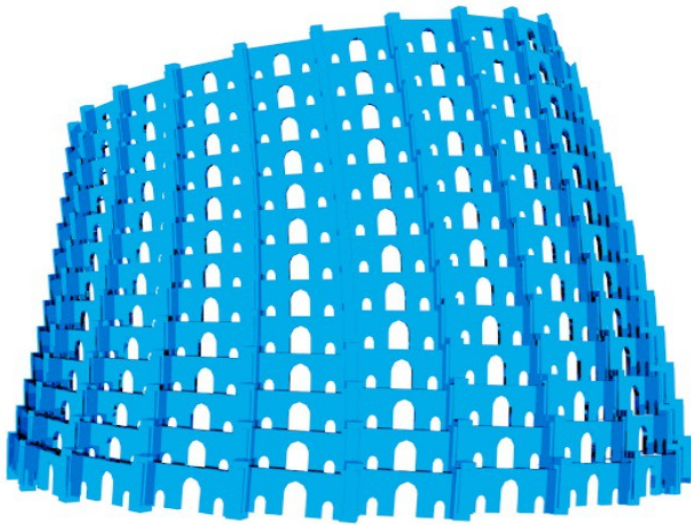


MegaL/Checker Vocabulary

A natural language description

Marcel Heinz
Software Languages Team
University of Koblenz-Landau



SOFTLANG

© 2016 Software Languages Team

MegaL

- MegaL is short for 'Megamodeling Language', where a model describes models and their relationships from a conceptual perspective.

MegaL

- Multiple implementations of multiple MegaL versions exist.
- **MegaL/Checker** is trimmed towards gathering facts and checking their well-formedness.
- MegaL/XText was implemented for validating facts in an actual system.

MegaL/Checker

- Textual syntax
- Stable, but minor evolution might happen
- Newest vocabulary diverges from the vocabulary in papers.
- There may be redundancies when stating all kinds of knowledge for a technology.

MegaL/Checker - Prelude

- The Prelude module contains all subtypes and possible relationships.
- It represents the ground truth for the vocabulary.
- It is imported automatically, when processing a new model.

MegaL/Checker - Language

- A language is a set of syntactic entities.
 - Language $<$ Entity
 - Java : Language
- For now, an instance of Language is specifically used in software development.

MegaL/Checker - Paradigm

- A programming paradigm is a concept that defines a way of thinking to have while programming in a language that supports it.
 - Paradigm < Entity

MegaL/Checker - Language

- A language is classified by the paradigms that it facilitates.
 - facilitates < Language # Paradigm
 - Java facilitates ObjectOrientation
- Besides being a way of thinking it has implications on the kinds of:
 - Semantics
 - Type System
 - Syntax

MegaL/Checker - Language

- A language is a set of syntactic entities.
 - Language < Entity
 - ~~Java : Language~~
- A language has one specific purpose.
 - Java : ProgrammingLanguage
 - XML : DataRepresentationLanguage
- A language can be a subset of another language.
 - XSD subsetOf XML
 - SQLDDL subsetOf SQL

MegaL/Checker - Artifact

- An artifact is a digital entity.
 - Artifact < Entity
- An artifact is further classified by a purpose.
 - Specification < Artifact
 - Value < Artifact
 - SyntaxDefinition < Artifact
- An artifact is element of a language.
 - ?models.py elementOf Python

MegaL/Checker - Artifact

- A manifestation describes the shape of an artifact at runtime.
 - Manifestation < Entity
 - File < Manifestation
 - Transient < Manifestation
- An artifact has a manifestation.
 - manifestsAs < Artifact # Manifestation
 - ?models.py manifestsAs File
 - ?schemaRequCmd manifestsAs Transient

MegaL/Checker - Artifact

- An artifact can define a language.
 - defines `< Artifact # Entity`
 - Java8Spec defines Java
 - FSMLGrammar defines FSML
- An artifact may be conform to another.
 - conformsTo `< Artifact # Artifact`
 - `?anXMLFile conformsTo ?anXSDFile`
 - `?aJavaObject conformsTo ?aJavaClass`

MegaL/Checker - Artifact

- An artifact can correspond to another in the sense that it is only syntactically different.
 - `correspondsTo < Artifact # Artifact`
 - `?aJavaObject correspondsTo ?anXMLFile`
 - `?aJavaClass correspondsTo ?anXSDFFile`

MegaL/Checker - Pattern

- A design pattern describes a reusable structure that addresses maintainability on the level of code.
 - DesignPattern < Entity
 - Subject-Observer : DesignPattern
- An architectural style describes a reusable structure that addresses maintainability on the level of components.
 - ArchitecturalStyle < Entity
 - Client-Server : ArchitecturalStyle

MegaL/Checker - Role

- A design pattern or an architectural style may describe a set of participants, namely Roles.
 - Role < Entity
 - participantOf < Role # DesignPattern
 - participantOf < Role # ArchitecturalStyle
- In the end an artifact plays a role in a system.
 - hasRole < Artifact # Role
 - ?models.py hasRole MvcModel

MegaL/Checker - Function

- A function defines a mapping between an input and an output, which are elements of some language.
 - $\text{Function} < \text{Entity}$
- A function has a specific syntax.
 - `serialize : JsonObject -> XML`
 - `cutBy : XML # Int -> XML`
 - `totalAndCount : XML -> Int # Int`

MegaL/Checker - Function Application

- A function application maps input to output.
 - `serialize(?aJavaObject)|->?anXMLFile`
 - `cutBy(?company1, 3)|-> ?company2`
 - `totalAndCountEmpl(?company)|-> (12000, 5)`

MegaL/Checker - Function

- An artifact may implement a function
 - implements < Artifact # Function
 - ?CutClass implements cut

MegaL/Checker - Abstract Process

- An abstract process is a specific kind of conceptual entity that represents a process that is independent from a technology or languages.
 - `AbstractProcess < Entity`
 - `Serialization : AbstractProcess`
- An artifact may realize such a process.
 - `realizes < Artifact # AbstractProcess`
 - `JAXBSerializer realizes Serialization`

MegaL/Checker - Technology

- A technology provides reusable functionality for many distinct application scenarios.
 - Technology < Entity
- A technology is classified by its purpose.
 - Compiler < Technology
 - WebAppFramework < Technology
 - JavaC : Compiler
 - Django : WebAppFramework

MegaL/Checker - Technology

- A technology can implement a function or an abstract process.
 - implements < Technology # Function
 - implements < Technology # AbstractProcess
 - JAXB implements serialize
 - JavaC implements Compilation
- A technology can implement a language in the sense that it is able to process it.
 - implements < Technology # Language
 - JavaC implements Java

MegaL/Checker - Technology

- A technology can use another technology in the sense that it has parts that refer to the other technology.
 - $\text{uses} < \text{Technology} \# \text{Technology}$
 - Hibernate uses JDBC
- A technology uses a language in the sense that some part is implemented in the language.
 - $\text{uses} < \text{Technology} \# \text{Language}$
 - Hibernate uses Java

MegaL/Checker - Technology

- A technology facilitates the use of a design pattern or architectural style or abstract process, in the sense of a deferred usage.
 - facilitates < Technology # DesignPattern
 - Django facilitates Model-View-Controller
 - facilitates < Technology # ArchitecturalStyle
 - ?
 - facilitates < Technology # AbstractProcess
 - ANTLR facilitates Parsing

MegaL/Checker - Technology

- A technology's implementation may use a design pattern or an architectural style
 - uses < Technology # DesignPattern
 - ?
 - uses < Technology # ArchitecturalStyle
 - ?

MegaL/Checker - TechnologySpace

- A technological space is a conceptual entity that describes a set of:
 - application scenarios.
 - software languages.
 - programming tools such as IDEs
 - technologies
 - knowledge corpora
 - conferences and communities

MegaL/Checker - TechnologySpace

- A technological space is a conceptual entity.
 - TechnologySpace < Entity
 - GrammarWare : TechnologySpace
 - JavaWare : TechnologySpace
- A technology can belong to a technological space.
 - belongsTo < Technology # TechnologySpace
 - JAXB belongsTo JavaWare
 - ANTLR belongsTo GrammarWare

Be careful here! It gets difficult to explain such relationships

MegaL/Checker - Domain

- A programming domain represents a specific kind of use for technologies and languages.
 - ProgrammingDomain < Entity
 - BusinessProgramming : ProgrammingDomain
 - ProgrammingEducation : ProgrammingDomain
- A technology supports a programming domain.
 - supports < Technology # ProgrammingDomain
 - SAPNetWeaver supports
BusinessProgramming

Be careful here! It gets difficult to explain such relationships

© 2016 Software Languages Team

MegaL/Checker - System

- Represents a set of artifacts in an actual technology usage scenario.
 - System < Entity

MegaL/Checker - Usage

- A system or artifact can use a system, technology, design pattern, architectural style, abstract process or language.

MegaL/Checker - Parthood

- There exist various partOf relations
 - $\text{partOf} < \text{Artifact} \# \text{Artifact}$
 - $\text{partOf} < \text{Artifact} \# \text{Technology}$
 - $\text{partOf} < \text{Artifact} \# \text{System}$
 - $\text{partOf} < \text{Technology} \# \text{Technology}$
 - $\text{partOf} < \text{System} \# \text{System}$

MegaL/Checker - Artifact

- Syntactic sugar for instanceOf, elementOf, hasRole and manifestsAs in one line :
 - ?models.py : Artifact<Python,MvcModel,File>
 - This may not always be feasible, since there may be more than one role.

MegaL/Checker - Abstraction

- Instances concerned with general facts need to be linked to describing resources.
 - Django = „<https://www.djangoproject.com/>“
- Artifacts that should exist in any usage scenario do not need to be linked.
 - ?models.py : Artifact<Python,MvcModel,File>

MegaL/Checker - Abstraction

- When describing a non-abstract usage scenario, artifacts need to be linked as well.
 - ContributionsController =
`"https://github.com/101companies/101rails/blob/326a894e38b164c1f1508a73b1954ff807e27cf3/app/controllers/contributions_controller.rb"`

Constraints

- Are implemented in the Checker and are stated in natural language here :

<https://github.com/softlang/megalib/blob/master/checker/Constraints.txt>