## Theoretical Concepts

### Model

A simplified or partial representation of the reality, or an abstraction, defined to accomplish a task or to reach an agreement.

### Metamodel

It can be seen as a model of a model. It's an abstraction of a model: every model has a Metamodel, which can be it-self.

### MDE

Model Driven Engineering (MDE) uses models and metamodels to represent software systems and implement part of it, by generating code from the models created. The models can also be used in the creation of prototypes and in requirements engineering.

### UML

Unified Modeling Language is a general purpose modelling language that can be used to design and visualize diagrams that portrays the behavior and structure of a system in a standard way.

The UML diagrams can be classified as: Structural Diagrams (static aspects or structure of a system) and Behavior Diagrams (dynamic aspects or behavior of the system)

### DSL

Domain Specific Language (DLS) is a programming language with a higher level of abstraction specialized to a particular application or use cases which uses the concepts and rules from the field or domain.

Making it more efficient to design an application and to find and correct errors of logic. Also allows a smoother development of the application since it's possible for non-developers and people who do not know the domain understand the overall design.

Some examples of DSL are: CSS, make and SQL

### SysML

Systems Modeling Language (SysML) is a general-purpose system architecture modeling language (General Porpose Language - GPL) for Systems Engineering applications.

SysML supports the specification, analysis, design, verification, and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities.

**MDA**

Model Driven Architecture® (MDA®) is an approach to software design, development and implementation. MDA provides guidelines for structuring software specifications that are expressed as models.

MDA separates business and application logic from underlying platform technology. Platform-independent models of an application or integrated system’s business functionality and behavior, built using UML and the other associated OMG modeling standards, can be realized through the MDA on virtually any platform, open or proprietary, including Web Services, .NET, CORBA R, J2EE, and others.

## DSL (MPS and Mbedder) vs GPL (Eclipse, Papyrus and SysML)

Eclipse and Papyrus offer a more graphical framework to develop models than the MPS with Mbeddr. Dispite that, being able to generate code and the flexibility of the DSLs gives a big advantage to MPS and Mbeddr. Both tools seem very complete, but it wasn't possible to study Papyrus as much as MPS in the scoop of this project. That being said, the main goal of MPS and Mbeddr is to create and manipulate DSLs, while Papyrus works with UML and SysML which are GPLs. So both tools have their utility in MDE.

### State Machine

A state machine is a behavior model that allows to represent the behavior and functionality of an system or application. It's, mainly, composed by: States, Transitions, Actions and Events.

The flow will always start at the initial State and given an Event (trigger or input) the system will Transition to another State and execute an Action according to the system design.

Note that a State Machine must follow some rules:

* Must have an initial State
* The system must be able to enter and leave every defined State
* In a given State a Transition can only enter one state

2.1- DSL vs GPL

GPL( General Purpose Language)

DSL(Domain Specific Language)

**Abstract Syntax**: Describes the structure of the language

**Concrete Syntax**: Describes specific representations of the Modeling Language

**Semantics**: Describes the meaning of the elements defined in the language.

Principles for DSL

* Provide good abstractions and make life easier
* Not depend on one-man expertise
* Evolution (must be update based on the user)
* Supporting Tools and Methods
* Open for extensions and closed for modifications(open-close principle)

Classification

* Focus
  + Vertical: Aimed at a specific field or industry
  + Horizontal: Broader applicability and concepts that apply across many fields.
* Style
* Declarative: Specific paradigm express logic of computation without describing its control flow(ex:SQL)
* Imperative: Define an executable algorithm that states she steps and control flow that need to be followed(ex:Java/C++)
* Notation
* Graphical:Visual models
* Textual: text notations and XML.based notations
* Internality
* Internal: host language and give it the feel of a particular domain
* External: have their custom syntax
* Executability
* Model Interpretation: executing DSL script at runtime one statement at a time
* Code Generation: complete model-to-text transformation, making a executable aplication

2.3- Developing a DSL

META Four-Layer Architecture:

-M3: Definition of EBNF

-M2:Definition of Java in EBNF

-M1:Program(in Java)

-M0:Execution of the program

Advantage of Metamodels:

* Precise
* Accessible
* Evolvable language

**Requirement** - A condition or capability needed by a user to solve a problem or achieve an objective

**Stakeholder** - A stakeholder of a system is a person or an organization that has an (direct or indirect) influence on the requirements of the system.

**Requirements Engineering** - Requirements engineering is a systematic and disciplined approach to the specification and management of requirements with the following goals:

Core activities of requirements engineering

* Elicitation
* Documentation
* Validation and negotiation
* Management

Capabilities Required:

* Analytic thinking
* Empathy
* Communication skills
* Conflict resolution skills
* Moderation skills
* Self-confidence
* Persuasiveness

Requirement Types

* **Functional requirements** - Define the functionality that the system to be developed offers.
  + Functional requirements
  + Behavioral requirements
  + Data requirements
* **Quality requirements** - Define desired qualities of the system to be developed and often influence the system architecture more than functional requirements do.
* Performance of the system
* security of the system
* reliability of functionalities
* usability of a system
* maintainability of a system
* portability of a system
* **Constraints** - Requirements of this type can constrain the system itself or the development process
  + Constraints cannot be influenced by the team members.

**System context** - the part of the system environment that is relevant for the definition as well as the understanding of the requirements of a system to be developed.

The following possible aspects of reality **influence the context of a system:**

* People (stakeholders or groups of stakeholders)
* Systems in operation (other technical systems or hardware)
* Processes (technical or physical processes, business processes)
* Events (technical or physical)
* Documents (e.g., laws, standards, system documentation)

Product satisfaction

* **Dissatisfiers** - properties of the system that are self-evident and taken for grante.
  + Must be fulfilled by the system in any case.
* **Satisfiers** - explicitly demanded system properties
  + Are properties that are consciously known to the stakeholders and explicitly demanded.
* **Delighters** - system properties that the stakeholder does not know or expect and discovers only while using the system
  + Over the time, **Delighters** turn into **Satisfiers** and finally into **Dissatisfiers.**