AMDARIS

REQUIREMENTS MODELLING

October 2019 | Olesea Oaserele

WHAT WILL WE COVER

- Why do we need this headache
- Basic concepts of requirement modelling
- Requirement dependencies
- Requirements views modelling
- Quality of the models
- Types of modelling

WHY DO WE HAVE TO MODEL REQUIREMENTS

THE BENEFITS OF MODELLING REQUIREMENTS

Typically, software systems today comprise significantly more **complex processes**, meaning that the associated **textual** requirements **are very extensive** and complex.
It is then difficult for the reader to **understand the interactions** within such complex processes solely on the basis of textual requirements.
The main benefit of modelling the requirements is their **clarity** over the textual representation.

- modeling the requirements shows the necessary behavior of the system in a more structured and understandable way
- the reader can follow the process step by step more easily

TEXTUAL VS MODELED REQUIREMENTS

Textual requirements

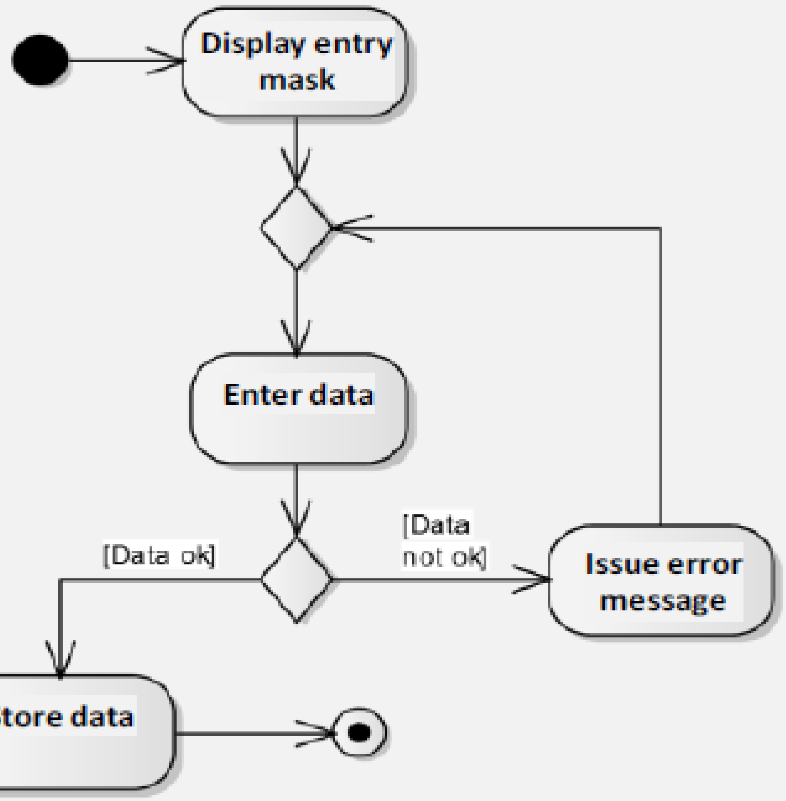
Req-1: The system shall display the entry mask

Req-2: After the action "Show entry mask" is completed, or after the action "Show error" is completed, the system shall offer the user the option to enter data

Req-3: After the action "Enter data" is completed and if the data is ok, the system shall store the data

Req-4: After the action "Enter data" is completed and if the data is not ok, the system shall issue an error message

Modeled requirements



ADVANTAGES OF MODELS OVER TEXT

Requirements are easier to understand
Cognitive research has shown that, generally, facts that are visualized in diagrams are easier to understand and remember than corresponding textual descriptions of these facts . *"A picture is worth a thousand words!"*

Support of the principle of "separation of concerns"
Requirements are modelled by different diagram types. Each diagram is designed for a specific purpose and, through the available notation elements (semantics) and the way the language allows these notation elements to be combined (syntax), force the modeler to **focus on one situation** at a time.
Ex: State machine diagrams model the reactive behavior of the system as part of requirements modeling and don not model processes or information structures.
The separation of concerns is established by **different views**.

Support of the principle "divide and rule"

- Specific requirements can initially be modeled in isolation, using different types of diagrams.
- The diagrams can be combined using common concepts or defined mapping relations in order to obtain an integrated requirements model.
- Diagram-based specification of requirements supports the breaking down the overall problem (specification) into manageable sub-problems.
- The merging of the individual requirements models of the sub-problems then forms the requirements model of the higher level system.

Reduced risk of ambiguity

Requirements specified in diagram form have a lower risk of ambiguity or misinterpretation due to the **higher degree of formality** of modeling languages compared to natural languages

Higher potential for automated analysis of requirements
Due to higher formality of modelling language, diagrams are better analysed by machines (software) than natural language.

Higher potential for automatic processing of requirements
The higher degree of formalization of requirements specified in diagram form also increases the possibility of processing the requirements of the system further automatically and using them in other development disciplines. For example, to derive test cases for system testing from requirements diagrams of the control flow-oriented view.

Requirements in context

The modeling of requirements offers the possibility to represent in the same model separate requirement elements and their relationships. This facilitates the handling of large and complex requirements and promotes their understanding because the context and relationships of a requirement is clearly visible in the model.

Ex: In an activity diagram, for every action it is immediately visible what’s next and what system state change is triggered by the execution of the action.

APPLICATION OF REQUIREMENTS MODELLING

Modeling Requirements as a Means of Specification

- requirements diagrams replace textually specified requirements
- requirements diagrams are used as the primary means for specifying the system requirements or part of the system requirements.
- requirements diagrams can (and should) be supplemented by textual requirements or textual explanations, specifically when a text is more compact or easier to handle than diagrams.
- if all requirements still need to be available in textual form (e.g., due to contractual conditions or certification requirements), they can be generated from the requirements models—for example, using templates for converting requirements diagrams into text form

Modeling Existing Textual Requirements for the Purpose of Testing

- requirement diagrams are created for a logically coherent set of textually specified requirements
- complex system behavior is very clear and understandably represented in models, hence granular test cases are easy to be created
- diagrams offer possibility to check the comprehensibility of textual requirements or to uncover inconsistencies or omissions in the textual requirements
- defects uncovered in models are then corrected in the textual requirements

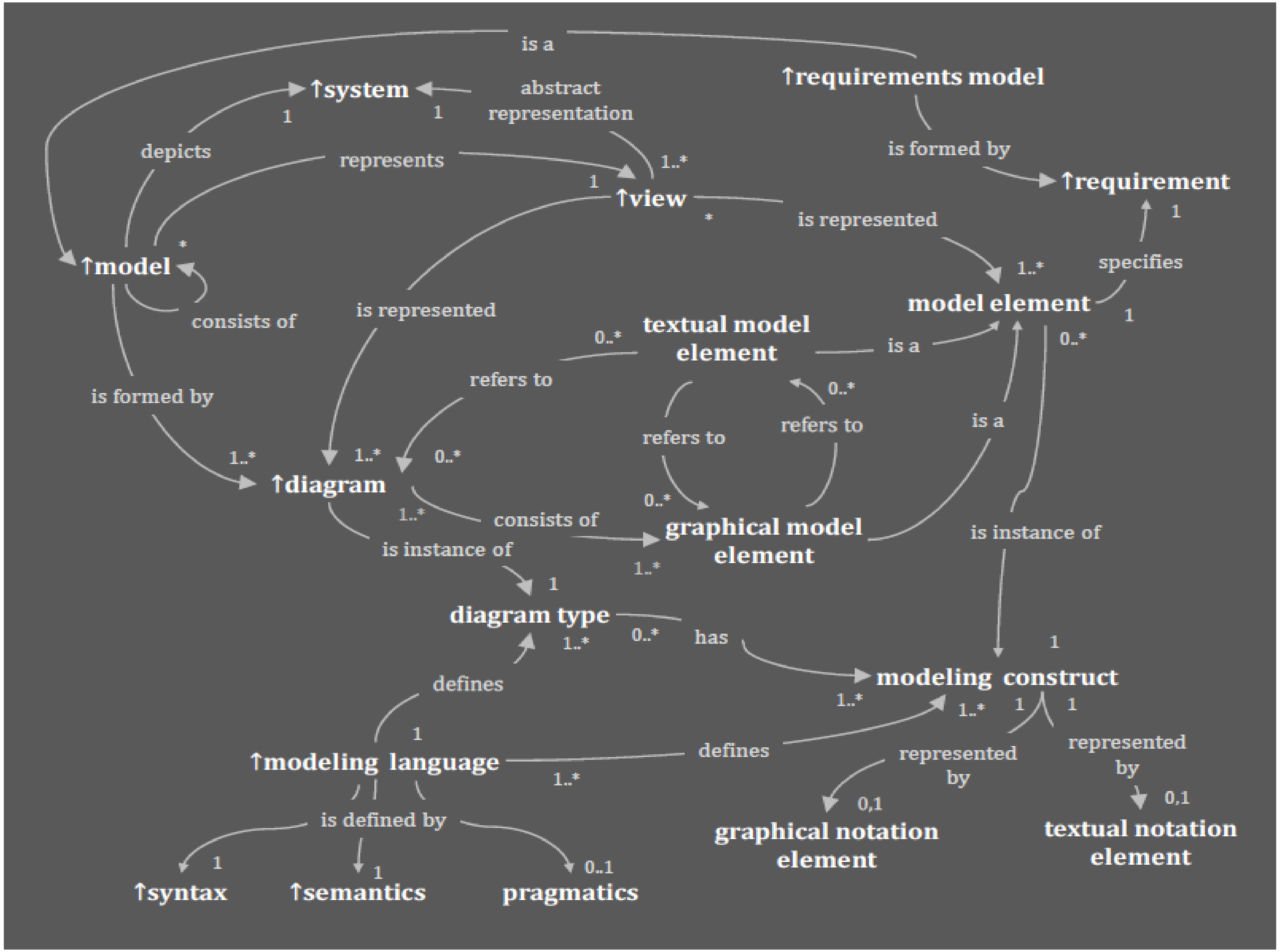
Modeling Existing Textual Requirements for Clarity

- modeled requirements are used to represent extensive and complex relationships that affect the behavior of the system.
- be aware that this redundant form of the specification can lead to significant problems with regard to contradictions between textually specified requirements and modeled requirements

BASIC CONCEPTS

TERMS AND CONCEPTS IN REQUIREMENTS MODELING

- A **model** – an abstracting image of the properties of a **system***.*
- To make the scope and complexity of the modeling manageable, various **views** of the system and its environment are documented
- The views discussed further are NOT the unique perspective and NOT a standard.
- The properties of the system in relation to each specific view are represented through **diagrams** and supplementary **textual** model elements.
- Each diagram is based on a specific **diagram type**
- Each diagram type is represented via a **modeling language** (more precisely by **syntax***,* **semantics**, and **pragmatics***).*
- The underlying modeling language of a diagram type defines the set of **modeling constructs** that can be used to construct the corresponding diagrams (e.g., class and association for the construction of class diagrams).
- In a modeling language, **graphical and/or textual notations** are defined for the modeling constructs.

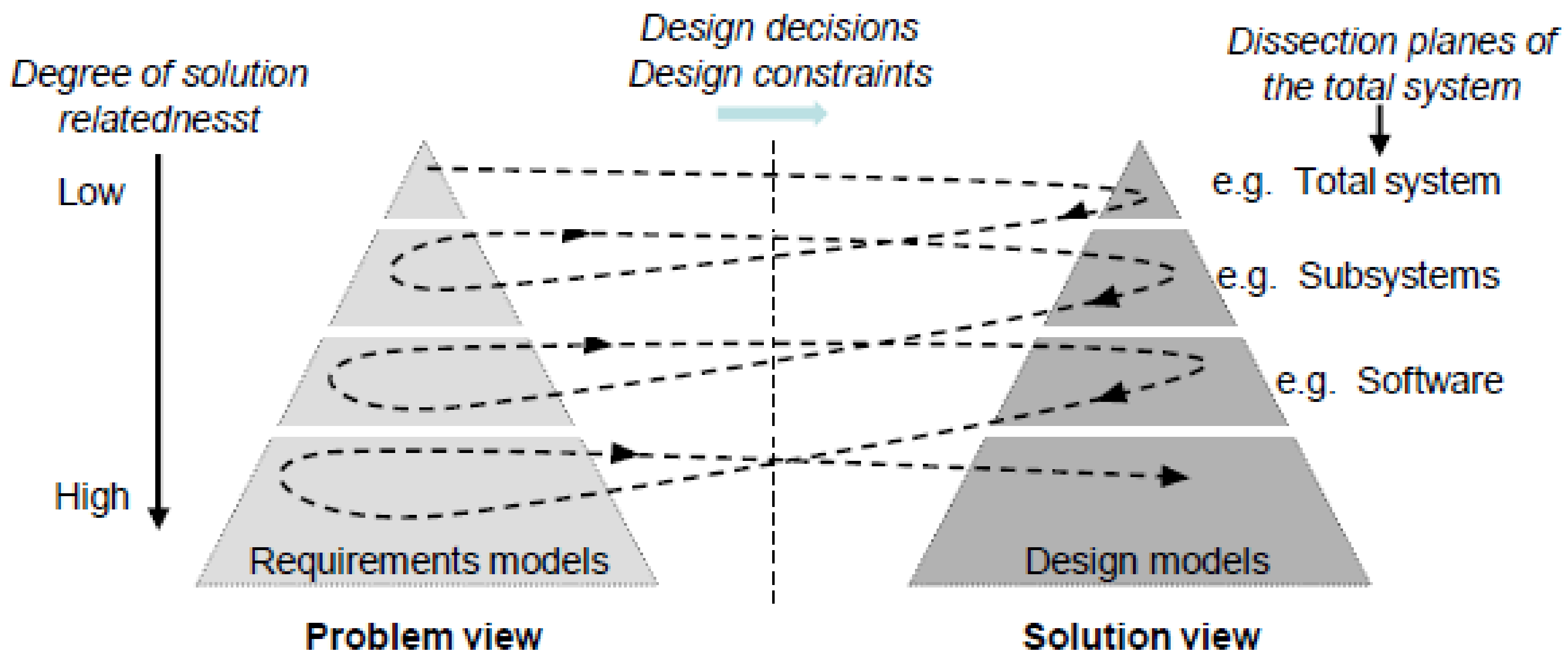

is a
↑system
abstract representation
↑requirements model
1
1
depicts
represents
1..*
is formed by
1
↑view
↑requirement
*
is represented
1
*
specifies
↑model
1..*
model element
consists of
is represented
1
textual model element
0..*
is a
0..*
is formed by
refers to
0..*
refers to
refers to
is a
1..*
1..*
0..*
↑diagram
0..*
1..*
consists of
graphical model element
is instance of
is instance of
1..*
1
diagram type
0..*
has
1..*
1
defines
modeling construct
1..*
1..*
1
1
1
represented by
defines
represented by
↑modeling language
1..*
0,1
0,1
is defined by
graphical notation element
textual notation element
1
1
0..1
↑syntax
↑semantics
pragmatics

REQUIREMENTS MODELING VERSUS SYSTEM DESIGN

- In practice, it is sometimes difficult to **distinguish** between requirements diagrams and design diagrams.
- The cause is **frequently seen** in the fact that the same universal modeling languages are used for design and requirements modeling, such as UML, SysML, BPMN.
- In fact, **the cause in most cases** is that the alleged requirements diagrams specify not requirements but rather the system design
- or that requirements and design are mixed in diagrams.

- during the development of complex software systems, there is a **strong interaction** between the definition of requirements and the system design
- it is often the case that both design diagrams and requirements diagrams are **created in parallel** at the same time
- requirements and design are often developed with very **strong links**
- typically, the first step is to produce a set of more **general requirements** for the complete system.
- this set of requirements is then the basis for the definition of the **preliminary system architecture** which satisfies these requirements.
- during the transition between requirements definition and system design, design decisions have to be made and the given **conditions for the design** (design constraints) have to be met
- Starting from the initial system architecture, the **requirements for the individual subsystems** can be specified
- If sufficiently detailed requirements are available, the **initial system design is refined.**

RELATIONSHIP BETWEEN REQUIREMENTS MODELS AND DESIGN MODELS



MODELING LANGUAGES FOR REQUIREMENTS

- For requirements modelling are available a number of diagram types and modelling languages
- Selection of the diagram and its representation (language) depends very closely on:
  - the target audience (who will be reading the model)
  - the purpose if the diagram (which specific requirement is to be modeled)
  - the type of the system to be modelled (operational or embedded)
  - application domain (banks, insurance, automotive, recruitment,…)
- Sometimes it’s more important that the model is understood and is useful that the exact language is used
- It’s important to coach all stakeholders to have the same understanding of the chosen representation.
- Requirements models can and should contain textually represented requirements in order to top up the graphical model when needed.

UML - The Unified Modeling Language is a **general-purpose** visual modeling language.

- used in the field of software engineering
- intended to specify, visualize, construct, and document the artifacts of a software system.
- is a standard notation for the **modeling of a system**, but **not a way of designing a system**.
- the UML specification is intended to support most existing object-oriented development processes.

UML diagrams are:

- **Structure diagrams:** Class Diagram, Object Diagram, Package Diagram, Composite Structure Diagram, Component Diagram, Deployment Diagram, Profile Diagram
- **Behavior diagrams**: Use Case Diagram, Information Flow Diagram, Activity Diagram, State Machine Diagram, Sequence Diagram, Communication Diagram, Timing Diagram,

BPMN - The Business Process Model and Notation (BPMN) is a graphical illustration of **business processes**

- its primary goal is to provide a notation that is easily understandable by all **business users**.
- the BPMN notation is used by business analysts and developers alike to represent a **business process** in an intuitive visual form.

BPMN diagrams are: Business Process Diagram, Choreography Diagram Collaboration Diagram, Conversation Diagram.

UML VS BPMN

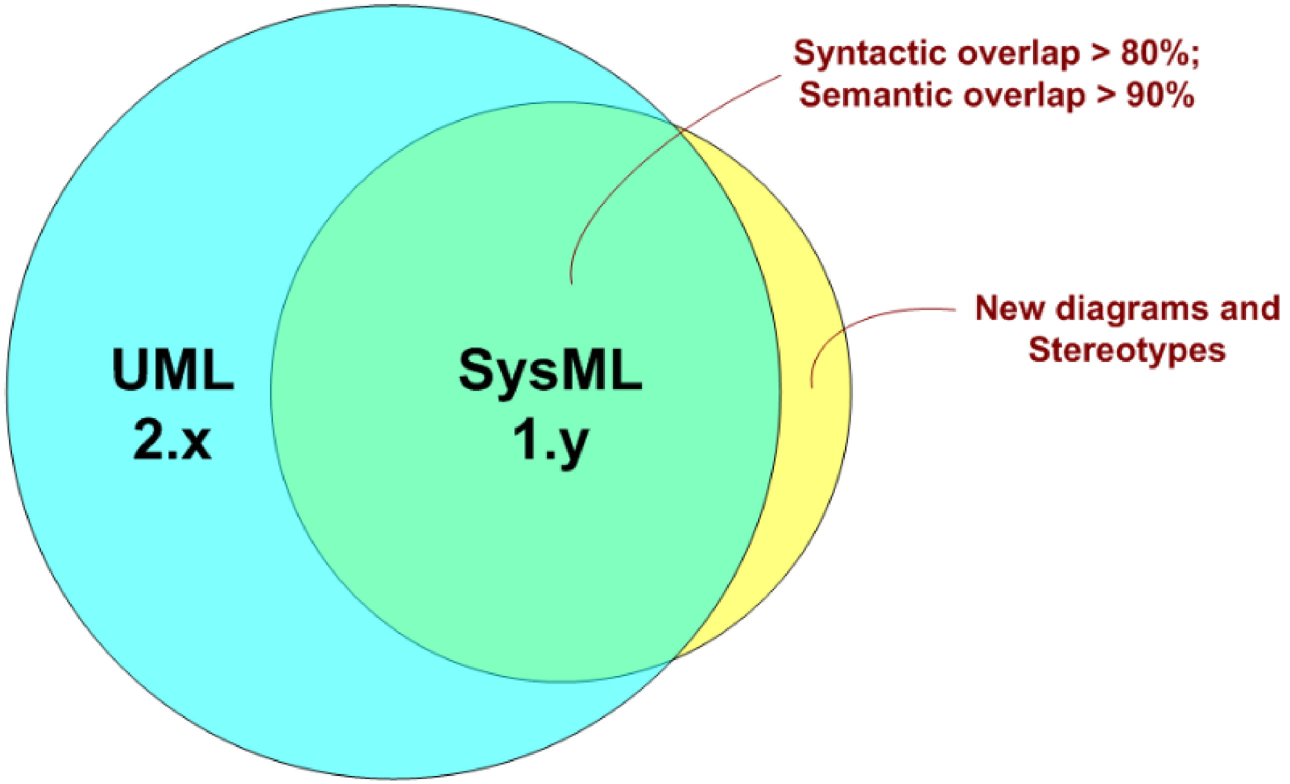
UML is a **general-purpose** visual modeling language for software engineering. A set of diagrams, intended to specify, visualize, construct, and document the **artifacts of a software system**.

The UML is an **object-oriented** modeling language which takes an object-oriented approach to the modeling of **applications**. UML focuses on a **standard language** but not a standard process, which reflects what happens in reality.

BPMN is a graphical illustration of a **business process** model with the goal of providing a notation that is easily understandable by **all business users**.

The BPMN, takes a **process-oriented** approach to modeling the **systems**. It focuses on **business processes** without covering other aspects of an organization and it covers only the description of the notation's elements.

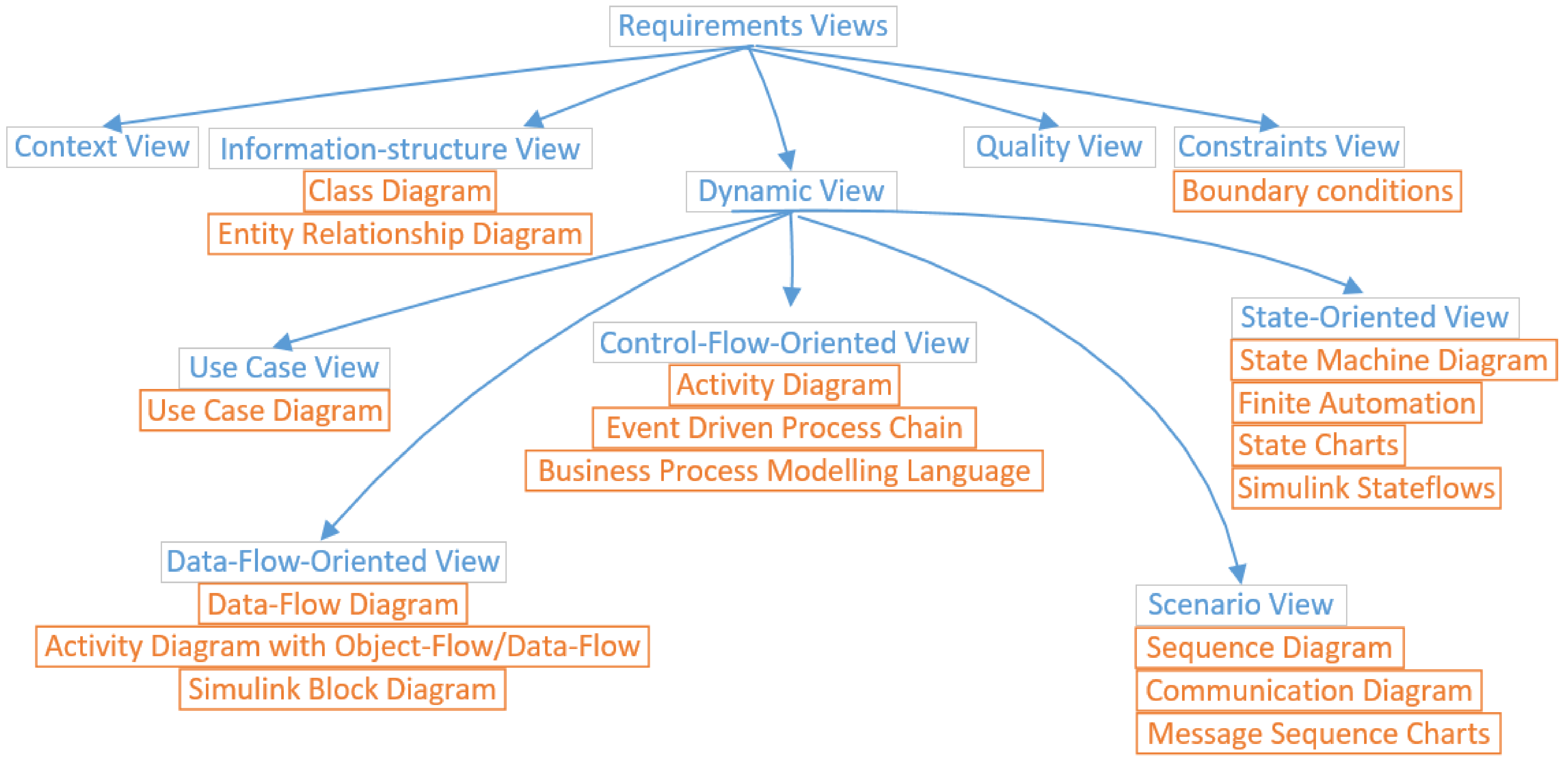
SysML - Systems Modelling Language is a general-purpose modeling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. SysML is defined as an extension of a subset of the (UML)



Common diagrams: Activity Diagram, Class Diagram (Block definition), Composite Structure Diagram (Internal Block), Sequence Diagram, State Machine Diagram, Use Case diagram, Package Diagram
SysML specific diagrams: Requirement Diagram, Parametric Diagram

VIEWS IN REQUIREMENTS MODELLING

- During requirements management various **views** can be created in order to **represent different perspectives** of the same functionality.
- Views can be defined to address **specific concerns** of stakeholders.
- A **user view** can be defined to represent the **user's perspectives** over the system. Persona and product champions are invaluable at this stage. This view models only those requirements that directly affects the way the system will be used
- Can be created views related to maintenance or to various **NFRs**
- Various "philosophies" for establishing views can be applied in combination to **control the scope and complexity** of requirements modeling.
- Through common concepts or map-ping relationships, the requirements models of the different views can then be integrated into an **overall model**.


Requirements Views
Context View
Information-structure View
Class Diagram
Entity Relationship Diagram
Dynamic View
Quality View
Constraints View
Boundary conditions
Use Case View
Use Case Diagram
Control-Flow-Oriented View
Activity Diagram
Event Driven Process Chain
Business Process Modelling Language
State-Oriented View
State Machine Diagram
Finite Automation
State Charts
Simulink Stateflows
Data-Flow-Oriented View
Data-Flow Diagram
Activity Diagram with Object-Flow/Data-Flow
Simulink Block Diagram
Scenario View
Sequence Diagram
Communication Diagram
Message Sequence Charts

Context View – understanding the context of the system
- the knowledge of what systems are related to the system under development
- properties of external systems
- which roles, people interact with the system
- which properties of the interacting roles are relevant for the system.
- identify the necessary interfaces between the system under development and its context.
- Typically is used the Context Diagram

Information Structure View – representing the static structural information
- static and structural aspects of the functionality,
- the structure of data to be processed by the system.
- Typical, but not exhaustive diagram types are class diagrams or various dialects of entity-relationship diagrams

Dynamic View – representing dynamical aspects of the system

- dynamic aspects of the functionality.
- behavioral models of the system
- chronological-logical relationships in the required behavior of the system
- divided in other views:
  - Use Case view
  - Data-Flow view
  - Control-Flow view
  - Scenario view
  - State view
- Typical, but not exhaustive diagram types used:
  - use case diagrams,
  - activity diagrams,
  - state machine diagrams,
  - data flow diagrams,
  - sequence diagrams.

Use Case View - user functions and dependencies to the system context

- high-level system structure
- user functions and their relationships to actors in the system context
- functionality that the system must offer for an actor within the context to gain a benefit.
- Use case diagrams are typically used

Data Flow-Oriented View - system functions and data dependencies

- the functions that manipulate date from the system interface
- the data dependencies between functions
- data dependencies with actors in the system context
- the functions can be analysed at various levels of granularity
- typical diagrams used are:
    - data flow diagrams,
    - activity diagrams that focus on the object flow between actions.

Control Flow-Oriented View - process flow logic

- the processes (activities, actions) conducted from the system interface
- processes flow logic
- the control flow relationships are considered in the form of sequential, alternating, or concurrent sequences
- typical diagrams:
  - activity diagram,
  - event-driven process chains
  - BPMN process diagrams

State-Oriented View - states and state changes

- the required state space of the system
- the reactive behavior of the system in relation to the system context
- state changes observable at the system interface
- state changes between the system and the system context
- events triggering a state change
- typical diagrams: State Machine, Finite Automation, State Charts

Scenario View – interaction sequences between Actors and the System

- considers interactions between actors and the system
- represents the added value or business goal obtained by the actors
- scenarios are frequently used to make use cases in use case diagrams more specific
- the scenarios modeled should always lead to successful execution of a use case
- the message exchange between actors in the context of the system are
- typical diagrams:
  - Sequence Diagram
  - Communication Diagram
  - Message Sequence Charts

Quality View – modelling the NFRs

- focus on quality non-functional requirements of the system or components
- mainly represented by textual supplements or annotation to other diagrams
- can be used taxonomy of quality attributes models (studied during types of requirements course)
  - Multi-level
  - McCall's
  - Boehm's
  - ISO/IEC 25010

Constraints View – requirements in terms of boundary conditions

- external constraints
- organizational conditions
- tech constraints – design, developments
- regulatory constraints
- often documented in textual form or by textual additions in requirements models
- can be used: class diagrams, component diagrams

INTEGRATING TEXTUAL REQUIREMENTS IN THE REQUIREMENTS MODEL

Textual adding or annotations are very useful in almost any diagram.

- describe those details which are not possible to represent graphically
- declutter the diagram of not-so-relevant elements
- add details for stakeholders which are not very involved

SysML has the **requirements diagram** which is assigned to neither the structure view nor to the behavior view

- used for modelling textual requirements
- allows the modeling of relationships between textual requirements and model elements of SysML diagrams
- used to include predetermined requirements in the requirements model (ex. specific requirements from a business domain)

Most commercially available **UML** tools offer the possibility of using textual requirements in any diagram type.

- allows the specification of textual requirements to be included in diagrams, so that they are expressed in the best possible way.

  Ex: an action in a flow can be refined through a number of textual requirements which are then included in the requirements model and related to this action (by means of an appropriate tracing relationship, for example).

- integrating textually specified requirements allows us to specify quality requirements that relate to a specific action

  Ex: requirements concerning the performance of an action are represented as textual requirements by placing them in a relationship with the action within the diagram.

DEPENDENCIES

DOCUMENTING DEPENDENCIES BETWEEN MODEL ELEMENTS

- Different requirements are modeled in different diagrams and the same requirement can have perspectives modeled in different diagrams.
- The dependencies should be registered thoroughly in order to easily navigate through models and to avoid gaps and overlaps.
- UML and SysML offer explicitly defined dependency relationships
- Sometimes you'll have to choose the dependencies stereotype function of the application domain or the diagram type.
- Dependency relationship types:
    - Refine
    - Satisfy
    - Copy
    - And other…

REFINE

A <<refines>> B

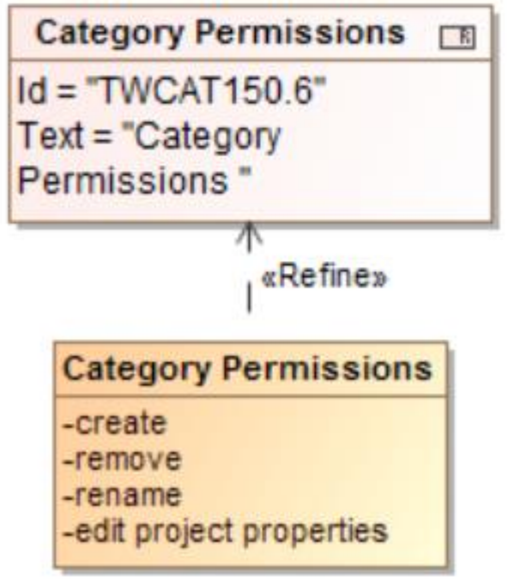
- Describes how a model element or a set of elements refine a requirement.
- A can show how B could be split to more fine grained requirements
- A adds details to B

Examples

✓ a use case or activity diagram may be used to refine a text-based functional requirement.

✓ a text-based requirement refines a model element

✓ some elaborated text could be used to refine a less fine-grained model element.



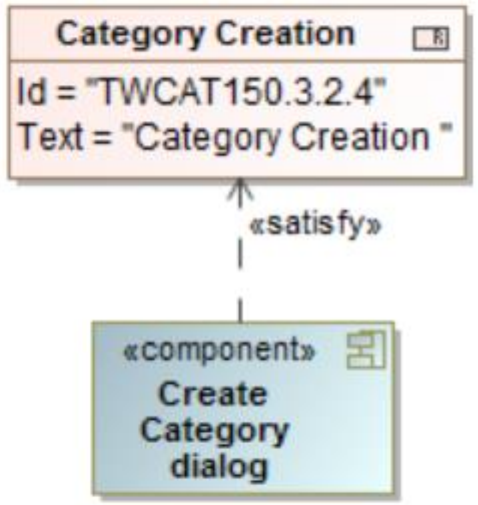
SATISFY

A <<satisfies>> B

- Describes the fact that A satisfies B
- A can be a component for B
- A fulfils the requirement from B
- A can become a list of more fine grained of B

Example

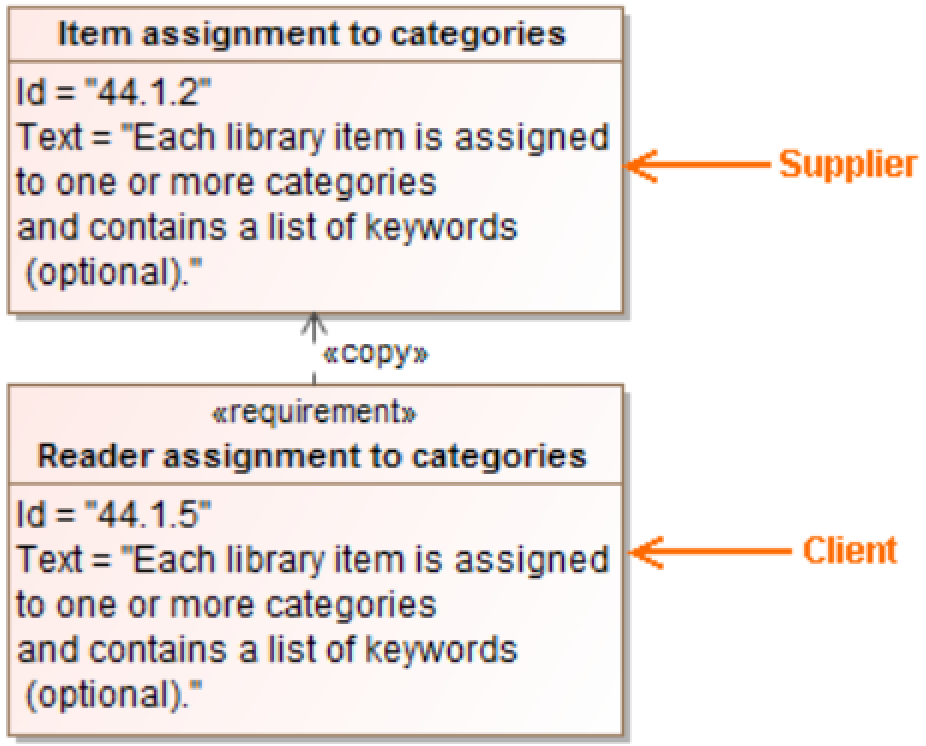
➢ When creating a category will be needed more components: Create dialog, filling in the fields, validation, accepting, declining. All these requirements satisfy the Category creation



COPY

A <<copies>> B

- Describes the fact that A is a read only copy of B
- A contains the exact details as B
- Copy dependency is useful for tracing the changes in A and B requirements. Existing tools will highlight the copy dependency when one of them is changed


Item assignment to categories
Id = "44.1.2"
Text = "Each library item is assigned to one or more categories and contains a list of keywords (optional)."
Supplier
«copy»
«requirement»
Reader assignment to categories
Id = "44.1.5"
Text = "Each library item is assigned to one or more categories and contains a list of keywords (optional)."
Client

OTHER DEPENDENCIES

Derive:

A<<derives>>B

- the derived requirement is generated or inferred from the source requirement.
- useful for splitting requirement B
- useful for impact analysis

Trace:

A<<traces>>B

- a dependency between a requirement and an arbitrary model element traced by this requirement.

Verify:

A<<verifies>>B

- a dependency between a requirement and a test case or a model element that can determine whether the system fulfills the requirement.

QUALITY OF REQUIREMENTS MODELS

THE QUALITY OF REQUIREMENTS MODELS

The quality of the requirements model, the requirements diagrams, and model elements can be assessed against three criteria:

Syntactic Quality

The syntactic quality expresses the extent to which a model element satisfies the syntax of a selected modelling language.

Ex: UML was selected as modelling language.

To assess the syntactic quality of a diagram should be examined how well the diagram meets the UML notation requirements.

Can be syntactically assessed:

- single model element (graphical or textual),
- a diagram
- entire requirements model

If appropriate modeling tools are used for modeling requirements, the **syntactic quality** of the diagrams created is usually **ensured by the tool**.

Semantic Quality

The semantic quality expresses how well a model represents the fact correctly and completely.

A semantically qualitative model does NOT have overlaps, gaps or erroneous information.

Stakeholders validation is very important for ensuring Semantic Quality.

Ex: Requirement Model for an ATM flow.

> If a an activity diagram models that after reading the card data, the customer is first asked for the payment amount, not the PIN code this represents a semantic defect, since the actual flow deviates from the diagram.

Can be semantically assessed

- single model element (graphical or textual),
- a diagram
- entire requirements model

Pragmatic Quality

The pragmatic quality expresses the extent to which a model is suitable for the selected use. It raises the question of how well the selected level of abstraction and detail represents the intended requirements

Pragmatic assessment can be performed only if the purpose and target audience of the diagram are known.

Pragmatic quality has a direct effect on the semantic quality – the completeness and correctness of the model.

Ex: A state transition diagram.

Pragmatic assessment will evaluate the level of details added. Is only the triggering event specified or are the alternate conditions described as well?

Can be pragmatically assessed

- single model element (graphical or textual),
- a diagram
- entire requirements model

CONTEXT MODELLING

PURPOSE

- During requirements engineering the system boundaries are specified and the system scope is clearly distinguished from its context.
- Understanding the context of the system is challenging
- The more complex and critical the system is, the more important it is to understand and document the context.
- Context includes:
    - knowledge about which other systems influence the system in an operational context,
    - properties of the external systems,
    - knowledge about which roles or people interact with the system in an operational context
    - which properties relevant for the system they have.
    - context modeling also helps to identify the necessary interfaces of the system

CONTEXT DIAGRAM

Studied during the Requirement Elicitation lesson

- from a requirements perspective, the context view defines the scope of a system, meaning that it draws a line between functionality **in** and **outside** the scope.
- the context view can help when considering the properties, functions and qualities of the **external systems** relevant for the system under development

The context diagram can be modelled using various notations:

- UML class diagram,
- use case diagram,
- component diagram
- tabular representation
- flow diagram
- SysML block diagram

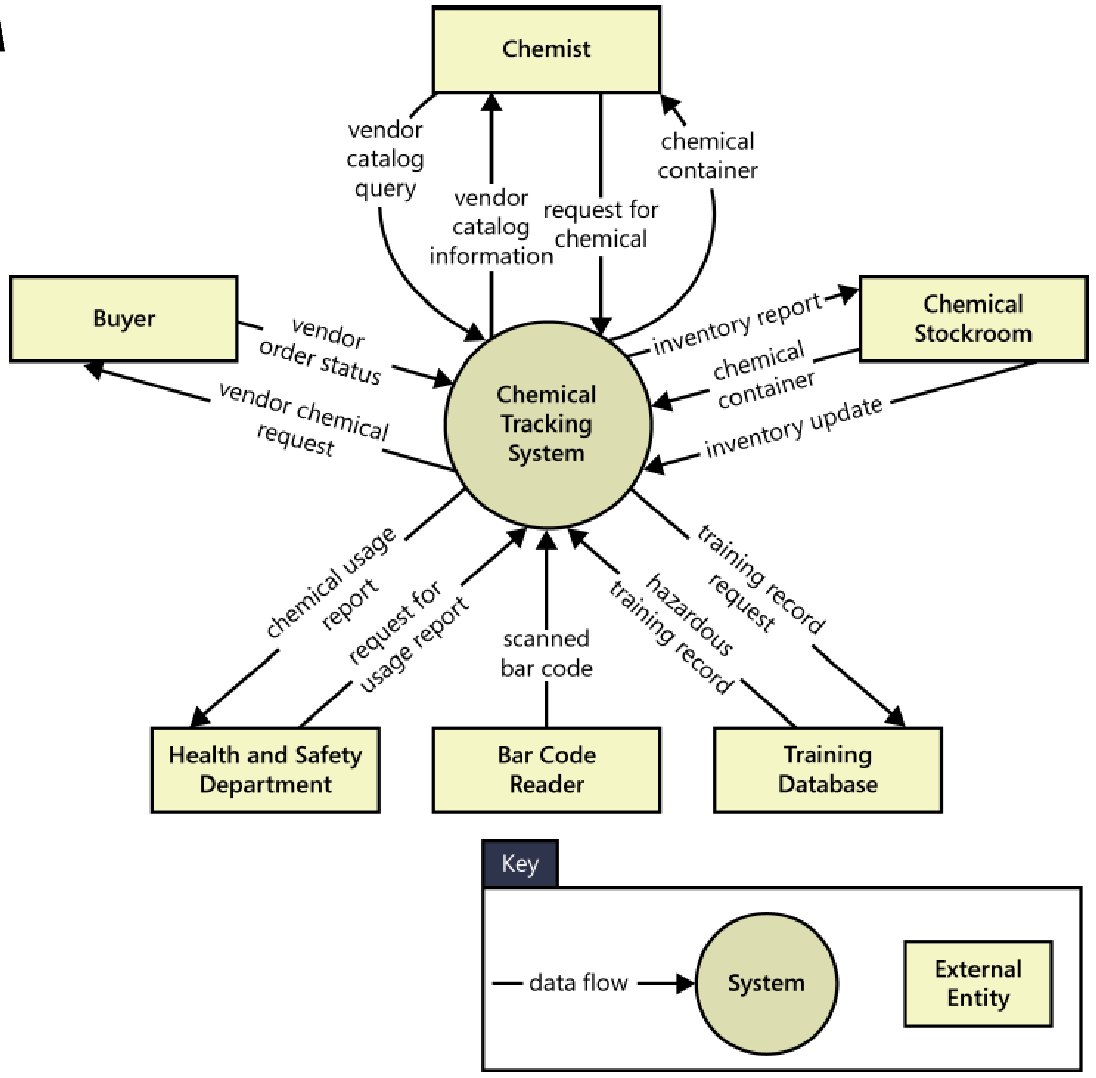
The three essential basic elements of a context diagram are:

- The system under development and its boundary
- Neighboring systems or actors
- The logical interfaces between the system and its neighboring systems

CONTEXT DIAGRAM USING DATA FLOW DIAGRAM TYPE

This example is more appropriate for the coarse grained level of requirements

Visually illustrates the boundary and connections between the system to be developed and any external entities

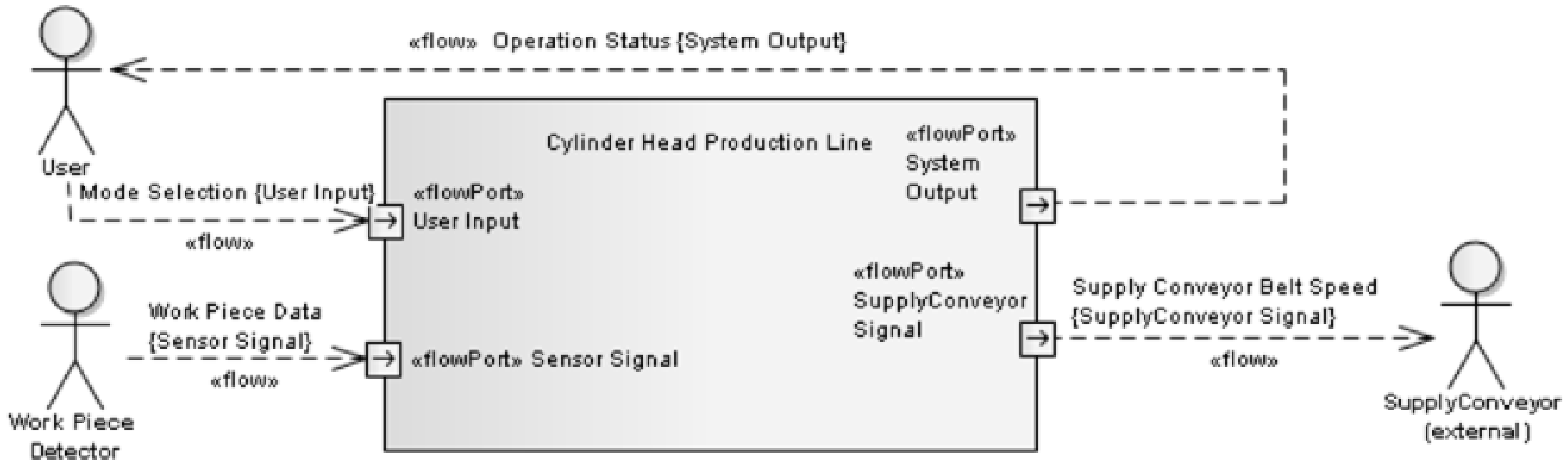


PRAGMATIC RULES FOR CONTEXT MODELING WITH DATA FLOW DIAGRAMS

The following pragmatic rules should be considered:

- All neighboring systems that interact with the system should be included in the diagram – **completeness** of the communication partners
- All neighboring systems should be **named** - to clearly specify where the input comes from and where the output goes to
- All inputs and outputs should be labeled with the logical name of the **data flows –** unnamed arrows indicate a lack of understanding of the interface

EXAMPLE OF CONTEXT DIAGRAM USING SYSML BLOCK DIAGRAM TYPE


«flow» Operation Status {System Output}
User
Mode Selection {User Input}
«flow»
Cylinder Head Production Line
«flowPort» System Output
«flowPort» User Input
«flowPort» SupplyConveyor Signal
Supply Conveyor Belt Speed {SupplyConveyor Signal}
«flow»
Work Piece Data {Sensor Signal}
«flow»
«flowPort» Sensor Signal
Work Piece Detector
SupplyConveyor (external)

The diagram shows actors in the system context and the data flows between actors and the system.

OTHER TYPES OF CONTEXT MODELING

The cooperation between the system under development and the neighboring systems in the context is also the subject of:

- the use case view – used to roughly structure the system boundaries and scoping.
- the scenario view – sequences of communication with external entities can be specified.
- state-oriented view – the state of the system context and corresponding state transitions are modeled.
- data flow-oriented view – modeling functions in the system context and documenting their relationship to functions of the system.

THANK YOU

AMDARIS