Systems Simulation
ECE 697 SI

SysML – System Modeling Language

Prof. Michael Zink
Overview

- Software Architecture
- OMG
- UML
- SysML Overview
Example: Software Architecture

**Software Architecture** of a program or computing system is the structure/structures of a system which comprises of software components, the externally visible properties of those components and the relationships among them.
Software Architecture Features

- Organization of system as a composition of components
- Identifies global control structures
- Provides protocols for communication
- Provides synchronization & data access
- Is a composition of design elements
- Provides physical distribution
- Adds dimension towards evolution
Software Architecture Roles

- Understanding
- Reuse
- Construction
- Evolution
- Analysis
- Management
- Communication
Technology Basis of Architecture

Three important advancements in the technology basis of Architecture:

- Development of Architecture description languages and tools.
- Emergence of product line engineering and architecture standards.
- Codification and dissemination of architecture design expertise.

ADL has tools for parsing, displaying, compiling, analyzing or simulating architectural descriptions:

Acme, Adage, Aesop, Darwin, Rapide, SADL, Unicon, Meta-H, Wright
Architecture Views

- Classes of views:
  - Code-oriented view
  - Execution oriented view

- Execution-oriented views should have

1. **Components** – multiple interfaces to environment
2. **Connectors** – Interactions among components; mediate communications and coordinate activities among components
3. **Systems** – Graphs of components and connectors
4. **Properties** – Used to represent anticipated & required extra-functional aspects of architecture designs
5. **Styles** – Vocabulary of design elements
Object Oriented (OO)

- **OO Analysis**
  - Builds a model of a system that is composed of objects.
  - Behavior of the system is achieved through collaboration between these objects.
  - The state of the system is the combined state of all the objects in it.
  - An analysis model will not take into account implementation constraints.

- **OO Design**
  - System is modeled as a collection of cooperating objects.
Architecture – OO Relationship

- Proposed perspectives:
  - OOD as Architectural Style
  - OOD as implementation base
  - OOD as modeling notation
Why Modeling?

- Need of unified view of various lifecycle activities and their interdependencies ...
- motivated model-based approaches
- ... which heavily rely on:
  - use of modeling tool
  - methods to provide support
  - guidance for development
  - validation.
Model Driven Architecture

Provides an open vendor neutral approach to the challenges of business and technology change. It separates business and application logic from underlying platform technology.
During the development process of any system it is necessary to support interoperability with specifications that address integration through entire lifetime of the system:
MDA Integration

- Embracing tech. like CORBA, J2EE, XML, .NET etc.

- Improving the portability of applications by allowing some model to be realized on multiple platforms through mapping standards.

- Improving integration based on models of relationships across different domain applications & interfaces allowing interoperability.
Models

PIM – Platform Independent Models (e.g. CORBA)
- Provides formal specification of the structure and function of the system that abstracts away technical details.
- Describes the computational components and their interactions in a platform-independent manner.

PSM – Platform Specific Models
- Functionality specified in PIM is realized in platform-specific way in the PSM.
- Derived from the PIM via some transformation.
PIM – PSM Combination

All OMG standards are based on this approach of defining systems in terms of PIM and one or more PSM (e.g., UML)

PIM has OCL (Object Constraint Language)
- It formalizes the vocabulary otherwise left imprecise in interface Specification (JAVA & Microsoft IDL).
- Abstract yet precise model of state of object providing interface and any parameter exchanged.

PIM to PSM mappings are maintained to provide the mapping between logical component model to existing commercial component model.
The Unified Modeling Language (UML) is a standard language for specifying, visualizing constructing and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.
Development of UML began in late 1994 when Grady Booch and Jim Rumbaugh of Rational Software Corporation began their work on unifying the Booch and OMT (Object Modeling Technique) methods.

In the Fall of 1995, Ivar Jacobson and his Objectory company joined Rational and this unification effort, merging in the OOSE (Object-Oriented Software Engineering) method.

As the primary authors of the Booch, OMT, and OOSE methods, Grady Booch, Jim Rumbaugh, and Ivar Jacobson were motivated to create a unified modeling language as all were working on common things.
Diagram Types

- **Structure Diagrams**
  - Class Diagram
  - Component Diagram
  - Package Diagram
  - Object Diagram
  - Composite Structure Diagram
  - Deployment Diagram

- **Behavior Diagrams**
  - Use Case Diagram
  - Activity Diagram
  - State Machine Diagram

- **Interaction Diagrams**
  - Sequence Diagram
  - Interaction Overview Diagram
  - Communication Diagram
  - Timing Diagram
UML Super Structure Specification
Structure Diagram: Class Diagram

Class diagram shows relationships between various classes in system. Classes being identified use OOD paradigm. It’s a static diagram and hence does not show the way in which the interaction happens.

- **association** -- an instance of one class must know about the other in order to perform its work. (  )

- **aggregation** -- an association in which one class belongs to a collection. (  )

- **composition** -- an association in which one class is composed of other classes. (  )

- **generalization** -- an inheritance link indicating one class is a superclass of the other. (  )
The relationships are shown in terms of the instances created of the objects that we defined. The various objects instantiated are shown as well the relationships between those objects.
Structure Diagram: Package Diagram

Package diagram shows the hierarchy in which the system will be modeled in the implementation. It gives a high level view for the distribution that can be created within the specified project and specify the package visibility.
Structure Diagram: Composite Structure Diagram

- Composite Structure diagram reflects the internal collaboration of classes, interfaces or components to describe a functionality.
- Composite Structure diagrams are similar to Class diagrams, except that they model a specific usage of the structure.
- Composite Structure diagram is used to express run-time architectures, usage patterns, and the participating elements' relationships, which might not be reflected by static diagrams.
Structure Diagram: Composite Structure Diagram
Structure Diagram: Deployment Diagram

Deployment diagram shows how and where the system will be deployed. Physical machines and processors are reflected as nodes, and the internal construction can be depicted by embedding nodes or artifacts.
Structure Diagram: Component Diagram

- Component diagram illustrates the pieces of software, embedded controllers, etc. that will make up a system.
- Component diagram has a higher level of abstraction than a Class diagram.
- Usually a component is implemented by one or more classes (or objects) at runtime.

Diagram:

- Seminar Management <<UI>>
- Student Administration <<UI>>
- Seminar
- Student
- Facilities
- Schedule
- University DB <<database>>
- Persistence <<infrastructure>>
- Encryption
- Access Control
- Persistence
- Encryption
- JDBC
Behavior Diagram: Use Case Diagram

Use cases diagram basically are a substitute of the requirements being Modeled into the architecture. The external viewpoint as to “what” the System should do rather “how”.

[Diagram of a use case diagram showing interactions between Patient, Scheduler, Doctor, and Clerk with activities including Cancel Appointment, Make Appointment, Request Medication, and Pay Bill.]
Behavior Diagram: Activity Diagram

- Activity diagrams are used to model the behaviors of a system, and the way in which these behaviors are related in an overall flow of the system.

- The logical paths a process follows, based on various conditions, concurrent processing, data access, interruptions and other logical path distinctions, are all used to construct a process, system or procedure.
Behavior Diagram: State Machine Diagram

A State Machine diagram illustrates how an element, often a class, can move between states classifying its behavior, according to transition triggers, constraining guards and other aspects of State Machine diagrams that depict and explain movement and behavior.
Interaction Diagram: Sequence Diagram

Sequence diagram is an interaction diagram that details how operations are carried out -- what messages are sent and when.

Sequence diagrams are organized according to time expressed in the sequential order along the vertical plane.

If a room is available for each day of the stay, make a reservation and send a confirmation.
Interaction Diagram: Comm./Coll. Diagram

Communication diagram shows the interactions between elements at run-time in much the same manner as a Sequence diagram. Communication diagrams are used to visualize inter-object relationships, while Sequence diagrams are more effective at visualizing processing over time.
Interaction Diagram: Timing Diagram

- Timing diagram defines the behavior of different objects within a time-scale.
- Provides a visual representation of objects changing state and interacting over time.
- Used for defining hardware-driven or embedded software components.
Interaction Diagram: Interaction Overview D.

- Interaction Overview diagrams visualize the cooperation between other interaction diagrams to illustrate a control flow serving an encompassing purpose.

- Interaction Overview diagrams are a variant of activity diagrams, most of the diagram notation is similar, as is the process in constructing the diagram.

- Interaction elements display an inline Interaction diagram, which can be a Sequence diagram, Communication diagram, Timing diagram, or Interaction Overview diagrams.
Customer places order

Check customer credit

Order exceeds credit limit

[true]

Increase customer credit

[true]

Ship order

Reject order

Sales Manager review

(max 4 business hours)
SysML was identified as the key modeling language that could integrate the disparate tools sets that SE’s use for a single project across various domains.

The idea behind it was that it would do the same trick for the Systems Engineering field as UML did for the loads of modeling languages in Software.
SysML

- Systems Engineering has not had a uniform modeling language.
- Requires language that is independent of specific disciplines (e.g., software, hardware, mechanics).
- Resulted in expanded language.
- Adapted UML is called *Systems Modeling Language (SysML)*.
Design Principles

1. **Parsimony** – Surgical reduction and augmentation of UML

2. **Reuse** – Reuse of UML 2.0

3. **Modularity** – Principle of strong cohesion and loose coupling

4. **Layering** – SysML packages as extensions layer to UML meta model.

5. **Partitioning** – Configure conceptional areas within some layer.

6. **Extensibility** – It offers the same extension mechanism as UML (meta classes, stereotypes, model libraries).

7. **Interoperability** – SysML is aligned with the semantics of ISO AP-232, same as XMI in UML.
Architecture

- SysML reuses and extends packages from UML, extension mechanisms like stereotypes, meta classes and model libraries.
- Structure still remains the same for the constructs.
- It uses combination of profiling and meta modeling techniques that use precise language to specify constraints and semantics.
Changes to UML

- Classes are called *blocks*. In SysML, the UML class diagram is called *block definition diagram*.
- The UML composite structure diagram is called *internal block diagram*.
- Item flows between elements in the internal block diagram can be modeled.
- Continuous functions are supported by actions and object nodes in activity diagrams, and *Enhanced Functional Flow Block Diagrams (EFFBD)* are also supported.
- New diagrams: *Requirement Diagram* and *Parametric Diagram*.
- ISO AP-233 data format was added to exchange data between different tools.
History

- Request for Information was published in 2002.
- Request for Proposal on “UML for Systems Engineering” was published in 2003 and SysML Partners group was formed.
- In 2006 groups re-united and SysML 1.0 was published in 09/2007.
- Latest standard: SysML 1.2 (6/2010)
Meta Model for SysML

- Structural Constructs
- Behavioral Constructs
- Auxiliary Constructs

+ State machines, Interactions Diagram, Use cases remain the same
+ New extensions to packages of activity, classes and auxiliary diagnostics.
+ New constructs in the form of Requirements, Allocation and Parametric Diagrams
Package Hierarchy in SysML

- SysML Diagram
  - Behavior Diagram
    - Activity Diagram
    - Sequence Diagram
  - Requirement Diagram
    - State Machine Diagram
    - Use Case Diagram
  - Structure Diagram
    - Block Definition Diagram
    - Internal Block Diagram
    - Parametric Diagram
    - Package Diagram

- Same as UML 2
- Modified from UML 2
- New diagram type
Structure of SysML

**Structure**
- Block definition diagram
- Internal block diagram
- Parametric diagram
- Package diagram

**Behavior**
- Activity diagram
- Use case diagram
- State machine diagram
- Sequence diagram

**Other**
- Requirement diagram, stereotype, model view, AP-232, XMI Metadata Interchange format
Structure of SysML

1. Structure

2. Behavior

3. Requirements

4. Parametrics

Note that the Package and Use Case diagrams are not shown in this example, but are respectively part of the structure and behavior pillars.
Structural Constructs: Class Diagram

- Addition to UML specification
- Dependency Set added to group dependency relationships
- Root notation added to depict multi level hierarchy
Requirement

- Describes a contract between principal and all those who create, system design, and implement system.
- Requirement specifies flows or conditions that have to be met by the system.
Cross Cutting Constructs: Requirement Diagram

- Requirement may specify a function, a system must perform/performance condition a system must satisfy.
- Formalized to connect to other modeling elements (itself, analysis, design, testing and implementing elements).
- Type of modeling element can be controlled by using the requirement diagram.
- Requirement may have its own property, hence computable value not only text.
Two elementary properties of requirements diagram:
- Unique identifier
- Descriptive text

Operations are not permitted

Functional requirements are described in more detail in other elements (e.g., interfaces)

Requirements are always abstract
- Since they are pure specification elements, no need to create instances of requirements.
Cross Cutting Constructs: Requirement Diagram
Requirement Diagram: Additional Requirements

**Derive requirement Relationship**: describes a requirement that was derived from another requirement.

- **Requirement:** Derive Requirement
- **Derive Requirement:**
  - **Requirement:** Regenerative Braking
    - **Requirement:**
      - **Derive Requirement:**
        - **Requirement:** HybridSUV: Performance: Range
        - **Requirement:**
          - **Derive Requirement:**
            - **Requirement:** HybridSUV: Performance: Acceleration
            - **Requirement:**
              - **Derive Requirement:**
                - **Requirement:** HybridSUV: Eco-Friendliness: Emissions
                - **Requirement:**
                  - **Derive Requirement:**
                    - **Requirement:** HybridSUV: Performance: Fuel Economy
                    - **Requirement:**
                      - **Derive Requirement:**
Requirement Diagram: Additional Requirements

- **Namespace containment**
  Describes that a requirement is contained in another requirement.

- **Satisfy relationship**
  Describes that a design element satisfies a requirement.

- **Copy relationship**
  Describes that a copy is a requirement of another requirement.

- **Verify relationship**
  Connects a test case with the requirement that is verified by the test case.
 Requirement Diagram: Additional Requirements

- **Test case**
  A flow that checks whether or not the system satisfies a requirement.

- **Refine relationship**
  Specifies that a model element describes the properties of a requirement in more detail.

- **Trace relationship**
  Relationship between a requirement and an arbitrary model element. For traceability reasons only.

- **Table notation**
  Tables instead of graphical form.
Structural Constructs: Assembly Diagram

- Capability to model systems as a tree of modular components. Can be used throughout the development process multiple times.
- Views and allocations specifically for multiple representations in SysML.
Structural Constructs: Parametric Diagram

- Parametric models are analysis models that define a set of system properties & parametric relationships among them.

- Used essentially with Assembly level diagram.

- Time can be modeled as an additional property & other properties may depend on it.
Behavioral Constructs: Activity Diagram

Extensions to UML2:

1. **Control as Data:**
   - Control can disable the actions that are executing.
   - Transform its inputs to produce an output to control other actions.

2. **Continuous systems:**
   - Any sort of distributed flow of information & physical item through system.
   - “Nobuffer” and “Overwrite” features added.

3. **Probability:**
   - Edges which have probabilities associated for the likelihood of values traveling on an edge.
Behavioral Constructs: Interaction Diagram

In SysML we only have the Timing Diagrams and Sequence diagrams defined. The support for Interaction Overview diagram has been removed.
Behavioral Constructs: Timing Diagram

Same as UML standard. No additions have been made to the Timing Diagram.
Cross Cutting Constructs: Allocation Diagram

- Term used by SE’s to denote organized cross-association of elements within the various structures /hierarchies of a user model. Support allocation in broad sense.
Allocation Diagram

- Interconnect elements from different model areas on an abstract level.
- Three types of allocations:
  - **Behavior allocation** is used to allocate a behavior (activity or single action) to a block that realizes this behavior.
  - **Structure allocation** allows to separate logical from physical structures. E.g., systems engineer first creates system logically and then building hardware in a second step.
  - **Object flow allocation** connects item flow in a structure diagram with object flow edge in activity diagram.
Block Diagrams

- Central elements in object orientation are classes and objects.
- SysML does not use them in this form since they are too software-oriented.
- All static concepts and objects are blocks in SysML.
- SysML uses the term *block definition diagram* instead of *class diagram*, and the term *internal block diagram* instead of *composite structure diagram*.
- Blocks describe a system as a collection of parts that each play a certain role in a defined context.
**Block Diagrams**

**ibd** [block] On-board computer [access system]

- **Card reader**
- **On-board computer control**
- **Customer card**
## Block Diagrams

**bdd [package] Card reader**

<table>
<thead>
<tr>
<th>Values</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency: MHz</td>
<td>rate: kBaud</td>
<td><strong>&lt;&lt;Interval&gt;&gt;{min=-20, max=65}</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operations</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>led(f: color, blinking: Boolean)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constraints</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(rate &gt; 32)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parts</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c: Chip</td>
<td>fastening: Suction cup</td>
<td>g: Enclosure</td>
<td></td>
</tr>
</tbody>
</table>
Block Diagrams

- **Distribution definition**
  Describes how values are distributed over a defined value range.

- **Value type**
  Defines values that have no identity, and cannot be referenced to by a block, but can have a unit or dimension.

- **Unit**
  Describes the structure of a physical unit.

- **Dimension**
  Describes the quantity of the unit.
Block Diagrams

- **Flow port**
  Describes an interaction point of a block used by the block to interact with its environment, and objects can flow into and out of the block.

- **Item flow**
  Is a special information flow that describes at a connector in the internal block diagram that specific objects are transported.

- **Data types**
  *Real* and *Complex* in addition to *Boolean, Integer, String, Unlimited* (UML).
Auxiliary Constructs

Includes notations and elements for the following auxiliary items:

- Item Flows
- Rationale – Documents the principles of reasons for a model choice
- Additional Data types – Complex/ Real
- Dimensional Quantities – Fundamental type that defines basic type of values expressed as quantity.
- Probability Distribution – Parametric constraints that constrain properties
- Property value Constraints
View and Viewpoints

- **View** is a representation of the entire system, seen from a defined viewpoint.

- **Viewpoint** specifies the structure of a view based on the goals of a set of stakeholders.

- **Conform relationship** binds view with the viewpoint the requirements of which it satisfies.

- View supports all kinds of interests by providing a view of the entire model for a specific target group (e.g., requirements analyst, software developer, etc.)
View and Viewpoints

Diagram showing relationships between maintenance, car electrical system, driver, rain sensing wiper system, initialization, automatic wiping, actuate, manual deactivate, and sensing.
Problem

- **Problem** documents an (potential) error or flow in the model or modeled system.

- Problems often cannot be solved ad-hoc.
- Way to document information in model.
Summary

- Started with software architecture
- OMG versus MDA
- Introduction of UML
- Introduction of SysML
- List of SysML tools:
  http://www.sysmlforum.com/tools.htm