) : Distance
+ dimension() : Integer
+ dimension(point : DirectPosition) : Integer
+ coordinateDimension() : Integer
+ envelope() : Envelop
+ boundary() : Set(GM_Object)
+ buffer(radius : Distance) : GM_Object
+ convexHull() : GM_Object
+ centroid() : DirectPosition
+ representativePoint() : GM_Point
+ isInComplexes() : Set(GM_Complex)
+ isPartOf(geomCplx : GM_Complex) : Boolean
+ universe() : GM_Complex
```

```
GM_Primitive
(from GeomPrimitive)
+ boundary() : Set(GM_Primitive)

GM_Complex
(from GeomComplex)
```
## Class attributes

<table>
<thead>
<tr>
<th>ClassName</th>
</tr>
</thead>
</table>

// /* derived attribute

+ /* public visibility

# /* protected visibility

- /* private visibility

underline /* class scope
Class attributes

[visibility] name [multiplicity] [:type] [= initial value] [{property-string}]

+ origin [0..1] : Point = (0,0) {frozen}

*defined properties:* changeable, addOnly, frozen (const)
Attributes and data types

- A data type specifies a legal value domain and the operations on values of that domain

- Four categories
  - a) Basic data types (*integer*, *real*, *string*, …)
  - b) Collection data types (from OCL)
  - c) Enumerated data types (user-definable finite sets)
  - d) Model types
Operations

An operation is a specification of
• a transformation, or
• a query

A method is a procedure that implements an operation.
Operations

[visibility] name [(parameter-list)] [:return-type] [{property-string}]

[(parameter-list element)] ::= 
[direction] name : type [= default-value]
[direction] ::= in | out | inout

+ set ( in name : Name, in place : String = ‘Oslo’) :
  Boolean {concurrency=sequential}

defined properties: isQuery, concurrency, ...

defined properties: isQuery, concurrency, ...
Class diagrams

<table>
<thead>
<tr>
<th>class name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>- private attribute</td>
</tr>
<tr>
<td>+ public attribute</td>
</tr>
<tr>
<td># protected attribute</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>- private operation</td>
</tr>
<tr>
<td>+ public operation</td>
</tr>
<tr>
<td># protected operation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>class name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>attributes</td>
</tr>
<tr>
<td>attr_name : type = initial value</td>
</tr>
<tr>
<td>operations</td>
</tr>
<tr>
<td>op_name (arg_list):type</td>
</tr>
</tbody>
</table>

parameter

parameterized class

association name

a class

another class

association class name

attributes

operations

/ derived association

role r1

association name

role 2

a class

another class
## Properties visibility

- **Public +**
  Accessible to all objects within the system

- **Protected #**
  Accessible to instances of implementing class and its subclasses

- **Private -**
  Accessible to instances of the implementing class

- **Underlined**
  Class variable/operation

### Class variable/operation

<table>
<thead>
<tr>
<th>Thing</th>
</tr>
</thead>
<tbody>
<tr>
<td>+public_attribute</td>
</tr>
<tr>
<td>#protected_attribute</td>
</tr>
<tr>
<td>-private_attribute</td>
</tr>
<tr>
<td>+public_operation()</td>
</tr>
<tr>
<td>#protected_operation()</td>
</tr>
<tr>
<td>-private_operation()</td>
</tr>
</tbody>
</table>

### Window

```c
{abstract, author=Joe, status=tested}
```

```c
+size: Area = (100,100)
#visibility: Boolean = true
+default-size: Rectangle
#maximum-size: Rectangle
-xptr: XWindow*
```

```c
+display ()
+hide ()
+create ()
-attachXWindow(xwin:Xwindow*)
```
Relations between classes

- Association
- Aggregation
- Composition
- Generalization
- Dependency

- An association expresses a bi-directional semantic connection between classes.
- An association is an abstraction of the links existing between the instances objects of the associated classes.
Optional/Conditional

- In UML all attributes and operations are per default mandatory.
- Optional data values for attributes can be shown through multiplicity [0..1].
- Conditional should relate to a note with constraint expressed in text/OCL (ISO 19103)
Relationships

---

**Association (the “glue”)**
A semantic relationship between elements
(involves connections among their instances)

---

**Generalization (inheritance)**
A relationship between an element
and the sub-elements that may be substituted
for it

---

**Dependency**
The use of one element by another

---

**Refinement/realisation**
A shift in levels of abstraction
Different relationships in UML Diagrams

Purpose: To show relationships between model entities
To define multi-way constraints

Multiplicity of an Association

Generalization

Dependency (Client-Server)

Association Notation used to “anchor” at note to a model entity

Aggregation

Composition (Strong Aggregation)

Link Attribute
Generalisation and Association

Superclass

Person

0..1 loans

Book

0..*

Loan-entity

Loan customer

Association

Generalisation

Man

hUSBAND

0..1

Married with

{not siblings}

wife

Woman

0..1

Subclasses (inherits attributes and operations)

{not siblings}
Example
Associations naming

- Indicating the reading direction

- University Welcomes Student

- University Studies in Student
Roles naming

- A role describes an association end

Diagram:
- University
- +Student
- Person
- +Employer
- +Teacher
Roles multiplicity

1 One only
0..1 Zero or one
M .. N From M to N (natural numbers)
* Several
0 .. * Zero to several
1 .. * One to several
Roles visibility

- Public
- Protected
- Private

Diagram:

- Public role
- Protected role
- Private role

Nodes:
- A
- B
- C

Connections:
- A to B: +Public role
- A to C: #Protected role
- B to C: -Private role
Navigability

- Given an unlabelled association between two classes, it is possible to navigate from objects of one kind to objects of the other kind.
- By default an association is bi-directional.
- A navigability indication suggests that an object at one end is directly and easily reachable from an object at the other end.
- Putting it into practice, for instance a source object memorizes a direct reference to the target objects.

- Given a user, it is desirable to have access to his / her passwords.
- Given a password, it is not desirable to have access to the corresponding user.
Example
Aggregation

- Antisymmetric bi-directional semantic connections
- An association expressing a stronger coupling between classes

- Relationship representation
  - master and slave
  - whole and parts
  - composed and composing
Composition

- Physical composition modeling
  - Maximum multiplicity of 1 on the aggregate side
  - Automatic propagation of the destruction
Classes hierarchy

- Managing the complexity
- Class tree of increasing abstraction

![Diagram of class hierarchy]

- Super-class
- More general class
- Sub-class
- More specific class

Personne:
- nom : String = noName
- age : Integer = 15
- numSS : Integer

Etudiant:
- numEtudiant : Integer

Enseignant:
- salaire : Integer

Moniteur:
Multippel arving - eksempel

Multiplicity shows how many instances of a class that can be related to one instance of the class at the other end of the association.
Aggregation

Loose (weak) part-of relationship

Multiplicity > 1 possible

Exists even if a phone book goes away

Open diamond

association role names
Composition
("strong aggregation")
Strong part-of relationship

File

1 contains

Record

Filled diamond

Multiplicity max 1

Existence dependent
Associations

Fig. 3-31, UML Notation Guide
Association Ends

Fig. 3-32, UML Notation Guide
Association class

Are used instead of link-attributes if
- The association are related to other objects
- Operations are attached to the association

![Diagram of association class with entities Person, Loans, Book, Loan, Loan date, Delivery data, CheckDate(), and Expired loans]
Constraints on relationships

- Portfolio
  - BankAccount
    - {secure}
  - {or}
    - Corporation
    - Person
- Department
  - {subset}
  - member
    - Person
  - manager
    - Person

1..*}
Packages in UML

- This is a grouping of model elements and diagrams.
- Package dependencies usually summarize dependencies among the contents of the packages.
- Packages can contain other Packages.
- Packages can show the Class/Entities found in a given Package.

Geometry Package
(from Logical View)
+ Accuracy
+ TopologicalGeometry
+ RepresentationalGeometry
+ TemporalGeometry
+ Geometry
+ Vector{dimension}
+ SpatialVector
Containment and dependency

- Behavioral Elements
- Model Management
- Foundation
- Core
- Extension Mechanisms
- Data Types
Common Mechanisms

- Adornments: notes
- Extensibility mechanisms: stereotypes, tagged values, constraints

Notes:

- <<requirement>>
- Shall conform to ....
- Comments and constraints
Stereotypes

- Used to define derivative modeling concepts based on existing generic modeling concepts

- Defined by:
  - base (meta-)class = UML meta-class or stereotype
  - constraints
  - required tags (0..*)
    - often used for modeling pseudo-attributes
  - icon

- A model element can have at most one stereotype
Tagged values - properties

Name (tag) separator (=) value (of the tag)

properties on an element - relevant for code generation or configuration management (Can be applied to all UML elements)

Server
{processors = 3}

Billing
{version=3.2
status=checkedOut
by=ajb}
OCL - Object Constraint Language

*First order predicate logic, for Boolean expressions - included in UML 1.1*

Can be used for:
- invariants,
- value restrictions,
- pre- and postconditions

Expressions with:
- and, or, not, implies, exists, for all,
- Collections (select, reject, collect, iterate)

**Person**
- `self.age > 0`

**Married people are of age >= 18**
- `self.wife->notEmpty implies self.wife.age >= 18` and
- `self.husband->notEmpty implies self.husband.age >= 18` and
UML profiles
UML profiles

- They allow us to adapt the UML language to the needs of the analysts or the application domain.
- Allows designers to model using application domain concepts.
- There are three extension mechanisms:
  - Stereotypes
  - Tagged values
  - Constraints
UML for a specific purpose (benefits, = a specific profile

Profiles exist for:
- technical targets (C++, RDB, Java, CORBA, XML, Real Time, etc.)
- specific disciplines (Analysis, Design, etc.)
- requirement analysis
- tests (test for Java, test for EJB, test for C++, etc.)
- metrics, quality checking, and profiles for managing configuration management rules.

From early requirement down to final tests, UML profiles can drive each specific discipline involved.
Stereotype

- Extends the vocabulary of UML with **new construction elements** derived from **existing** UML but specific to a problem domain
- Can have associated **tagged values and constraints**
- Possibility of assigning an **icon** for a better graphical representation
Examples of other stereotypes

- «datatype»
  - Integer

- «utility»
  - Maths

- «subsystem»
  - User Interface

- «exception»
  - Overflow

- «traces»
- «refines»
Tagged value

- Is a **property** associated to a **model element**
- Used to store **information** about the element
  - Management information, documentation, coding parameters, ...
- Generally, the **tools** store this information but **it is not shown in the diagrams**
Constraints

- Any semantic relationship between model elements
- Expressed in OCL (Object Constraint Language) or in natural language
  - \{constraint\}, inv, pre-, post-condition

```
A

«postcondition»
{Count > 10}
```
Profile: Example (Usage of UML extensions)

- Below is a user model example, with extensions. That example supports extensions.
- We will see how they are defined at the profile definition level.

```plaintext
<<persistent>>
Customer
{storageMode = hash}

<<identifier>> number : int

<<identifier>>
```

**Diagram:**
- **PROFILE**
- **Stereotype**
- **Tagged Value**
- **Operation**
- **UML**
  - **Class**
  - **Attribute**

**Toolbox:**
- Edit
- Check
- Generate SQL

...
Extending Metaclasses

- A profile consists primarily of stereotypes
  - restricted metaclasses that can only be used to extend existing metaclasses
Applying a profile makes the profile extension available to the model packages to which it is applied.
ISO 19103 – Conceptual modeling with UML - a UML profile for Information modeling

*Implementation neutral! Not implementation specific!*

- Basic data types
- Stereotypes
- Naming
- Documentation of models
- Information modelling guidelines
Basic Types

- **Primitive**
  - Decimal, Integer, Number, Real, Vector, Character, CharacterString, Date, Time, DateTime, Binary
  - Sign, Boolean, Logical, Probability

- **Collections**
  - Set<T>, Bag<T>, Sequence<T>, Dictionary<K,T>

- **Measures**
  - Angle, Area, Distance, length, Scale, MTime, Velocity, Volume, UnitOfMeasure

- **Records, Namespace**
Number

<<BasicType>>

Number

<<BasicType>>

Integer

<<BasicType>>

Real
User defined data types

- **<<DataType>>**
  - DirectPosition
    - coordinate : Sequence<Number>
    - dimension : Integer

- **<<Enumeration>>**
  - Sign
    - positive
    - negative

- **<<CodeList>>**
  - MD_Restrictions1
    - copyright = 1
    - patent = 2
    - patentPending = 3
    - trademark = 4
    - licence = 5
    - intellectualPropertyRights = 6
    - restricted = 7
    - otherRestrictions = 8

- **<<CodeList>>**
  - MD_Restrictions2
    - 1 copyright
    - 2 patent
    - 3 patentPending
    - 4 trademark
    - 5 licence
    - 6 intellectualPropertyRights
    - 7 restricted
    - 8 otherRestrictions

- **<<Union>>**
  - GM_Position
    - direct : DirectPosition
    - indirect : GM_PointRef
Class Stereotypes

- NONE

- <<Interface>>

- <<Type>>

- <<Abstract>>

- <<MetaClass>>
Class (package, association) names

- All classes must have unique names.
- Names shall start with upper case letter.
- Should not have a name that is based on its external usage, since this may limit reuse.
- Should not contain spaces
  - Separate words in a class should be concatenated, e.g. “XnnnYmmmm”
- Class names should start with bialpha prefixes for each standard part.
  
  GM_CurveSegment
  MD_Citation
  GM_FeatureAttribute
Attribute, operation and role names

- Shall start with lower case letter
- Concatenate words shall begin with capital letter
- Do not repeat class names in attribute names
- Keep names technical, meaningful and short, if possible

computePartialDerivates
compute Partial Derivates
compute_partial_derivatives
Associations

- Multiplicity shall be defined for both associations ends
- All associations ends (roles) representing the direction of a relationship must be named or else the association itself must be named
- The role name must be unique within the context of a class and its supertypes.
- The direction of an association must be specified
Documentation of models

- Diagrams
  - Package dependency diagrams
  - Class diagrams
  - Class context diagrams

- Text
  - Class
    - General description, semantics, supertypes/subtypes
  - Attributes, Associations
  - Operations
    - preconditions, input/output parameters, return value, post conditions, exceptions, constraints
  - Constraints

NB! Font size > 8pt