Software Design, Modelling and Analysis in UML

Lecture 22: Meta-Modelling

2015-02-10

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- Inheritance in UML: concrete syntax
- Liskov Substitution Principle — desired semantics

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What's the Liskov Substitution Principle?
  - What is late/early binding?
  - What is the subset, what the uplink semantics of inheritance?
  - What's the effect of inheritance on LSCs, State Machines, System States?
  - What's the idea of Meta-Modelling?
  - How to read the OMG UML standard documents
- **Content:**
  - The UML Meta Model
  - Wrapup & Questions

Meta-Modelling: Idea and Example

Meta-Modelling: Why and What

- **Meta-Modelling** is one major prerequisite for understanding
  - the standard documents [OMG, 2007a, OMG, 2007b], and
  - the MDA ideas of the OMG.
- The idea is **simple**:
  - if a **modelling language** is about modelling **things**,
  - and if UML models are and comprise **things**,
  - then why not **model** those in a modelling language?
- In other words:

  Why not have a model $\mathcal{M}_U$ such that
  - the set of legal instances of $\mathcal{M}_U$

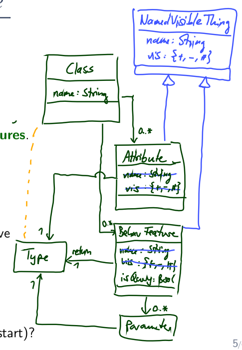
  is
  - the set of well-formed (!) UML models.

Meta-Modelling: Example

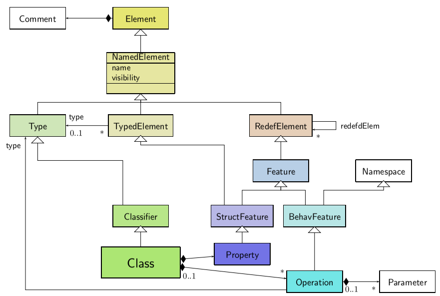
- For example, let's consider a class.
- A **class** has (on a superficial level)
  - a **name**,
  - any number of **attributes**,
  - any number of **behavioural features**.

  Each of the latter two has
  - a **name** and
  - a **visibility**.

  Behavioural features in addition have
  - a boolean attribute **isQuery**,
  - any number of parameters,
  - a return type.
- Can we model this (in UML, for a start)?

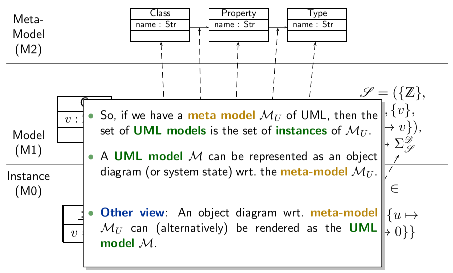


UML Meta-Model: Extract from UML 2.0 Standard



Meta-Modelling: Principle

Modelling vs. Meta-Modelling



- So, if we have a **meta model** $\mathcal{M}_U$ of UML, then the set of **UML models** is the set of **instances** of $\mathcal{M}_U$.
- A **UML model** $\mathcal{M}$ can be represented as an object diagram (or system state) wrt. the **meta-model** $\mathcal{M}_U$.
- **Other view**: An object diagram wrt. **meta-model** $\mathcal{M}_U$ can (alternatively) be rendered as the **UML model** $\mathcal{M}$.

Well-Formedness as Constraints in the Meta-Model

- The set of **well-formed UML models** can be defined as the set of object diagrams satisfying all constraints of the **meta-model**.

  For example,

  > "[2] Generalization hierarchies must be directed and acyclical. A classifier cannot be both a transitively general and transitively specific classifier of the same classifier.
  >
  > not self . allParents() -> includes(self)" [OMG, 2007b, 53]

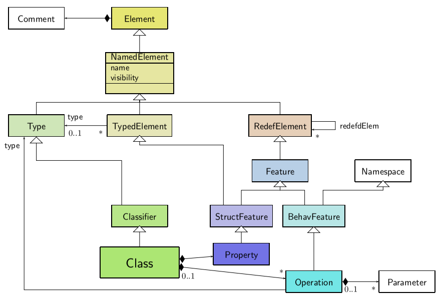
- The other way round:

  Given a **UML model** $\mathcal{M}$, unfold it into an object diagram O_1 wrt. $\mathcal{M}_U$. If O_1 is a **valid** object diagram of $\mathcal{M}_U$ (i.e. satisfies all invariants from $Inv(\mathcal{M}_U)$), then $\mathcal{M}$ is a well-formed UML model.

  That is, if we have an object diagram **validity checker** for of the meta-modelling language, then we have a **well-formedness checker** for UML models.

The UML 2.x Standard Revisited

Claim: Extract from UML 2.0 Standard



Classes [OMG, 2007b, 32]

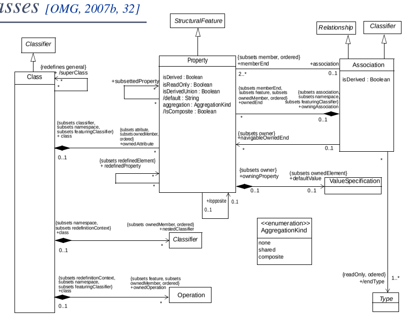


Figure 7.12 - Classes diagram of the Kernel package

Operations [OMG, 2007b, 31]

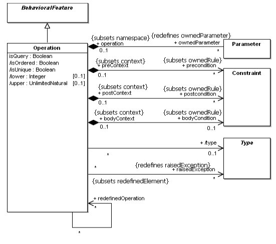


Figure 7.11 - Operations diagram of the Kernel package

Operations [OMG, 2007b, 30]

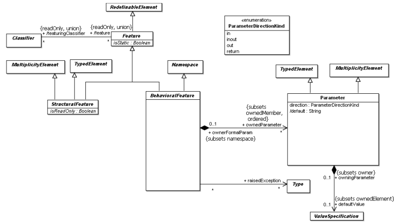


Figure 7.10 - Features diagram of the Kernel package

Classifiers [OMG, 2007b, 29]

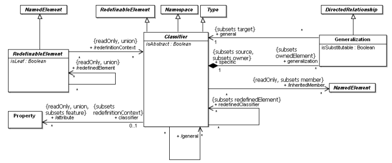


Figure 7.9 - Classifiers diagram of the Kernel package

Namespaces [OMG, 2007b, 26]

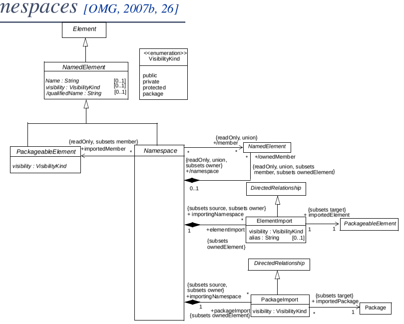


Figure 7.4 - Namespaces diagram of the Kernel package

Root Diagram [OMG, 2007b, 25]

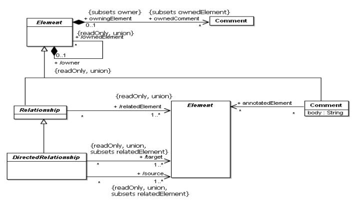


Figure 7.3 - Root diagram of the Kernel package

Interesting: Declaration/Definition [OMG, 2007b, 424]

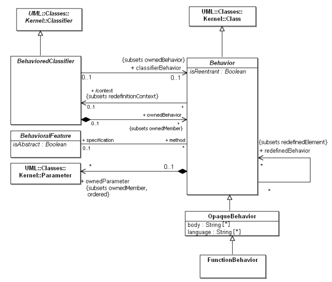


Figure 13.6 - Common Behavior

UML Architecture [OMG, 2003, 8]

- Meta-modelling has already been used for UML 1.x.
- For UML 2.0, the request for proposals (RFP) asked for a separation of concerns: **Infrastructure** and **Superstructure**.
- **One reason**: sharing with MOF (see later) and, e.g., CWM.

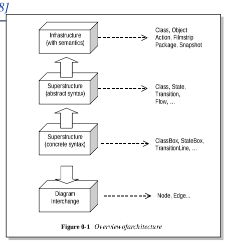
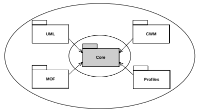


Figure 9-1 Overview of architecture



UML Superstructure Packages [OMG, 2007a, 15]

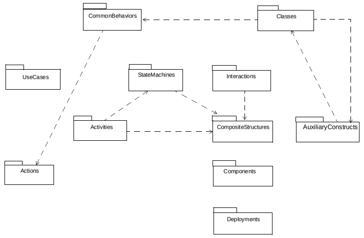


Figure 7.5 - The top-level package structure of the UML 2.1.1 Superstructure

Reading the Standard

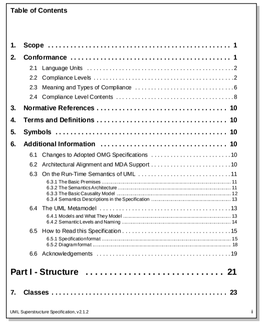
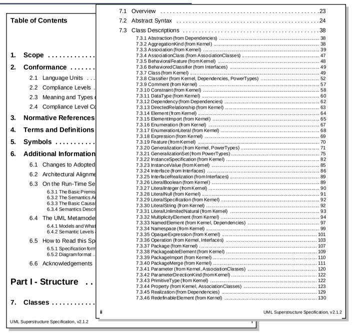


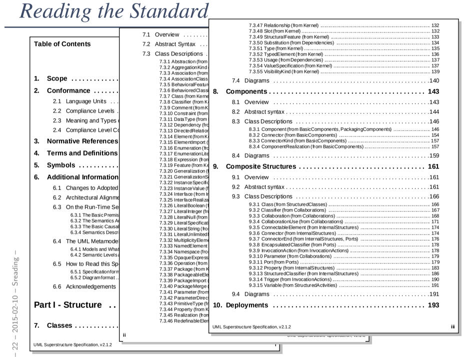
Table of Contents

UML Superstructure Specification, v2.1.2 i

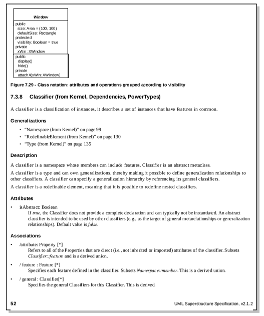
Reading the Standard



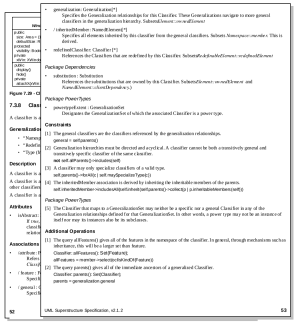
Reading the Standard



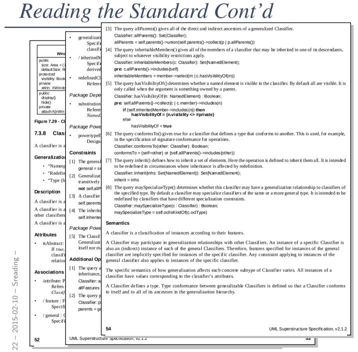
Reading the Standard Cont'd



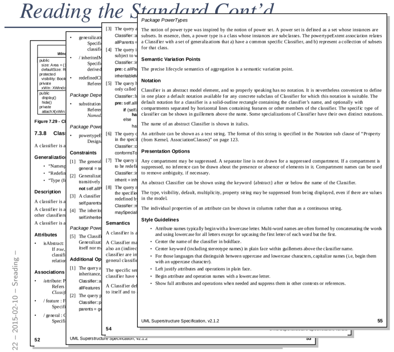
Reading the Standard Cont'd



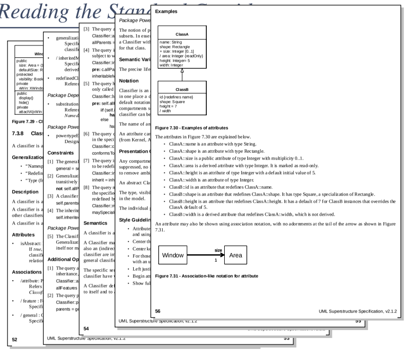
Reading the Standard Cont'd



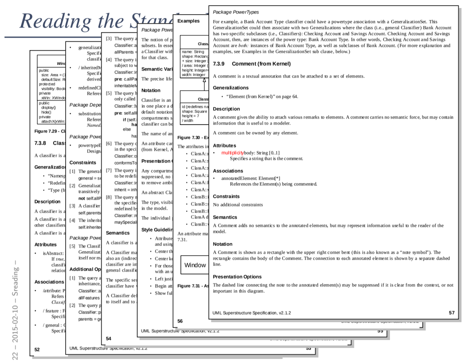
Reading the Standard Cont'd



Reading the Standard Cont'd



Reading the Standard Cont'd



Meta Object Facility (MOF)

Open Questions...

- Now you've been "**tricked**" again. Twice.
  - We didn't tell what the **modelling language** for meta-modelling is.
  - We didn't tell what the **is-instance-of** relation of this language is.
- **Idea**: have a **minimal object-oriented core** comprising the notions of **class**, **association**, **inheritance**, **etc.** with "self-explaining" semantics.
- This is **Meta Object Facility** (MOF), which (more or less) coincides with UML Infrastructure [OMG, 2007a].
- So: things on meta level
  - M0 are object diagrams/system states
  - M1 are **words of the language UML**
  - M2 are **words of the language MOF**
  - M3 are **words of the language** . . .

MOF Semantics

- One approach:
  - Treat it with **our signature-based theory**
  - This is (in effect) the right direction, but may require new (or extended) signatures for each level.
    (For instance, MOF doesn't have a notion of Signal, our signature has.)
- Other approach:
  - Define a **generic**, **graph based** "is-instance-of" relation.
  - Object diagrams (that **are** graphs) then **are** the system states — not **only graphical representations** of system states.
  - If this works out, good: We can easily experiment with different language designs, e.g. different flavours of UML that immediately have a semantics.
  - Most interesting: also do generic definition of behaviour within a closed modelling setting, but this is clearly still research, e.g. [Buschermöhle and Oelerink, 2008].

Meta-Modelling: (Anticipated) Benefits

Benefits: Overview

- We'll (superficially) look at three aspects:
  - Benefits for **Modelling Tools**.
  - Benefits for **Language Design**.
  - Benefits for **Code Generation and MDA**.

Benefits for Modelling Tools

- The meta-model $\mathcal{M}_U$ of UML **immediately** provides a **data-structure** representation for the abstract syntax ($\sim$ for our signatures).

  If we have code generation for UML models, e.g. into Java, then we can immediately represent UML models **in memory** for Java.

  (Because each MOF model is in particular a UML model.)
- There exist tools and libraries called **MOF-repositories**, which can generically represent instances of MOF instances (in particular UML models).

  And which can often generate specific code to manipulate instances of MOF instances in terms of the MOF instance.

Benefits for Modelling Tools Cont'd

- And not only **in memory**, if we can represent MOF instances in files, we obtain a canonical representation of UML models **in files**, e.g. in XML.
  $\rightarrow$ XML Metadata Interchange (XMI)
- **Note**: A priori, there is no graphical information in XMI (it is only abstract syntax like our signatures) $\rightarrow$ OMG Diagram Interchange.
- **Note**: There are slight ambiguities in the XMI standard.
  And different tools by different vendors often seem to lie at opposite ends on the scale of interpretation. Which is surely a coincidence.
  In some cases, it's possible to fix things with, e.g., XSLT scripts, but full vendor independence is today not given.
  Plus XMI compatibility doesn't necessarily refer to Diagram Interchange.
- **To re-iterate**: this is **generic for all** MOF-based modelling languages such as UML, CWM, etc.
  And also for **Domain Specific Languages** which don't even exit yet.

Benefits: Overview

- We'll (superficially) look at three aspects:
  - Benefits for **Modelling Tools**. ✔
  - Benefits for **Language Design**.
  - Benefits for **Code Generation and MDA**.

Benefits for Language Design

- Recall: we said that code-generators are possible "readers" of stereotypes.
- For example, (heavily simplifying) we could
  - introduce the stereotypes **Button**, **Toolbar**, ...
  - for convenience, instruct the modelling tool to use special pictures for stereotypes — in the meta-data (the abstract syntax), the stereotypes are clearly present.
  - instruct the code-generator to automatically add inheritance from Gtk::Button, Gtk::Toolbar, etc. **corresponding** to the stereotype.

  Et voilà: we can model Gtk-GUIs and generate code for them.
- Another view:
  - UML with these stereotypes **is a new modelling language**: Gtk-UML.
  - Which lives on the same meta-level as UML (M2).
  - It's a **Domain Specific** Modelling **Language** (DSL).

One mechanism to define DSLs (based on UML, and "within" UML): **Profiles**.

Benefits for Language Design Cont'd

- For each DSL defined by a Profile, we immediately have
  - in memory representations,
  - modelling tools,
  - file representations.
- **Note**: here, the **semantics** of the stereotypes (and thus the language of Gtk-UML) **lies in the code-generator**.

  That's the first "reader" that understands these special stereotypes.
  (And that's what's meant in the standard when they're talking about giving stereotypes semantics).
- One can also impose additional well-formedness rules, for instance that certain components shall all implement a certain interface (and thus have certain methods available). (Cf. [Stahl and Völter, 2005].)

Benefits for Language Design Cont'd

- One step further:
  - Nobody hinders us to obtain a model of UML (written in MOF),
  - throw out parts unnecessary for our purposes,
  - add (= integrate into the existing hierarchy) more adequat new constructs, for instance, **contracts** or something more close to hardware as **interrupt** or **sensor** or **driver**,
  - and maybe also stereotypes.

  → a new language standing next to UML, CWM, etc.
- Drawback: the resulting language is not necessarily UML any more, so we **can't use** proven UML modelling tools.
- But we can use all tools for MOF (or MOF-like things).
  For instance, Eclipse EMF/GMF/GEF.

Benefits: Overview

- We'll (superficially) look at three aspects:
  - Benefits for **Modelling Tools**. ✔
  - Benefits for **Language Design**. ✔
  - Benefits for **Code Generation and MDA**.

Benefits for Model (to Model) Transformation

- There are manifold applications for model-to-model transformations:
  - For instance, tool support for **re-factorings**, like moving common attributes upwards the inheritance hierarchy.

    This can now be defined as **graph-rewriting** rules on the level of MOF.
    The graph to be rewritten is the UML model
  - Similarly, one could transform a **Gtk-UML** model into a **UML model**, where the inheritance from classes like Gtk::Button is made explicit:

    The transformation would add this class Gtk::Button and the inheritance relation and remove the stereotype.
  - Similarly, one could have a **GUI-UML** model transformed into a **Gtk-UML** model, or a Qt-UML model.

    The former a PIM (Platform Independent Model), the latter a PSM (Platform Specific Model) — cf. MDA.

Special Case: Code Generation

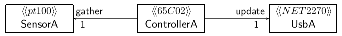
- Recall that we said that, e.g. Java code, can also be seen as a model.
  So code-generation is a **special case** of model-to-model transformation; only the destination looks quite different.
- **Note**: Code generation needn't be as expensive as buying a modelling tool with full fledged code generation.
  - If we have the UML model (or the DSL model) given as an XML file, code generation can be **as simple as** an XSLT script.

  "Can be" in the sense of

  > *"There may be situation where a graphical and abstract representation of something is desired which has a clear and direct mapping to some textual representation."*

In general, code generation can (in colloquial terms) become **arbitrarily difficult**.

Example: Model and XMI

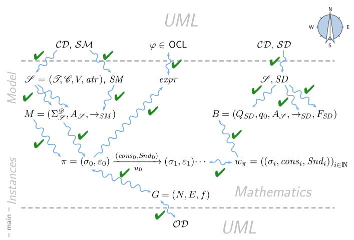



```xml
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML' timestamp = 'Mon Feb 02 18:23:12 CET 2009'>
  <XMI.content>
    <UML:Model xmi.id = '...'>
      <UML:Namespace.ownedElement>
        <UML:Class xmi.id = '...' name = 'SensorA'>
          <UML:ModelElement.stereotype>
            <UML:Stereotype name = 'pt100'/>
          </UML:ModelElement.stereotype>
        </UML:Class>
        <UML:Class xmi.id = '...' name = 'ControllerA'>
          <UML:ModelElement.stereotype>
            <UML:Stereotype name = '65C02'/>
          </UML:ModelElement.stereotype>
        </UML:Class>
        <UML:Class xmi.id = '...' name = 'UsbA'>
          <UML:ModelElement.stereotype>
            <UML:Stereotype name = 'NET2270'/>
          </UML:ModelElement.stereotype>
        </UML:Class>
        <UML:Association xmi.id = '...' name = 'in' >...</UML:Association>
        <UML:Association xmi.id = '...' name = 'out' >...</UML:Association>
      </UML:Namespace.ownedElement>
    </UML:Model>
  </XMI.content>
</XMI>
```


Wrapup & Questions

Content

Course Path: Over Map



- Motivation
- Semantical Model
- OCL
- Object Diagrams
- Class Diagrams
- State Machines
- Live Sequence Charts
- ~~Real-Time~~
- ~~Components~~
- Inheritance
- Meta-Modeling

Wrapup: Motivation

Wrapup: Motivation

Lecture 1:

- **Educational Objectives:** you should
  - be able to explain the term **model**.
  - know the idea (and hopes and promises) of **model-driven** SW development.
  - be able to explain how **UML** fits into this general picture.
  - know **what** ~~we'll do~~ we've done in the course, and **why**.
  - thus be able to decide whether you want to stay with us...
  - How can UML help with software development?
  - Where is which sublanguage of UML useful?
  - For what purpose? With what drawbacks?

Wrapup: Examining Motivation

- what is a model? for example?
- "a model is an image or a pre-image" — of what? please explain!
- when is a model a good model?
- what is model-based software engineering?
  - MDA? MDSE?
  - what do people hope to gain from MBSE? Why? Hope Justified?
  - what are the fundamental pre-requisites for that?
- what are purposes of modelling guidelines?
  - could you illustrate this with examples?
  - how can we establish/enforce them? can tools or procedures help?
- what's the qualitative difference between the modelling guideline "all association ends have a multiplicity" and "all state-machines are deterministic"?
- ...

Wrapup: Examining Motivation

- what is UML (definitely)? why?
- what is it (definitely) not? why?
- how does UML relate to programming languages?
- what are the intentions of UML?
- what is the history of UML? Why could it be useful to know that?
- where can (what part of) UML be used in MBSE?
  - for what purpose? to improve what?
- we discussed a notion of "UML mode" by M. Fowler.
  - what is that? why is it useful to think about it?

Wrapup: Examining "The Big Picture"

- what kinds of diagrams does UML offer?
- what is the purpose of the X diagram?
- what do the diagrams X and Y have in common?
- what is a UML model (our definition)? what does it mean?
- what is the difference between well-formedness ruless and modelling guidelines?
- what is meta-modelling?
  - could you explain it on the example of UML?
- what is a class diagram in the context of meta-modelling?
- what benefits do people see in meta-modelling?
- the standard is split into the two documents "Infrastructure" and "Superstructure". what is the rationale behind that?
- in what modelling language is UML modelled?

Wrapup: Modelling Structure

- Lecture 1: Motivation and Overview
- **Lecture 2: Semantical Model**
- **Lecture 3: Object Constraint Language (OCL)**
- **Lecture 4: OCL Semantics**
- **Lecture 5: Object Diagrams**
- **Lecture 6: Class Diagrams I**
- **Lecture 7: Type Systems and Visibility**
- **Lecture 8: Class Diagrams II**
- **Lecture 9: Class Diagrams III**
- Lecture 10: Constructive Behaviour, State Machines Overview
- Lecture 11: Core State Machines I
- Lecture 12: Core State Machines II
- Lecture 13: Core State Machines III
- Lecture 14: Core State Machines IV
- Lecture 15: Core State Machines V, Rhapsody
- Lecture 16: Hierarchical State Machines I
- Lecture 17: Hierarchical State Machines II
- Lecture 18: Live Sequence Charts I
- Lecture 19: Live Sequence Charts II
- Lecture 20: Inheritance I
- Lecture 21: Meta-Modelling, Inheritance II
- Lecture 22: Wrapup & Questions

Wrapup: Modelling Structure

Lecture 2:

- **Educational Objectives:** Capabilities for these tasks/questions:
  - Why is UML of the form it is?
  - Shall one feel bad if not using all diagrams during software development?
  - What is a signature, an object, a system state, etc.? What's the purpose in the course?
  - How do Basic Object System Signatures relate to UML class diagrams?

Lecture 3 & 4:

- **Educational Objectives:** Capabilities for these tasks/questions:
  - Please explain/read out this OCL constraint. Is it well-typed?
  - Please formalise this constraint in OCL.
  - Does this OCL constraint hold in this (complete) system state?
  - Can you think of a system state satisfying this constraint?
  - Please un-abbreviate all abbreviations in this OCL expression.
  - In what sense is OCL a three-valued logic? For what purpose?
  - How are $\mathscr{D}(C)$ and τ_C related?

Wrapup: Modelling Structure

Lecture 5:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What is an object diagram? What are object diagrams good for?
  - When is an object diagram called partial? What are partial ones good for?
  - How are system states and object diagrams related?
  - What does it mean that an OCL expression is satisfiable?
  - When is a set of OCL constraints said to be consistent?
  - Can you think of an object diagram which violates this OCL constraint?
  - Is this UML model $\mathcal{M}$ consistent wrt. $Inv(\mathcal{M})$?

Lecture 6:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What is a class diagram?
  - For what purposes are class diagrams useful?
  - Could you please map this class diagram to a signature?
  - Could you please map this signature to a class diagram?

Wrapup: Modelling Structure

Lecture 7:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - Is this OCL expression well-typed or not? Why?
  - How/in what form did we define well-definedness?
  - What is visibility good for? Where is it used?

Lecture 8 & 9:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - Please explain/illustrate this class diagram with associations.
  - Which annotations of an association arrow are (semantically) relevant? In what sense? For what?
  - What's a role name? What's it good for?
  - What's "multiplicity"? How did we treat them semantically?
  - What is "reading direction", "navigability", "ownership", . . . ?
  - What's the difference between "aggregation" and "composition"?

Wrapup: Modelling Structure

Lecture 9:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What are purposes of modelling guidelines? (Example?)
  - When is a class diagram a good class diagram?
  - Discuss the style of this class diagram.

Lecture 20 & 21:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What's the effect of inheritance on System States?
  - What does the Liskov Substitution Principle mean regarding structure?
  - What is the subset, what the uplink semantics of inheritance?
  - What's the idea of Meta-Modelling?

Wrapup: Modelling Behaviour, Constructive

- Lecture 1: Motivation and Overview
- Lecture 2: Semantical Model
- Lecture 3: Object Constraint Language (OCL)
- Lecture 4: OCL Semantics
- Lecture 5: Object Diagrams
- Lecture 6: Class Diagrams I
- Lecture 7: Type Systems and Visibility
- Lecture 8: Class Diagrams II
- Lecture 9: Class Diagrams III
- **Lecture 10: Constructive Behaviour, State Machines Overview**
- **Lecture 11: Core State Machines I**
- **Lecture 12: Core State Machines II**
- **Lecture 13: Core State Machines III**
- **Lecture 14: Core State Machines IV**
- **Lecture 15: Core State Machines V, Rhapsody**
- **Lecture 16: Hierarchical State Machines I**
- **Lecture 17: Hierarchical State Machines II**
- Lecture 18: Live Sequence Charts I
- Lecture 19: Live Sequence Charts II
- Lecture 20: Inheritance I
- Lecture 21: Meta-Modelling, Inheritance II
- Lecture 22: Wrapup & Questions

Wrapup: Modelling Behaviour, Constructive

Main and General:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What does this State Machine mean?
  - What happens if I inject this event?
  - Can you please model the following behaviour.
    (And **convince** readers that your model is correct.)

Wrapup: Modelling Behaviour, Constructive

Lecture 10:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What's the difference between reflective and constructive descriptions of behaviour?
  - What's the Basic Causality Model?
  - What does the standard say about the dispatching method?
  - What is (intuitively) a run-to-completion step?

Lecture 11:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - Can you please model the following behaviour.
  - What is: trigger, guard, action?
  - Please unabbreviate this abbreviated transition annotation.
  - What is an ether? Example? Why did we introduce it?
  - What's the difference: signal, signal event, event, trigger, reception, consumption?
  - What's a system configuration?

Wrapup: Modelling Behaviour, Constructive

Lecture 12 & 13:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What is a transformer? Example? Why did we introduce it?
  - What is a re-use semantics? What of the framework would we change to go to a non-re-use semantics?
  - What labelled transition system is induced by a UML model?
  - What is: discard, dispatch, commence?
  - What's the meaning of stereotype "signal,env"?
  - Does environment interaction necessarily occur?
  - What happens on "division by 0"?

Lecture 14 & 15:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What is a step (definition)? Run-to-completion step (definition)? Microstep (intuition)?
  - Do objects always finally become stable?
  - In what sense is our RTC semantics not compositional?

Wrapup: Modelling Behaviour, Constructive

Lecture 16:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What's a kind of a state? What's a pseudo-state?
  - What's a region? What's it good for?
  - What is: entry, exit, do, internal transition?
  - What's a completion event? What has it to do with the ether?

Lecture 17:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What's a state configuration?
  - When are two states orthogonal? When consistent?
  - What's the depth of a state? Why care?
  - What is the set of enabled transitions in this system configuration and this state machine?

Wrapup: Modelling Behaviour, Constructive

Lecture 18:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What's a history state? Deep vs. shallow?
  - What is: junction, choice, terminate?
  - What is the idea of "deferred events"?
  - What is a passive object? Why are passive reactive objects special? What did we do in that case?
  - What's a behavioural feature? How can it be implemented?

Wrapup: Modelling Behaviour, Reflective

- Lecture 1: Motivation and Overview
- Lecture 2: Semantical Model
- Lecture 3: Object Constraint Language (OCL)
- Lecture 4: OCL Semantics
- Lecture 5: Object Diagrams
- Lecture 6: Class Diagrams I
- Lecture 7: Type Systems and Visibility
- Lecture 8: Class Diagrams II
- Lecture 9: Class Diagrams III
- Lecture 10: Constructive Behaviour, State Machines Overview
- Lecture 11: Core State Machines I
- Lecture 12: Core State Machines II
- Lecture 13: Core State Machines III
- Lecture 14: Core State Machines IV
- Lecture 15: Core State Machines V, Rhapsody
- Lecture 16: Hierarchical State Machines I
- Lecture 17: Hierarchical State Machines II
- **Lecture 18: Live Sequence Charts I**
- **Lecture 19: Live Sequence Charts II**
- Lecture 20: Inheritance I
- Lecture 21: Meta-Modelling, Inheritance II
- Lecture 22: Wrapup & Questions

Wrapup: Modelling Behaviour, Reflective

Lecture 18, & 19:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - Is each LSC description of behaviour necessarily reflective?
  - There exists another distinction between "inter-object" and "intra-object" behaviour. Discuss in the context of UML.
  - What does this LSC mean?
  - Are this UML model's state machines consistent with the interactions?
  - Please provide a UML model which is consistent with this LSC.
  - What is: activation (mode, condition), hot/cold condition, pre-chart, cut, hot/cold location, local invariant, legal exit, hot/cold chart etc.?

Wrapup: Inheritance

- Lecture 1: Motivation and Overview
- Lecture 2: Semantical Model
- Lecture 3: Object Constraint Language (OCL)
- Lecture 4: OCL Semantics
- Lecture 5: Object Diagrams
- Lecture 6: Class Diagrams I
- Lecture 7: Type Systems and Visibility
- Lecture 8: Class Diagrams II
- Lecture 9: Class Diagrams III
- Lecture 10: Constructive Behaviour, State Machines Overview
- Lecture 11: Core State Machines I
- Lecture 12: Core State Machines II
- Lecture 13: Core State Machines III
- Lecture 14: Core State Machines IV
- Lecture 15: Core State Machines V, Rhapsody
- Lecture 16: Hierarchical State Machines I
- Lecture 17: Hierarchical State Machines II
- Lecture 18: Live Sequence Charts I
- Lecture 19: Live Sequence Charts II
- **Lecture 20: Inheritance I**
- **Lecture 21: Meta-Modelling, Inheritance II**
- Lecture 22: Wrapup & Questions

Wrapup: Inheritance

Lecture 20 & 21:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What's the effect of inheritance on LSCs, State Machines, System States?
  - What's the Liskov Substitution Principle?
  - What is commonly understood under (behavioural) sub-typing?
  - What is the subset, what the uplink semantics of inheritance?
  - What is late/early binding?
  - What's the idea of Meta-Modelling?

Hmm...

- Open book or closed book...?

References

[Buschermöhle and Oelerink, 2008] Buschermöhle, R. and Oelerink, J. (2008). Rich meta object facility. In Proc. 1st IEEE Int'l workshop UML and Formal Methods.

[OMG, 2003] OMG (2003). Uml 2.0 proposal of the 2U group, version 0.2, http://www.2uworks.org/uml2submission.

[OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

[OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.

[Stahl and Völter, 2005] Stahl, T. and Völter, M. (2005). Modellgetriebene Softwareentwicklung. dpunkt.verlag, Heidelberg.