# 14. Structure of M2 in a Technical Space (Language Families and Composition of Tools)

Prof. Dr. U. Aßmann

Technische Universität Dresden

Institut für Software- und Multimediatechnik

http://st.inf.tu-dresden.de /teaching/most

Version 17-0.6, 23.10.17

1) Problem of Tool Composition
2) Data definition languages
3) Query languages
4) Constraint languages
5) Reuse languages
6) Transformation and Restructuring languages
7) Behavior specification languages
8) Language families in several technical spaces
9) .. and all together now...

# Obligatory Literature

- http://en.wikipedia.org/wiki/List_of_UML_tools
- [HRC-MM] The SPEEDS project. Deliverable D.2.1.5. SPEEDS L-1 Meta-Model, Revision: 1.0.1, 2009
    - http://speeds.eu.com/downloads/SPEEDS_Meta-Model.pdf
- [HRC-Kit] The SPEEDS project. SPEEDS Training Kit.
    - http://www.speeds.eu.com/downloads/Training_Kit_and_Report.zip
    - Training_Kit_and_Report.pdf: Overview
    - Contract-based System Design.pdf: Overview slide set
    - ADT Services Top level Users view.pdf: Slide set about different relationships between contracts

# References

- [Vered Gafni] Presentation Slides about the Heterogeneous Rich Component Model (HRC).
- [CSL] The SPEEDS Project. Contract Specification Language (CSL)
    - http://www.speeds.eu.com/downloads/D_2_5_4_RE_Contract_Specification_Language.pdf
- G.Gößler and J.Sifakis. Composition for component-based modeling. Science of Computer Programming, 55(1-3):161–183, 2005.

# Other Literature

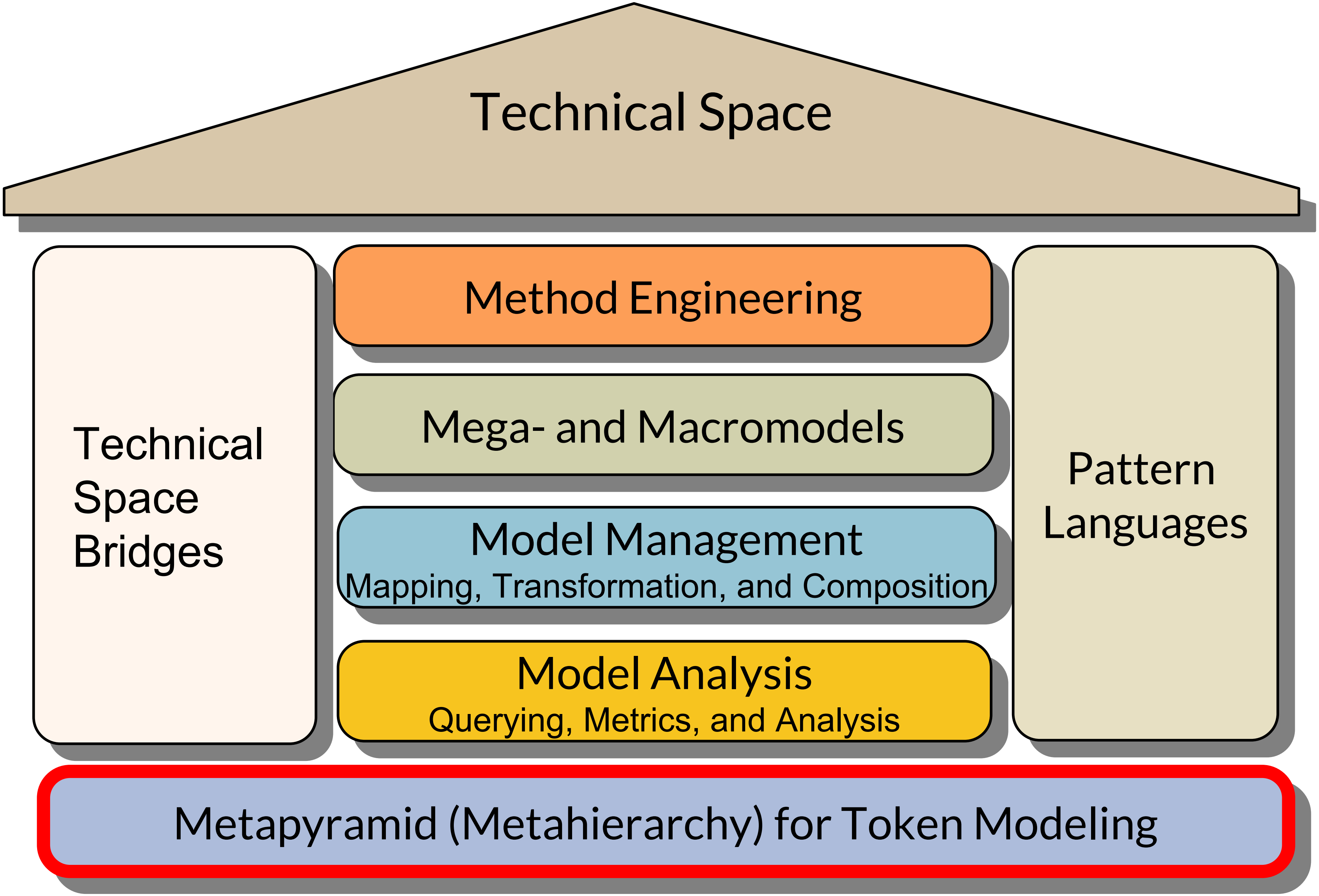
- Informatik Forum http://www.infforum.de/
- Data-Flow Diagrams:
  - De Marco, T.: Structured Analysis and System Specification; Yourdon Inc. 1978/1979. Siehe auch Vorlesung ST-2
  - McMenamin, S., Palmer, J.: Strukturierte Systemanalyse; Hanser Verlag 1988
- Workflow languages:
  - ARIS tool (IDS Scheer, now Software AG)
    - http://en.wikipedia.org/wiki/Architecture_of_Integrated_Information_Systems
- Big CASE IDE
  - MID Innovator (insbesondere für Informationssysteme)
  - http://www.modellerfolg.de/
  - MagicDraw http://www.nomagic.com/

# Objectives

- Understand language families and their relationship to M2 layers
- Understand the need to compose them for MDSD tool chains, CASE tools

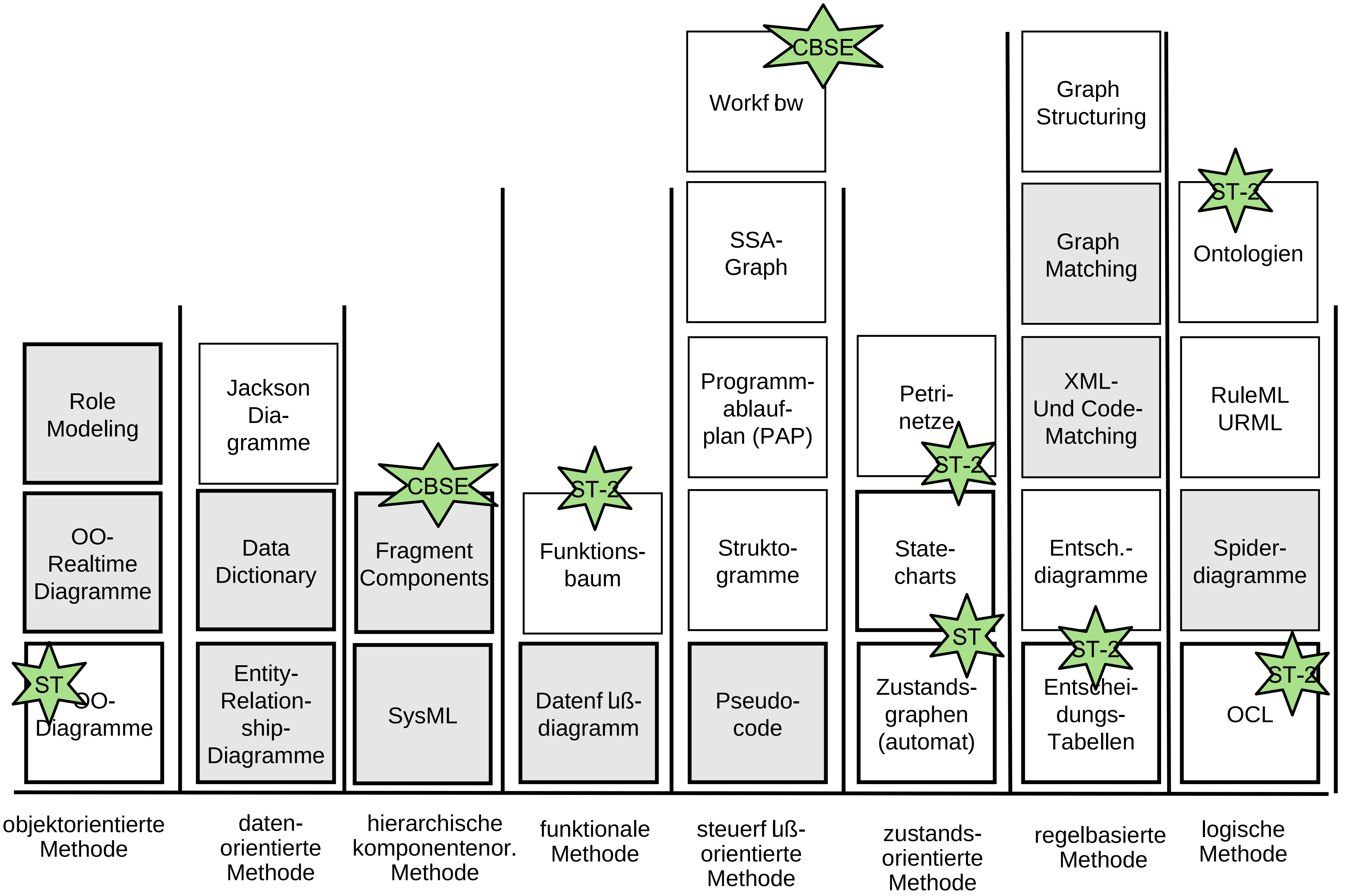
# 14.1 Basic Techniques of Software Engineering, Language Families, and Tool Composition

# Q10: The House of a Technical Space



# Basic Techniques and Languages for Modeling

Derived from [Balzert]



objektorientierte Methode
- ST OO-Diagramme
- OO-Realtime Diagramme
- Role Modeling

daten-orientierte Methode
- Entity-Relationship-Diagramme
- Data Dictionary
- Jackson Diagramme

hierarchische komponentenor. Methode
- SysML
- CBSE Fragment Components

funktionale Methode
- Datenflußdiagramm
- ST-2 Funktionsbaum

steuerfluß-orientierte Methode
- Pseudocode
- Struktogramme
- Programmablaufplan (PAP)
- SSA-Graph
- CBSE Workflow

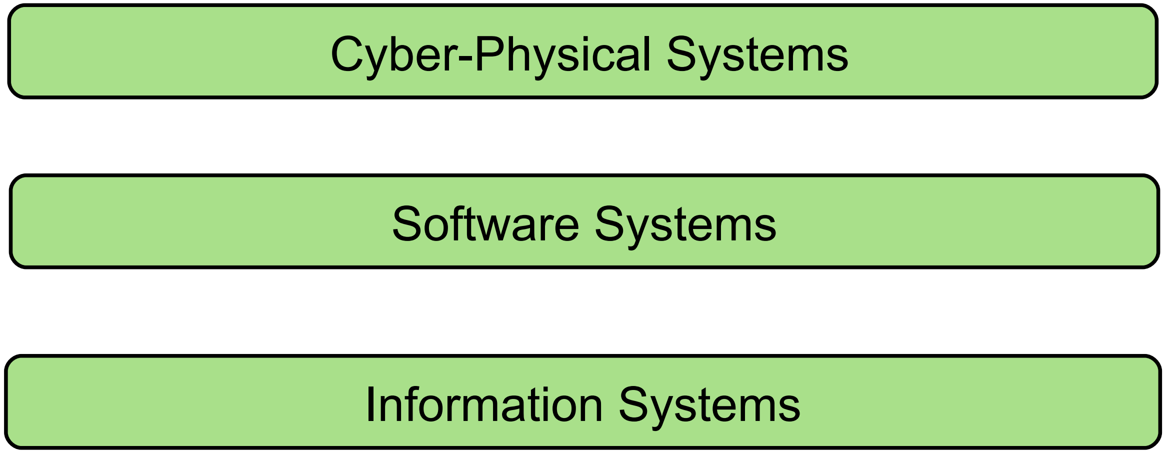
zustandsorientierte Methode
- Zustandsgraphen (automat)
- ST Statecharts
- ST-2 Petrinetze

regelbasierte Methode
- ST-2 Entscheidungs-Tabellen
- Entsch.-diagramme
- XML- Und Code-Matching
- Graph Matching
- Graph Structuring

logische Methode
- ST-2 OCL
- Spiderdiagramme
- RuleML URML
- ST-2 Ontologien

# Metamodel Layering for Language Families

- M2 can systematically be divided into **M2 layers**
- The layers contain metamodel packages, which can be varied so that **language families** result:
    - Language engineering by composition
    - Tool construction by composition
    - Basic technique composition from several languages
    - Method engineering by method composition of basic techniques
- Productivity of Process
- Reliability of Software
- Different forms of Systems need different M2 layers:



Cyber-Physical Systems

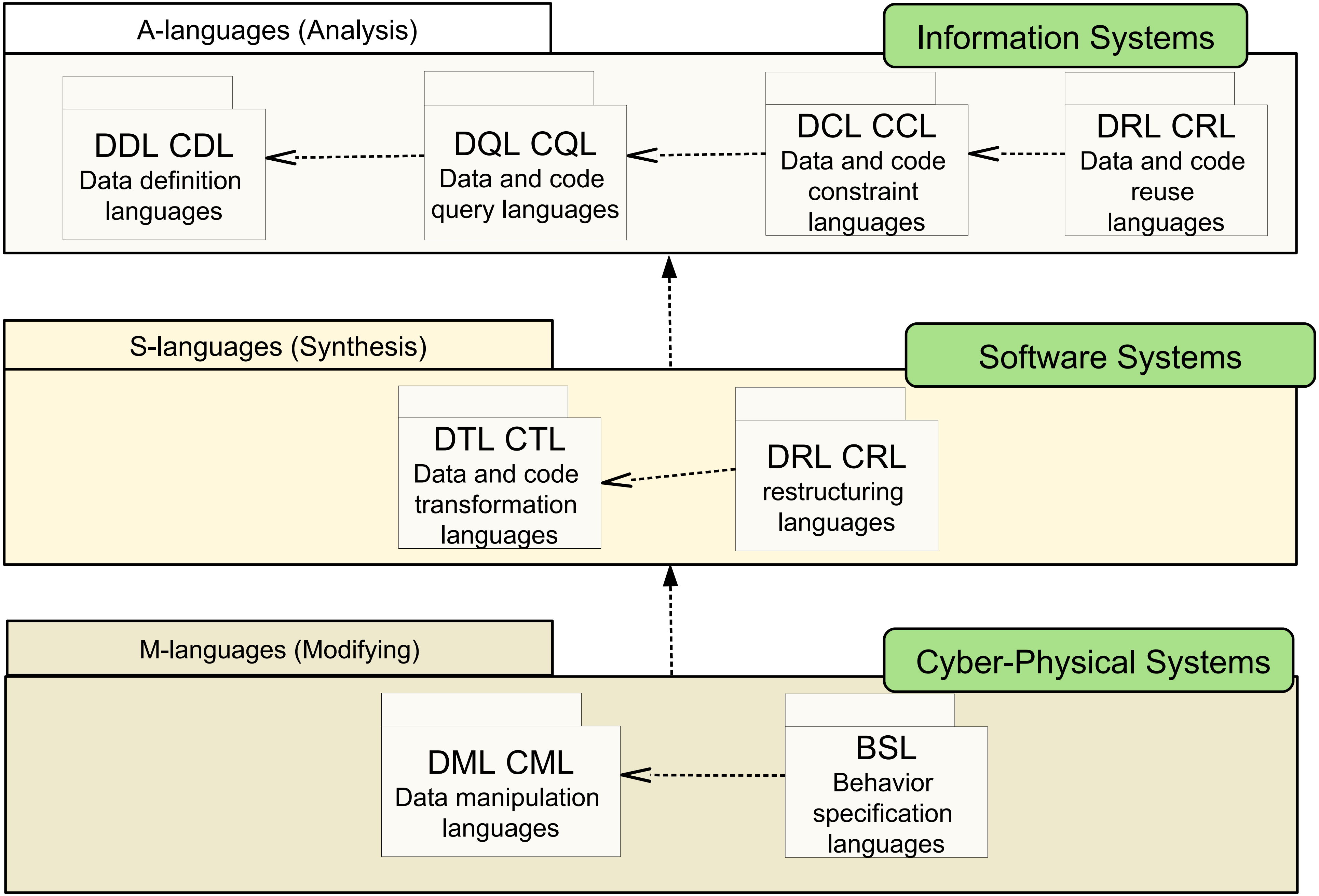
Software Systems

Information Systems

# CPS and  Simulation

Public domain
https://cdn.pixabay.com/photo/2017/05/14/20/11/simulator-2312973_960_720.jpg

# Basic Language Families (Layer Structure of M2)



A-languages (Analysis)

Information Systems

DDL CDL
Data definition languages

DQL CQL
Data and code query languages

DCL CCL
Data and code constraint languages

DRL CRL
Data and code reuse languages

S-languages (Synthesis)

Software Systems

DTL CTL
Data and code transformation languages

DRL CRL
restructuring languages

M-languages (Modifying)

Cyber-Physical Systems

DML CML
Data manipulation languages

BSL
Behavior specification languages

# Basic Language Families (Structure of M2)

- In the metahierarchy, code covers M3-M0, because M0 is populated by objects of the dynamic semantics
    - Data does not have dynamic semantics, so it only covers M3-M1 (or M2-M0); however, when data is loaded as code, it changes its nature.
- **Data and code modeling with definition languages (DDL, CDL**)
    - DDL form the basci packages of M2 to be imported by all other packages
    - Ex.: MOF → UML-CD → UML-Statecharts
    - Ex: lifted metamodels, such as EBNF-Grammars, Relational Schema (RS), Entity-Relationship-Diagrams (ERD), UML-CD, SysML-Component diagrams
- **Analysis languages (A-languages):**

Information Systems

- 
    - Queries with **query languages** (DQL, CQL)
    - Consistency checking with data and code **constraint languages** (DCL, CCL) on wellformedness of data and code
    - **Reuse languages: Contract languages and composition languages**
        - Architectural description languages (ADL)
        - Template-Sprachen (template languages, TL) → course CBSE

# Basic Language Families (Structure of M2) (ctd.)

Software Systems

- **Synthesis languages** (S-languages)
  - **Declarative Transformation Languages** (DTL, CTL)
    - Data flow diagrams (DFD)
    - Term- und graph rewrite systems
    - XML transformation languages
  - **Restructuring** (data and code restructuring languages, DRL, CRL)
    - **Wide Spectrum Languages** for refinement **(broadband languages, Breitbandsprachen)**
    - **Data exchange languages** (data exchange languages)

Cyber-Physical Systems

- Data and State **Manipulation Languages** (M-languages)
  - (non-declarative) Data manipulation languages (DML)
    - Workflow Languages, Petri Nets
    - Imperative languages
- Languages for **behavior specification language** (BSL)
  - Action-based state transition systems (finite automata and transducers)
  - Condition-Action-languages, Event-Condition-Action-languages (ECA)
  - → course Softwaretechnologie-2

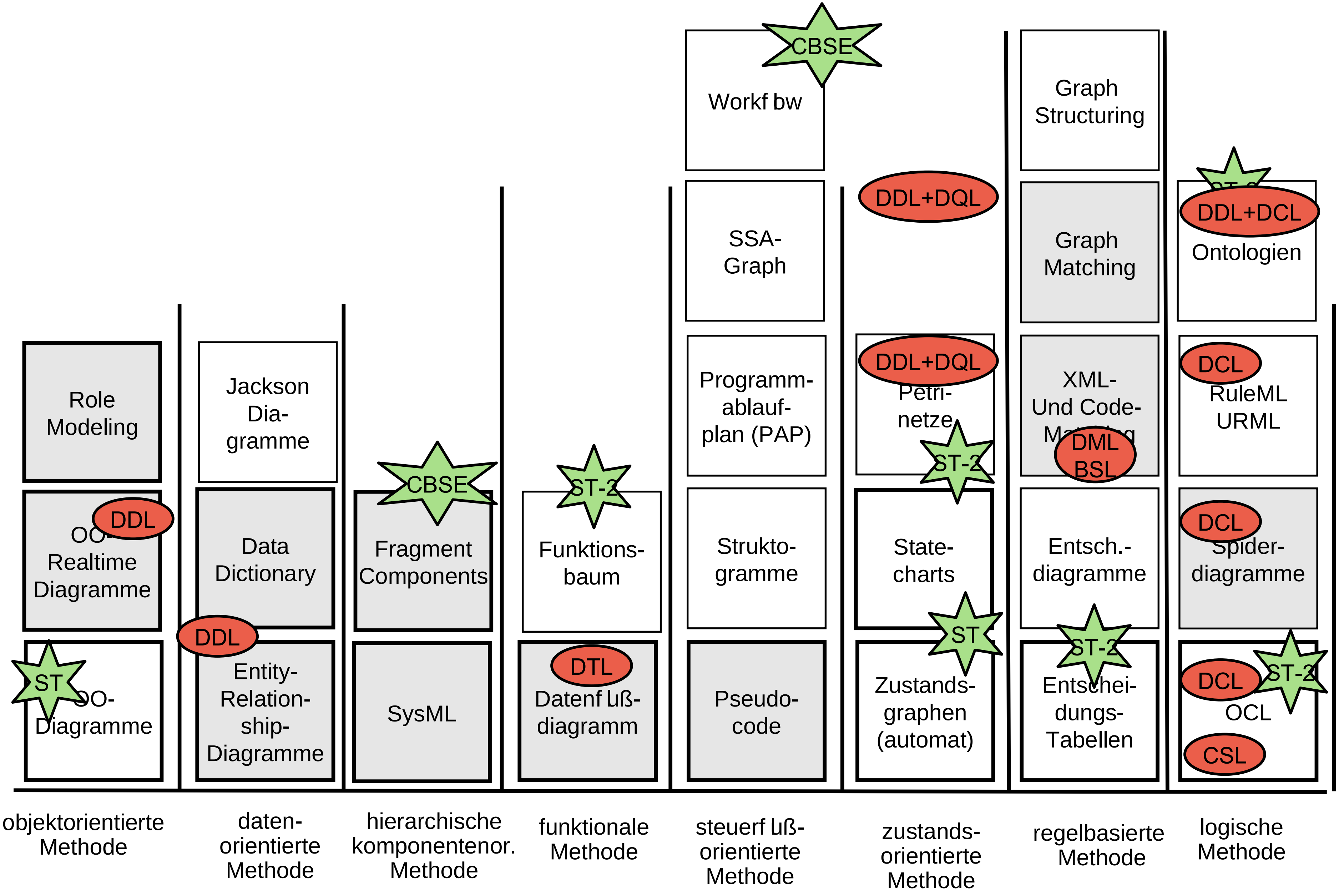
# Software Engineering of Heterogeneous Systems

- A complex MDSD tool chain or Software IDE uses many base techniques and languages
- There is no homogeneous software construction

How to compose tools for base techniques
to a large MDSD tool chain
or CASE IDE?

# Basic Techniques and Languages for Modeling in this Course and Elsewhere

Derived from [Balzert]



# 14.2 Data Definition Languages (DDL) and Code Definition Languages (CDL)

The basic layer of M2

Usually lifted to M3 (i.e., self-descriptive)

All materials are shaped by a DDL or CDL

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

# Data Dictionaries (Data Catalogues) as Basis for all Tools and IDE

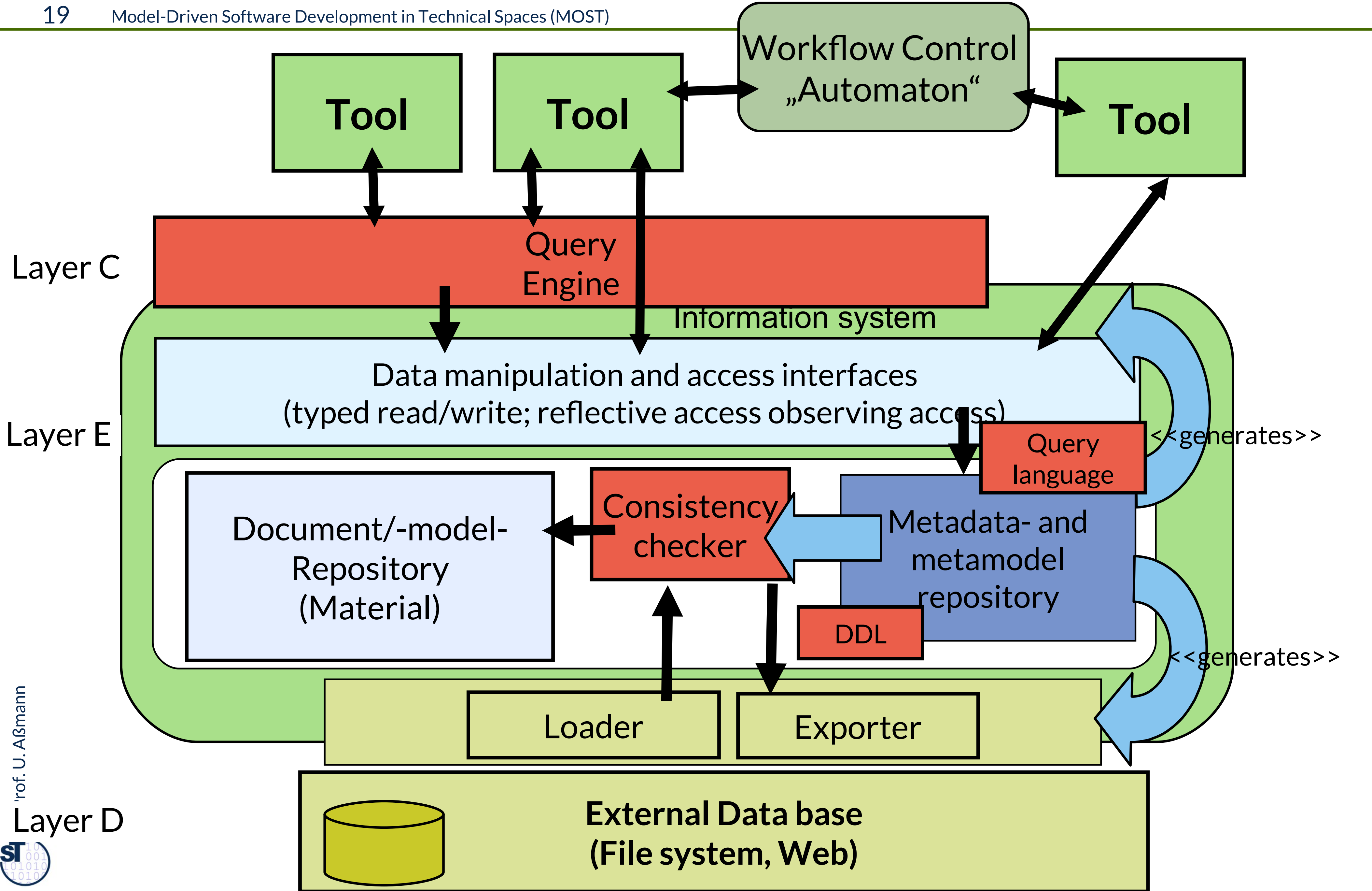
- A **data dictionary (data schema)** contains all types of data flowing through a system, including those stored in a repository (on M1)
  - Scope: local for an application, for several applications, for an entire company or even for a supply chain
  - A data dictionary is a special kind of model repository
  - If the data are models, it is called **metamodel repository**
- A **homogeneous data dictionary** is specified in a DDL
  - EBNF definies text languages (sets of text types)
  - Relational Schema (RS) defines relations and tabels
  - XML Schema (XSD) defines tree languages
  - ERD or UML-CD define graph languages
- A **heterogeneous data dictionary** is specified in several DDL
  - Usually, MDSD tool chains and Software IDE maintain heterogeneous metamodel repositories

# Information Systems are based on DDL

- An **information system** is a software system conducting data analysis about a repository
    - Data warehouses, business intelligence, data analytics
- A **stream-based information system** is a software system conducting data analysis on a set of data streams

- Every software tool, every IDE relies on an information system
    - maintaining artefacts (data, programs, models, documents)
    - giving information about them
    - typed by the types in a data dictionary
- The data dictionary is described in a **data definition language (DDL)** or **code definition language (CDL)**
- The repository and the data streams are queried and analyzed by A-languages

# Q7: Tool Architecture with Data Sharing in a Metamodel-Driven Repository


Tool
Tool
Workflow Control „Automaton“
Tool
Layer C
Query Engine
Information system
Data manipulation and access interfaces (typed read/write; reflective access observing access)
Layer E
Query language
<<generates>>
Document/-model-Repository (Material)
Consistency checker
Metadata- and metamodel repository
DDL
<<generates>>
Loader
Exporter
Layer D
External Data base (File system, Web)

# 14.3 Query Languages (QL)

DQL – Data Query Languages

CQL – Code Query Languages

All materials are queried by technical tools shaped by a DQL or CQL.

# DQL and CQL

- Querying
  - Pattern matching of structural patterns
  - Joining information
  - Reachability queries
- Metrics : counting of patterns
- Analysis: Deeper knowledge (implicit knowledge)
  - Program and model analyses on value and type flow

# 14.4 Constraint Languages (DCL,CCL) for Consistency Checking

All materials are constraint-checked by techncial tools shaped by a DCL or CCL.

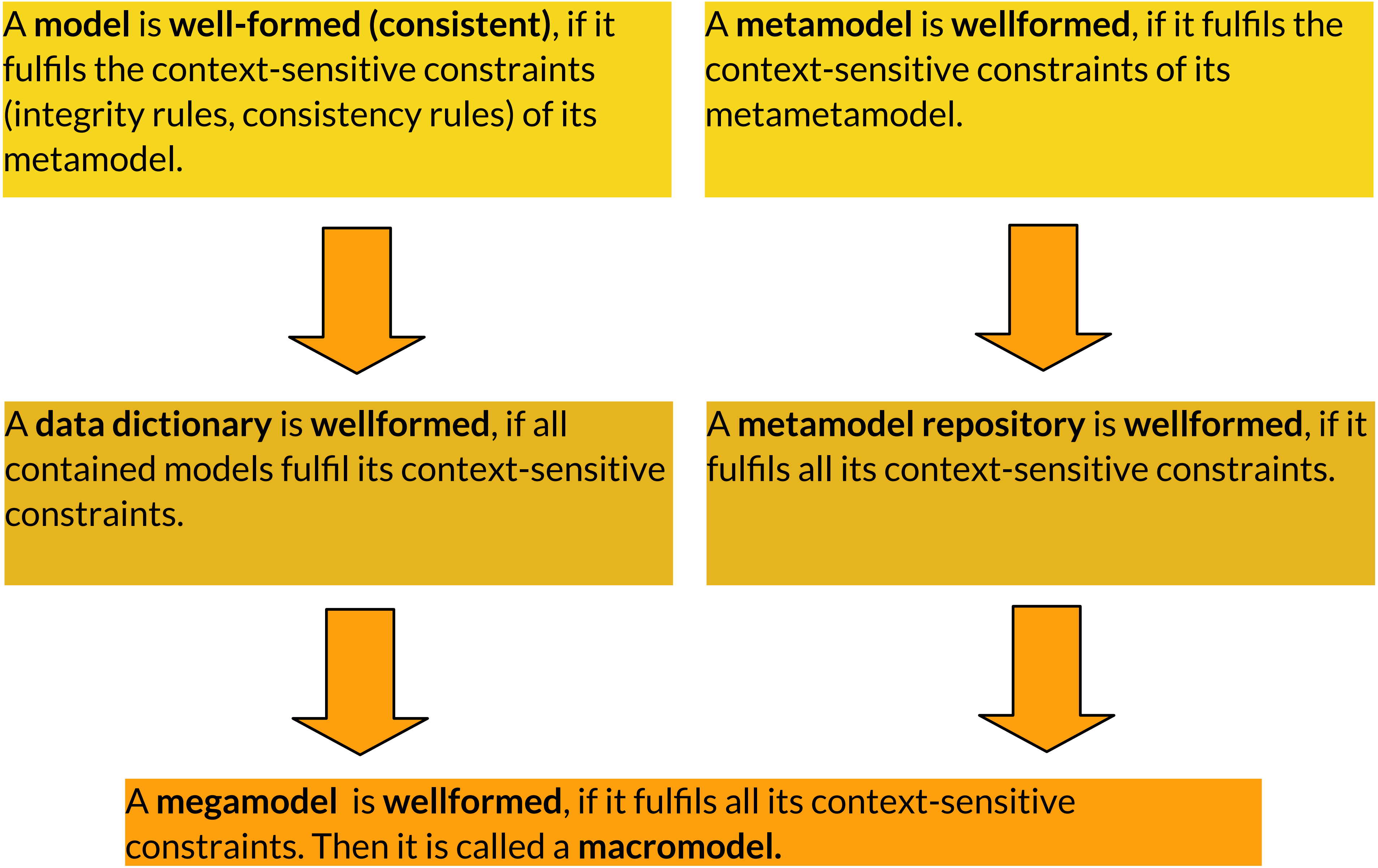
DRESDEN concept Exzellenz aus Wissenschaft und Kultur

# Well-formedness of Models

A model is **well-formed (consistent)**, if it fulfils the context-sensitive constraints (integrity rules, consistency rules) of its metamodel.

- Wellformedness is checked by **semantic analysis (context analysis)**:
  - **Name analysis (Namensanalyse)** finds the meaning of a name
    - **Type analysis (Typananalyse) f**inds the meaning of a type
  - **Type checking** checks the use of types with their definition
  - **Invariant checks**
    - **Range checks** (Bereichsprüfungen) test the validity of variables in ranges
    - **Structuring** of data structures: Acyclicity, layering, connected components, reducibility
  - **Forbidden combinations**
  - **Replicated definitions**

# Well-formedness of Metamodels and Data Dictionaries



A **model** is **well-formed (consistent)**, if it fulfils the context-sensitive constraints (integrity rules, consistency rules) of its metamodel.

A **metamodel** is **wellformed**, if it fulfils the context-sensitive constraints of its metametamodel.

A **data dictionary** is **wellformed**, if all contained models fulfil its context-sensitive constraints.

A **metamodel repository** is **wellformed**, if it fulfils all its context-sensitive constraints.

A **megamodel** is **wellformed**, if it fulfils all its context-sensitive constraints. Then it is called a **macromodel.**

# Reuse Languages and Contract Languages

A **reuse language** is a (sub-)language) controlling the reuse of program or model elements.

Examples:

- **Contract languages** check whether components, modules, classes, procedures and methods are applied correctly
- **Component model definition languages** define reuse languages and contract languages [Johannes-PhD]

# 14.5 Data Transformation Languages (DTL)

Text, XML, Term, and Graph Rewriting

see separate Chapter

# DTL and DML

- Mit DML (Datenmanipulationssprachen) formt man Daten um.
- **Declarative DTL (Datentransformationssprachen, DTL)** consist of declarative rule systems transforming a repository
  - Term rewriting for trees, terms, link trees, and XML trees
  - Graph rewriting for graphs
- **Imperative DML (general DML)** know states and side effects.

# Restructuring Languages (DRL)

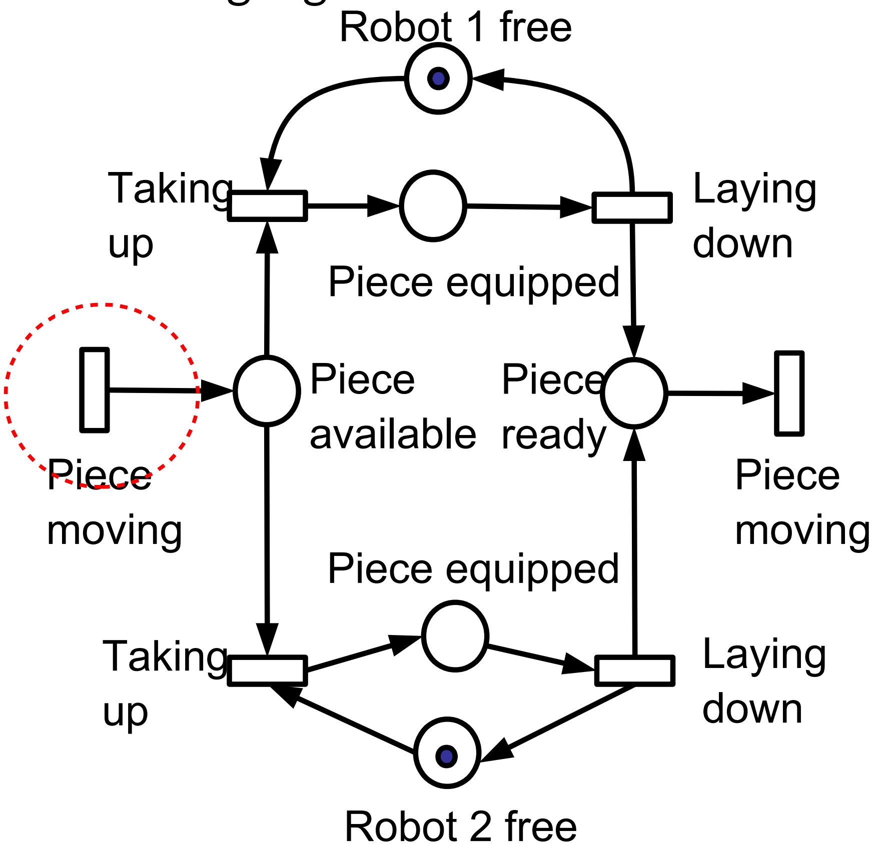
- **Restructuring** means to transform while to retain invariants.
- A **restructuring language** is a DTL giving guarantees about the transformed materials.
- A **refactoring language** restructures code and retains some of its invariants
- Languages for **Refinement**:
  - Refinement means that a transformed program *implies the semantics* of the original
  - A **wide spectrum language** transforms programs by refinement, generating more and more versions *implying* the requirements specification (the original)

# 14.6. Behavior Specification Languages (BSL)

All automata (workflow engines) in a TS execute workflows written in a BSL.

# Automaten, Petri-Nets, DFD and Workflow Languages

- **State-oriented Behavior specification languages** enable the specification of **interpreters (operational dynamic semantics)** and **simulators** (interpreters with a specific platform)
  - Automata, Transducers, Statecharts → course Softwaretechnologie-I
  - DFD, Petri-Nets and Workflow languages → course Softwaretechnologie-II
  - Appendix: DFD



[Balzert]

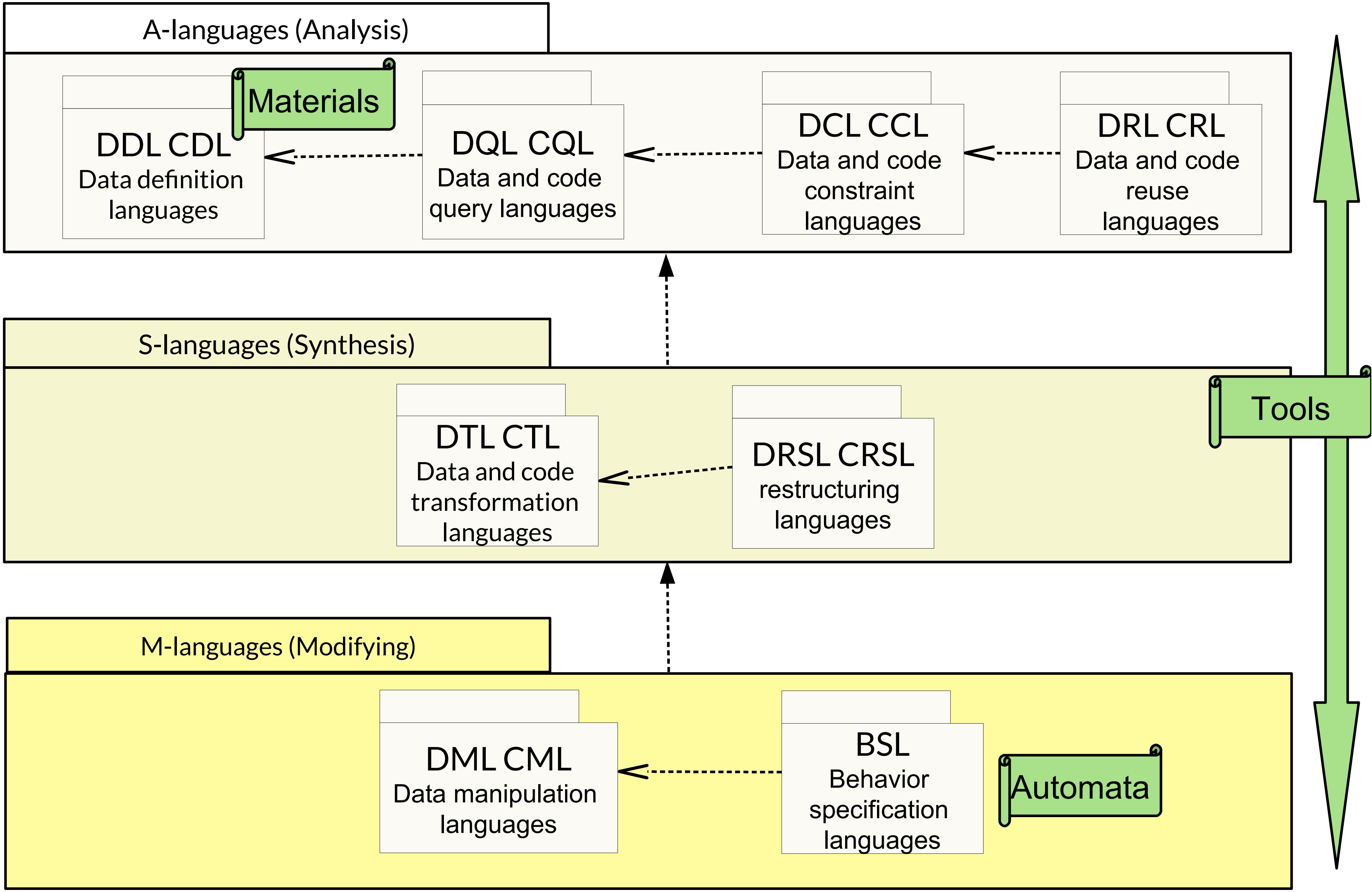
# 14.7 Language Families on the M2 Layers

Every technical space has a language hierarchy on M2 with a similar, layered structure.

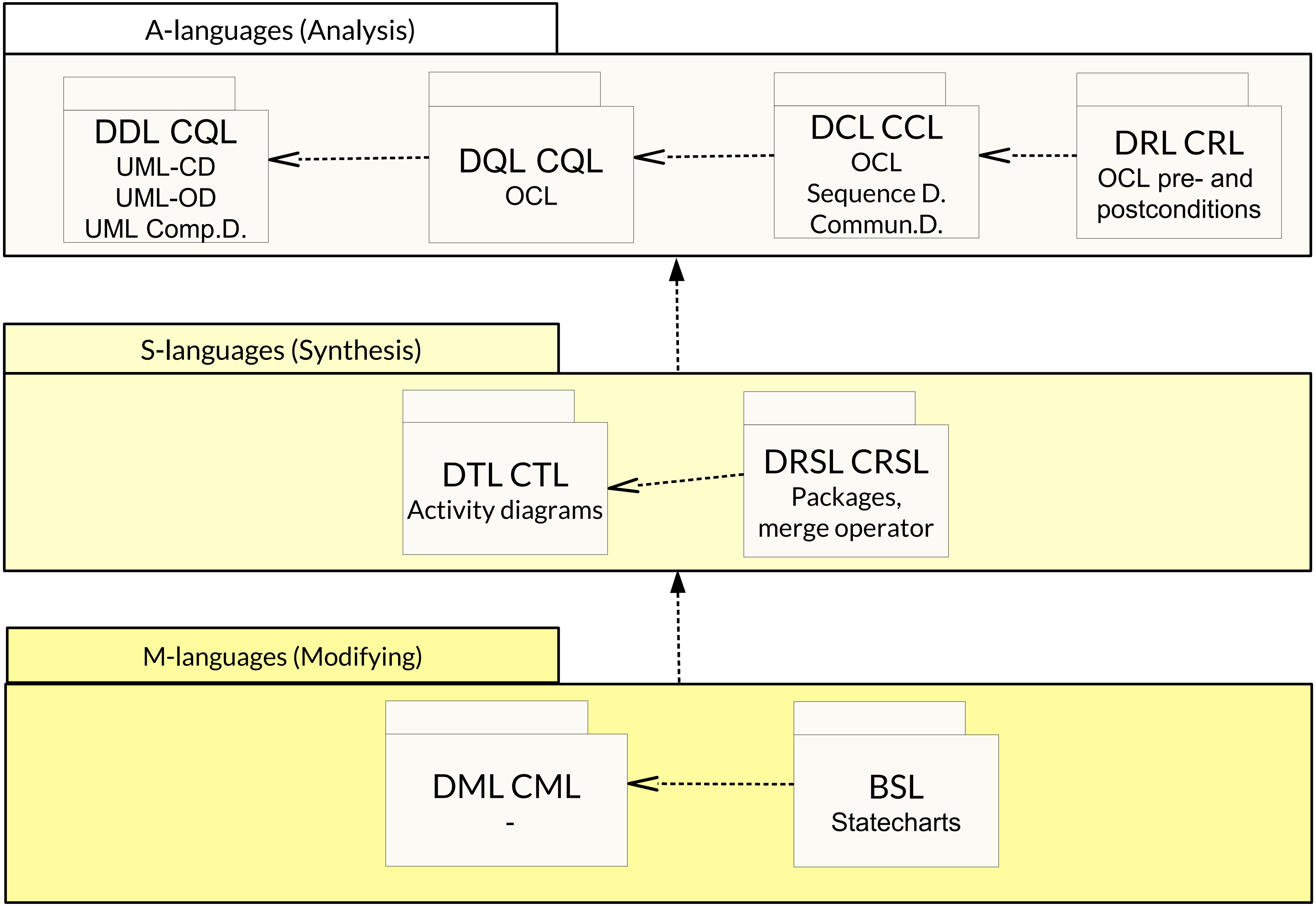
All tools have an underlying language family.

Every IDE has an underlying language family.

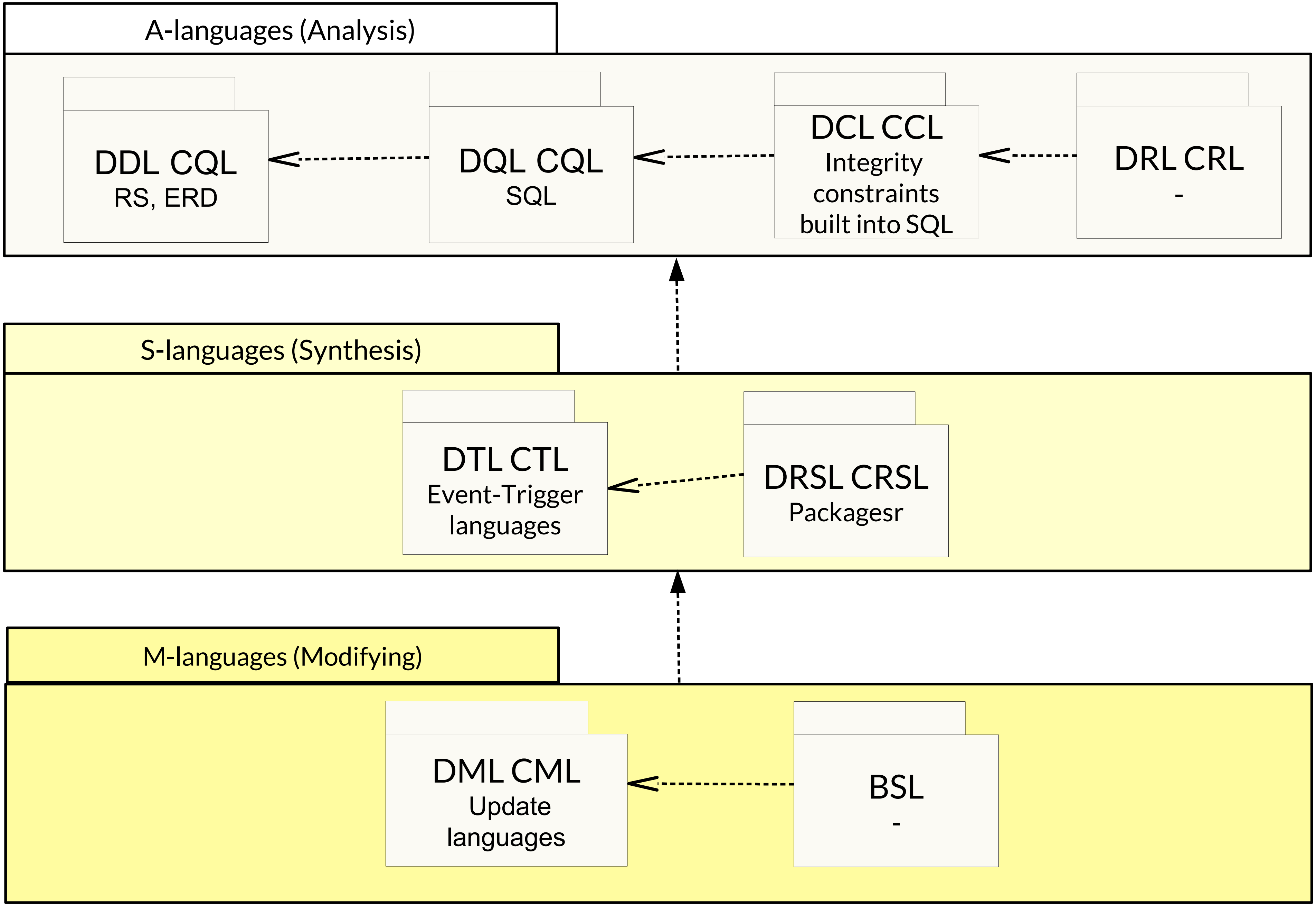
# Basic Language Families (Layer Structure of M2)


A-languages (Analysis)
Materials
DDL CDL
Data definition languages
DQL CQL
Data and code query languages
DCL CCL
Data and code constraint languages
DRL CRL
Data and code reuse languages
S-languages (Synthesis)
DTL CTL
Data and code transformation languages
DRSL CRSL
restructuring languages
Tools
M-languages (Modifying)
DML CML
Data manipulation languages
BSL
Behavior specification languages
Automata

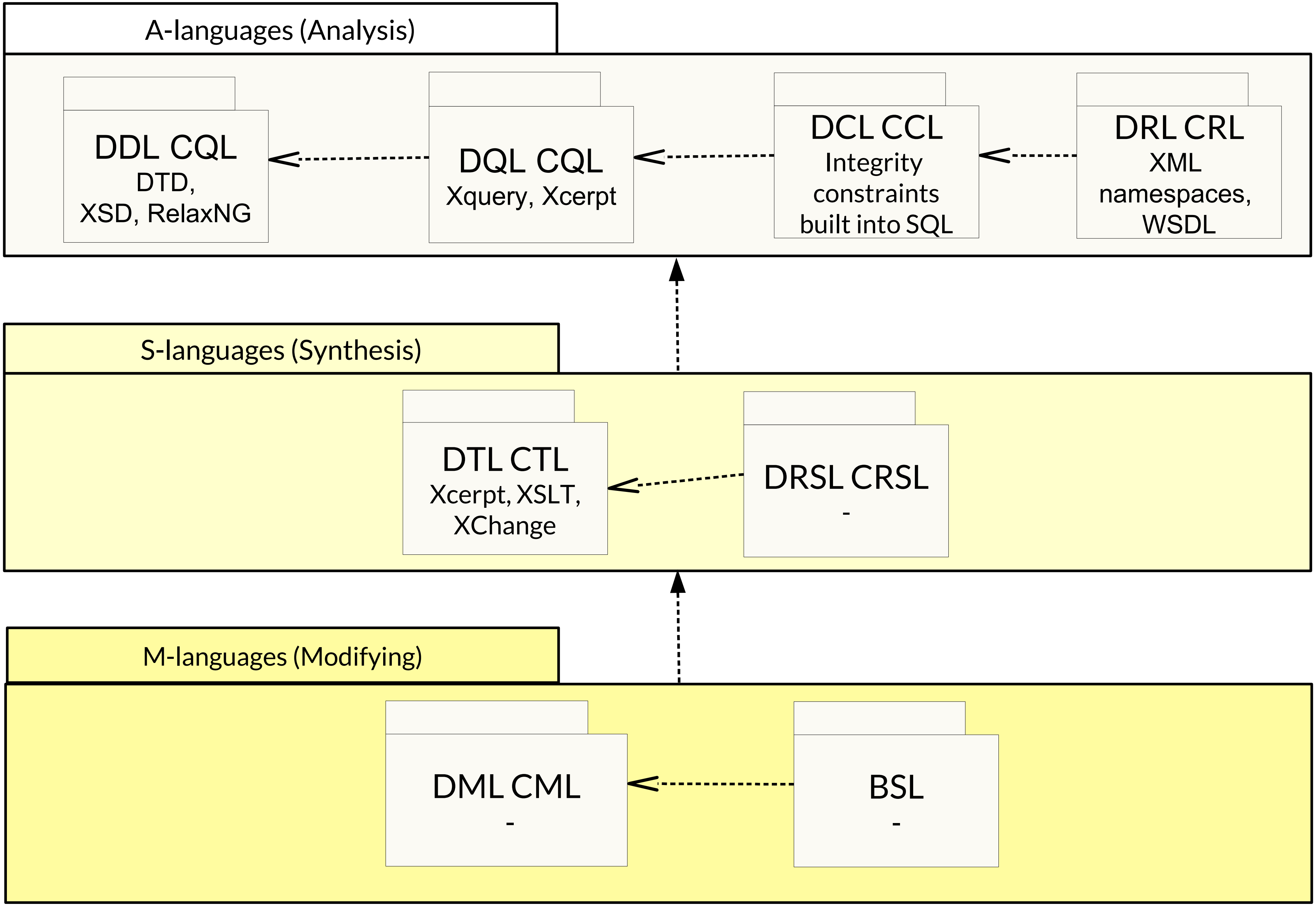
# UML Language Family in the ModelWare TS



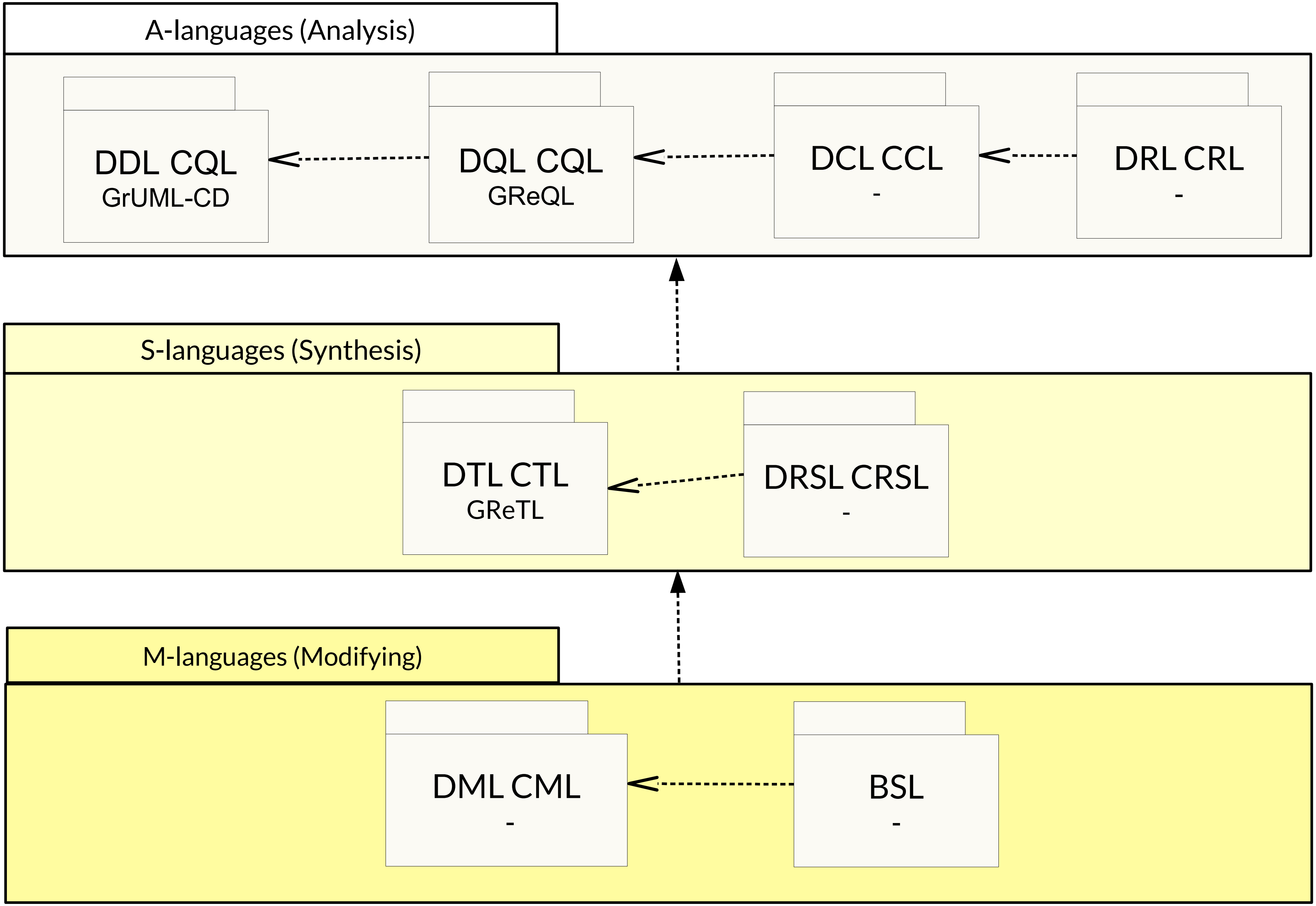
# ERD/RS Language Family in the Relational TS



# XML Language Family in the Link Tree TS



# GrUML Language Family [Ebert]



# The Generic Tools of a Technical Space (TS)

M3 metametamodel level

Metalanguage

Modelling concepts

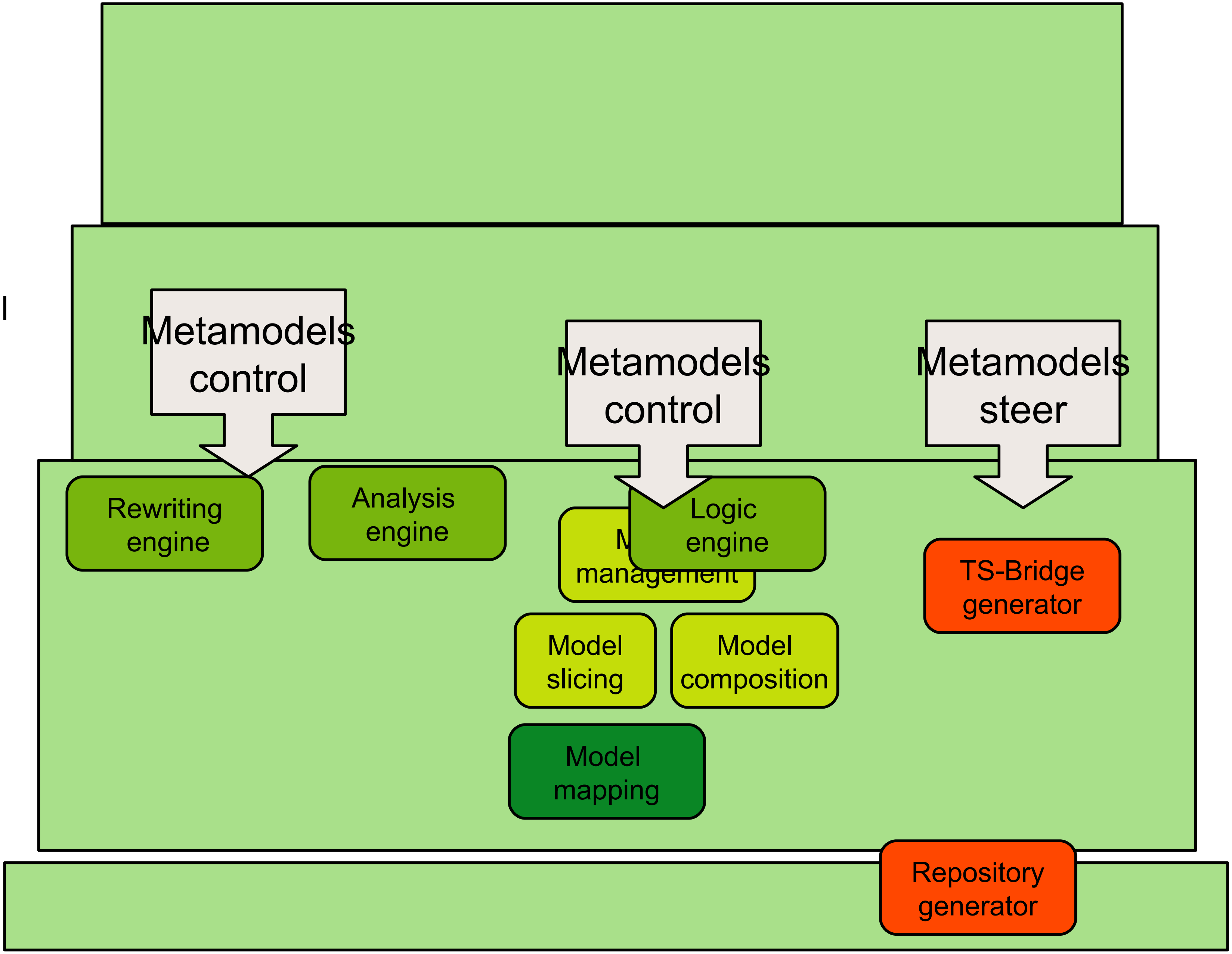
M2 metamodel level

Metamodels (languages)

M1 model level

Models, Programs

M0 object level



# The Generic Tools of a Technical Space (2)

M3 metametamodel level

Metalanguage

Modelling concepts

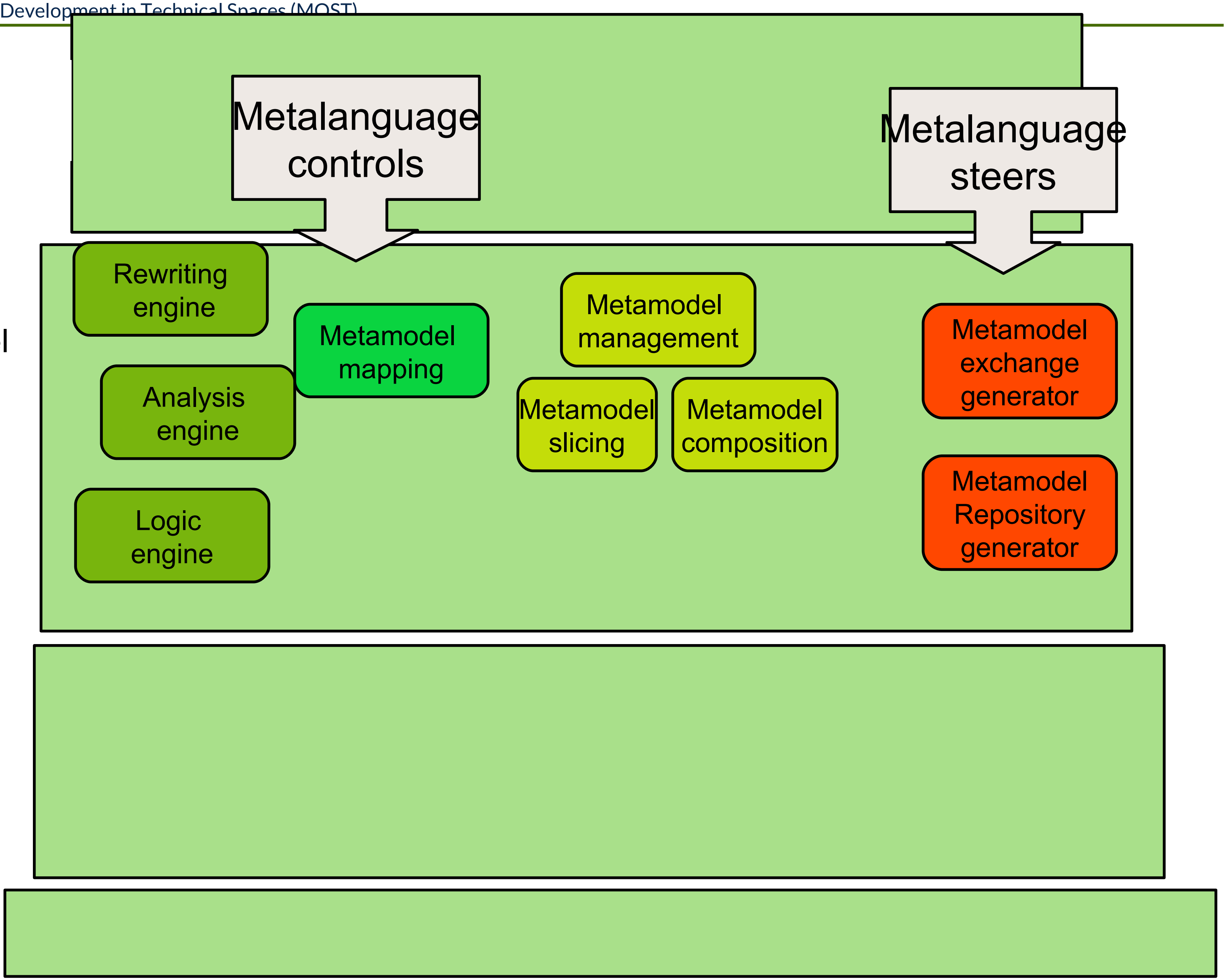
M2 metamodel level

Metamodels (languages)

M1 model level

Models, Programs

M0 object level

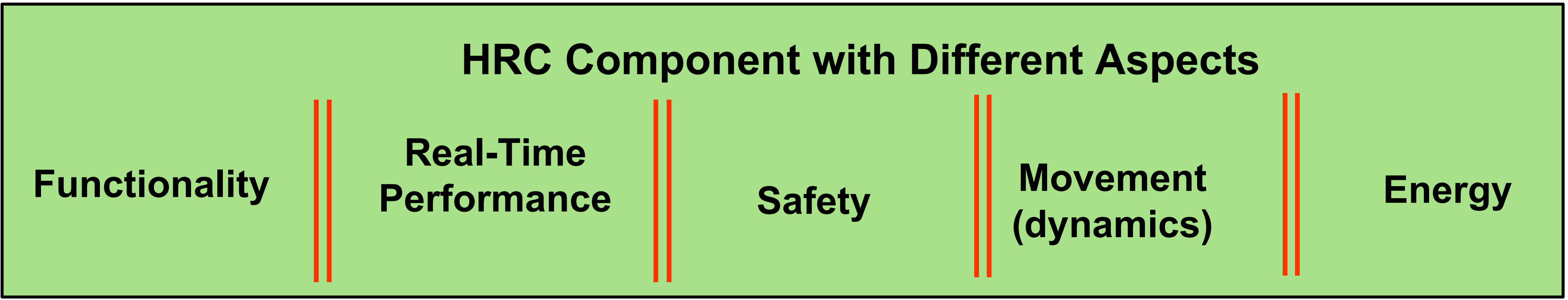


# 14.8. … and all together now...

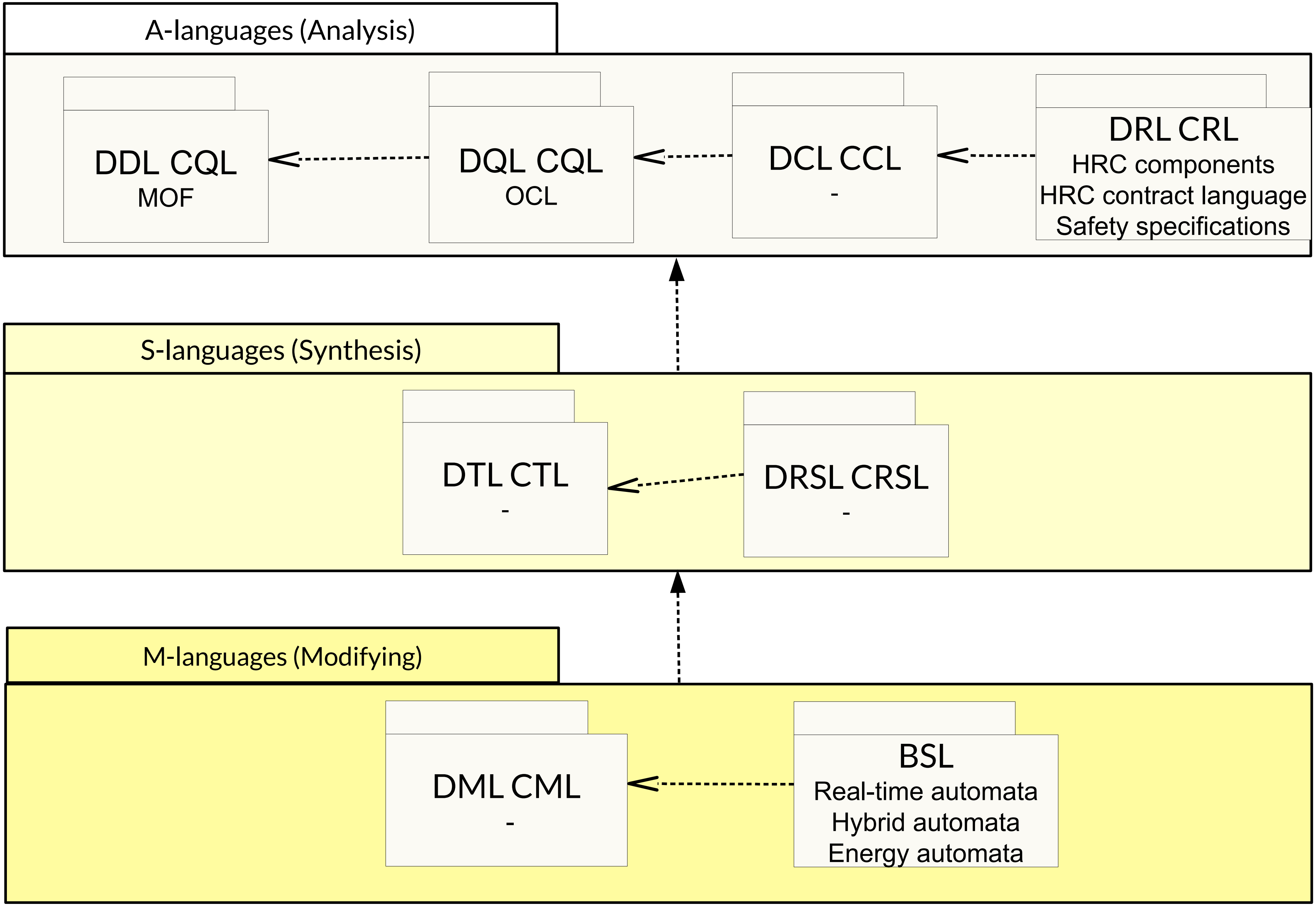
# Example Megamodel: Composition of Contracts in the HRC (Heterogeneous Rich Components) MDSD Tool Chain for Complex Embedded Systems

[Vered Gafni]

- Within a HRC component, contracts *in different views* can be synchronized (synchronized token-based modeling)
  - The real-time assertions can be coupled with functional, real-time, safety, physical movement (dynamics), and energy view
  - Every contract has a different contract language
- Between different components, the contracts of a certain viewpoint can be composed and checked (viewpoint-specific modeling)



# Example Megamodel: HRC Language Family for Safety-Critical Embedded Software



# Why is it Important to Know about the M2-Layers?

The megamodels managed by MDSD Tool Chains and Software IDE very often combine different languages from several layers of M2 (**M2-Mix**)

- ERD – MOF – XSD – UML-CD
- Xquery – XSLT – SQL – SPARQL
- OCL – SpiderDiagrams – OntologyLanguages
- Java – C++ – C#
- Petrinets – DFD – WorkflowNets – BPMN

Domain-specific languages always consist of an M2-Mix

Basic techniques (Basistechniken) also

Methods also

# Why is it Important to Know about the M2-Structure?

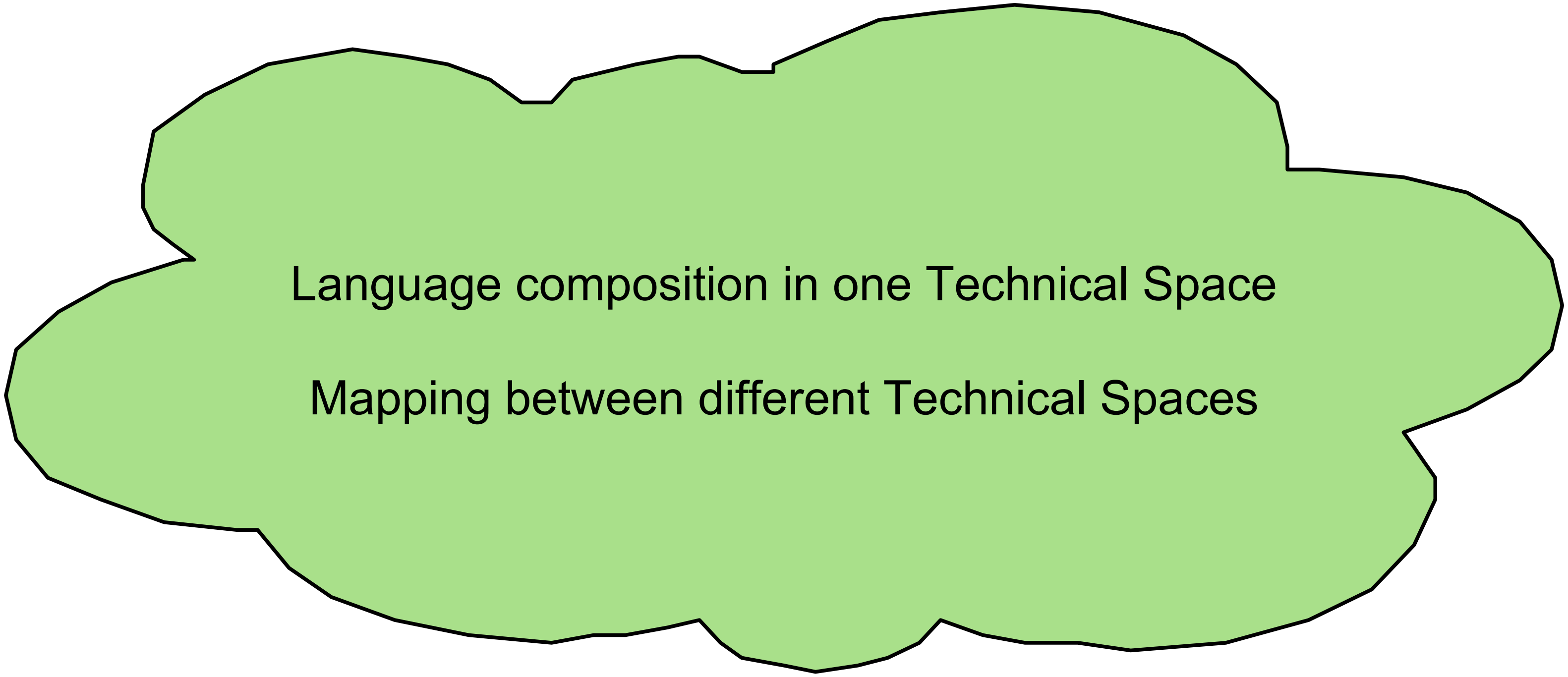
How can we compose metamodels for tool composition?

- Language families can be arranged in M2 layers
  - Many languages on upper layers can be composed with languages on lower layers
- If everything is in one Technical Space, composition of tools relies on the composition of languages
  - For that we need Model Composition Systems (forthcoming, ⍰ course CBSE)
    - Example: UML-Package Merge-Operator

**Language composition**: Compose new language constructs from layers further down

# How Can We Compose Tools for Base Techniques for MDSD Tool Chains and Software IDE?

- If we have to treat several Technical Spaces, Bridges between TS have to be built



Language composition in one Technical Space

Mapping between different Technical Spaces

# The End – Exam Questions

- Where would you position a query tool on M1 – in the tools layer or in the materials layer? Does the tool's metamodel in its query language belong then to the tools metamodel layer of M2 or to the materials metamodel layer?
- Why can we compose different DQL with a given DDL?
- How is it possible to apply a graph query language on XML trees?
- Why is UML such a complex language?
- A MDSD tool chain such as the HRC IDE for embedded systems works with many languages in different technical spaces. Explain some ingredients of such a complex IDE.

# A.1 Data Flow Diagrams (Datenflussdiagramme, DFD)

Repetition from course ST-II

DFD are special Petri nets resp. Workflow languages without global state

# Datenflußmodellierung

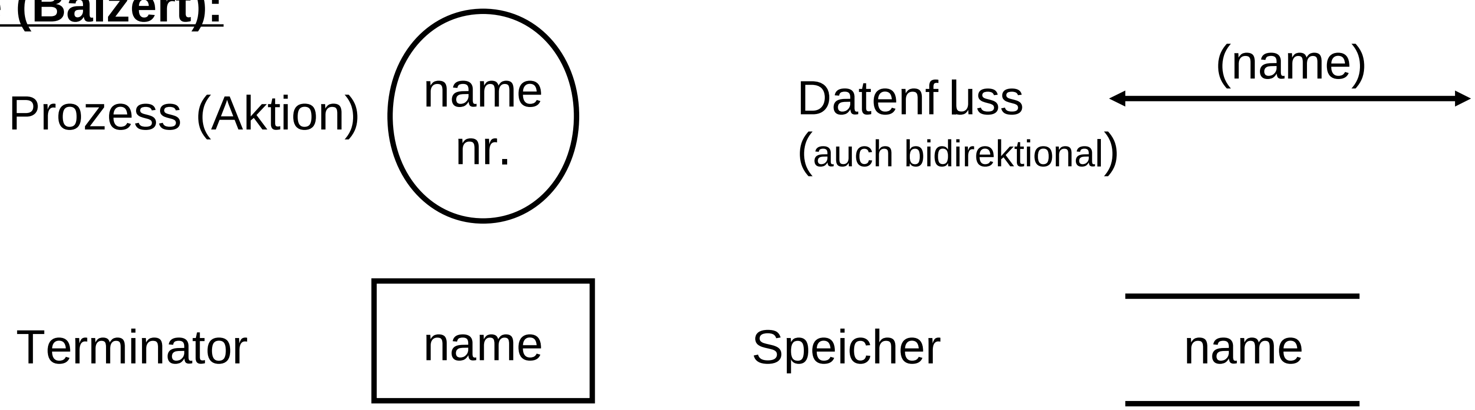
- **Datenfluss-Modellierung:** Prozesse (Iterierte Aktionen) auf Datenflüssen, ohne gemeinsames Repository
  - Datenfluss (Datenströme, streams, channels, pipes) zwischen Prozessen (immerwährenden Aktivitäten auf einem Zustand)
  - Datenflussdiagramme werden für strukturierte Prozesse (Geschäftsprozesse, technische Prozesse, Abläufe in Werkzeugen) eingesetzt
- Datenfluss-Modellierung ist Hauptbestandteil der **Strukturierten Analyse (SA)**

# DFD-Modellierung

- Hierarchische (reduzible) Prozessspezifikationen:
  - Kontextdiagramm (oberstes Diagramm, mit Terminatoren)
  - Parent-Diagramme
  - Child-Diagramme (Verfeinerte Prozesse)
- Datenkatalog wird benutzt zur Typisierung (spezifiziert in einer DDL)
- Minispezifikationendienen der Beschreibung der in Elementarprozessen durchzuführenden Transformationen.
  - mit Pseudocode
  - mit einer Transformationssprache wie Xcerpt

–

**Symbole (Balzert):**



Prozess (Aktion)

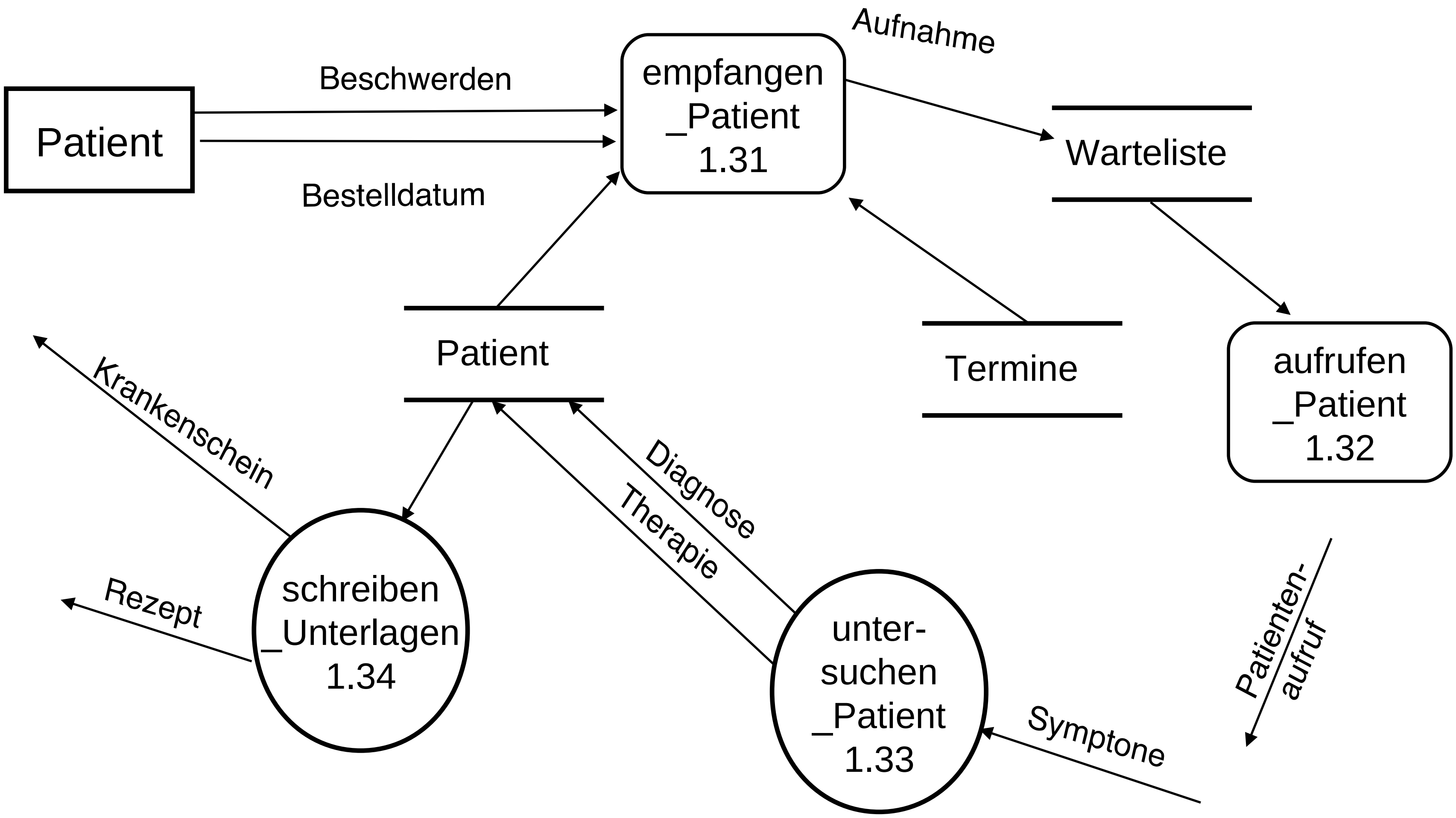
Datenfluss
(auch bidirektional)

Terminator

Speicher

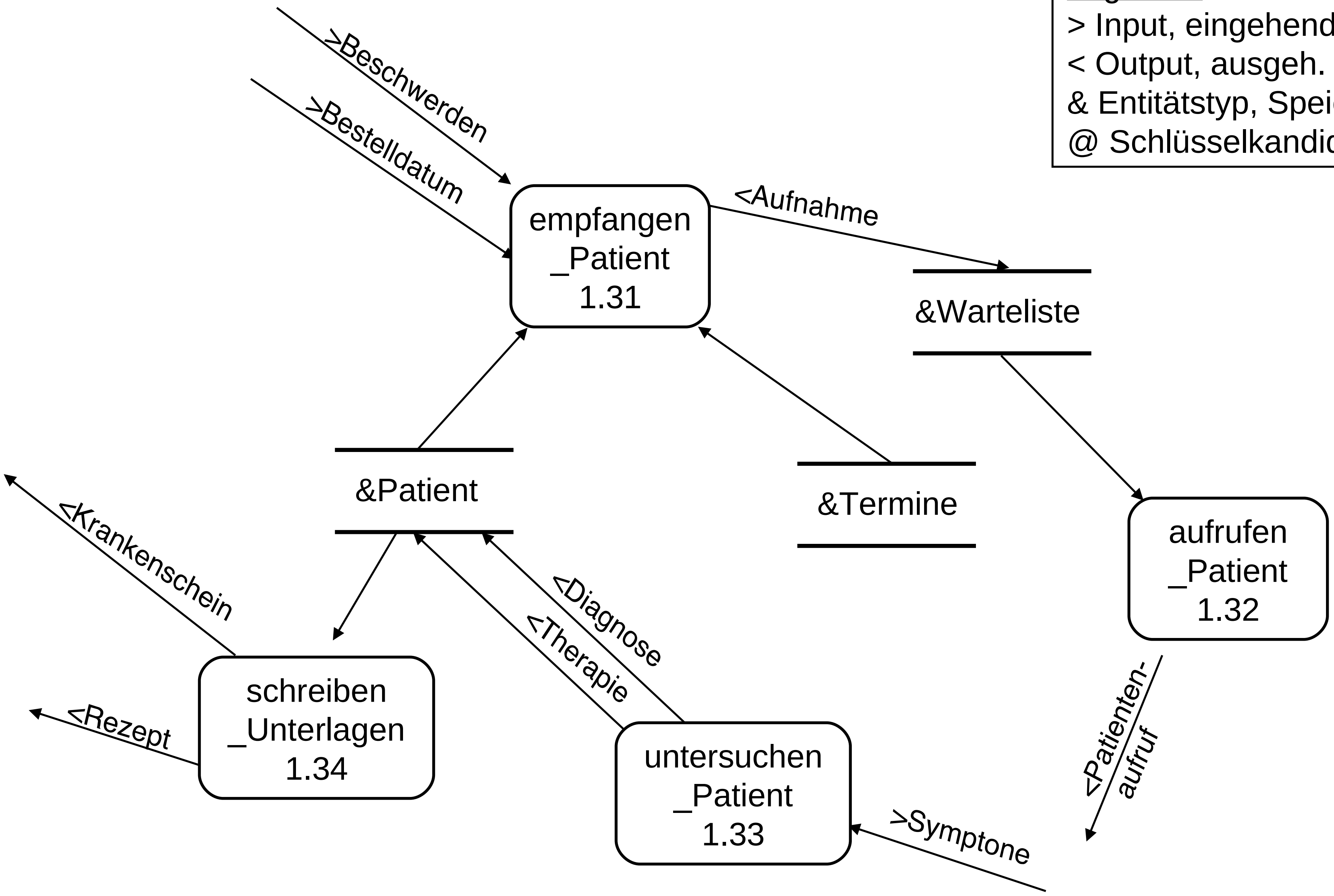
# DFD-Beispiel "behandeln_Patient"

- Prozesse auf Datenströmen, auch Geschäftsprozesse
- Kein zentrales Repositorium, lokale Daten, explizite Definition des Datenflusses
- UML notiert Aktivitäten und Prozesse mit Ovalen, SA/Balzert mit Kreisen



# Verfeinertes DFD-Beispiel "behandeln_Patient"

Legende:
> Input, eingehender Datenf l
< Output, ausgeh. Datenf luss
& Entitätstyp, Speicher
@ Schlüsselkandidat



# Wohlgeformtheitsregeln (Integritätsregeln) von DFD

- **Syntaktische Regeln** zur graphischen DFD-Darstellung:
  - Jeder Datenfluss muß mit mindestens einem Prozess verbunden sein.
  - Datenflüsse zwischen Terminatoren und zwischen Speichern sind nicht erlaubt.
  - Datenspeicher, die nur einseitig beschrieben (ohne zu lesen) und nur einseitig gelesen (ohne zu beschreiben) werden, sind nicht erlaubt.
  - Prozesse, die Daten ausgeben, ohne sie erhalten zu haben oder umge- kehrt, die Daten erhalten, ohne sie auszugeben oder zu verarbeiten, sind nicht erlaubt.
  - Im Kontextdiagramm darf es keine Speicher geben, in Verfeinerungen keine Terminatoren
  - Jeder Prozess, Speicher und Datenfluss muss einen Namen haben. Nur in dem Fall, wo der Datenfluss alle Attribute des Speichers beinhaltet, kann der Daten- flussname entfallen.
- **Semantische Konsistenzregeln** zur Wohlgeformtheit der Namensgebung:
  - Prozessnamen: Verb_Substantiv zur aussadgekräftigen Beschreibung (z.B. berechne_Schnittpunkt)
  - Datenflussnamen: [<Modifier>]Substantiv beschreibt momentanen Zustand des Datenflusses (z.B. <neue>Anschrift )
  - Speichernamen: Substantiv, das den Inhalt des Speichers (identisch Entity im DD) beschreibt (z.B. Adressen)

**Weiterführende Literatur:** [BAL]

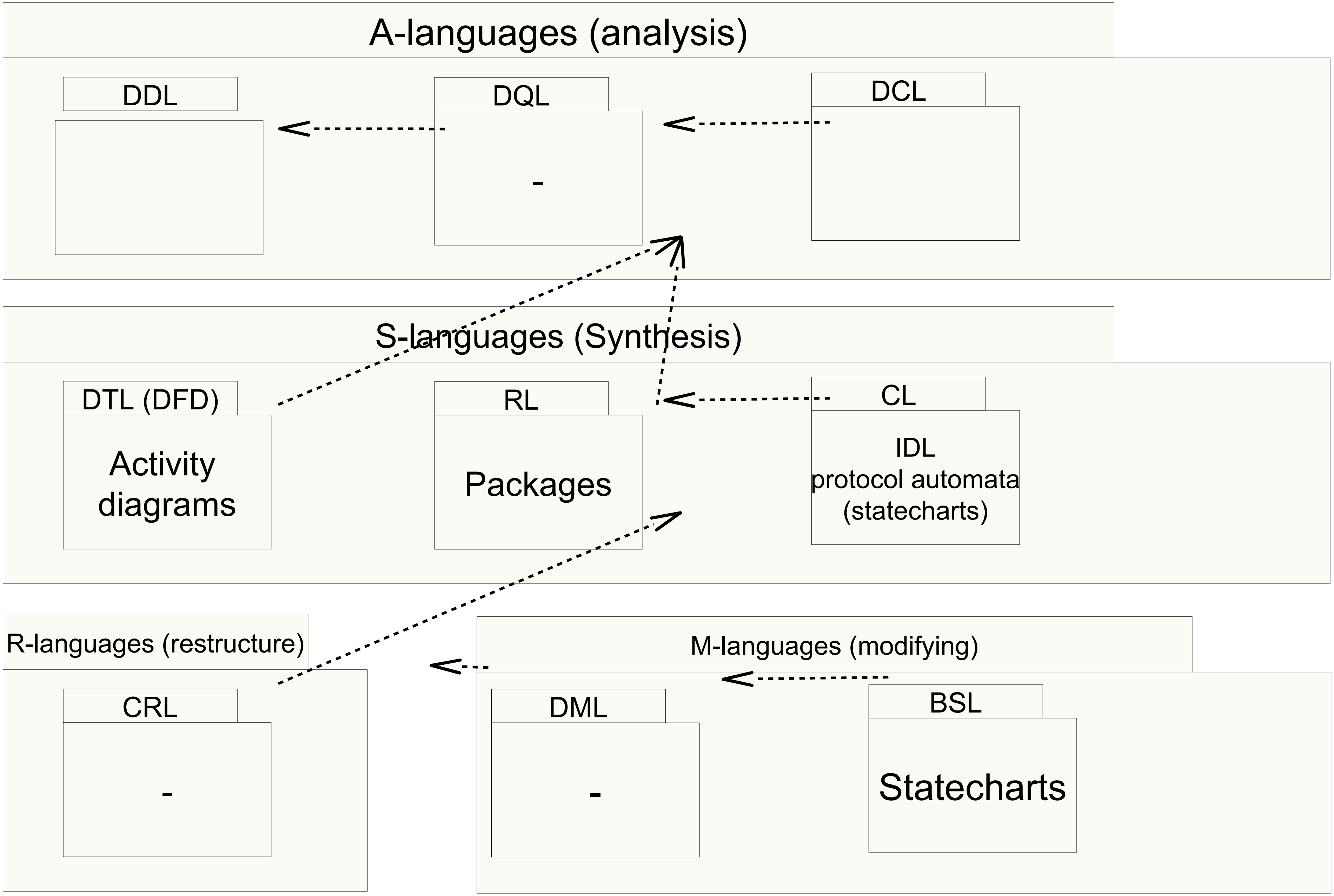
# Integritätsregeln der DFD-Erstellung: Balancieren zwischen DFD und anderen Sprachen

- **Vertikales Balancing** zwischen Knoten und Verfeinerungen
  - Alle Komponenten der im Vater referenzierten Flüsse sind zu benutzen.
- **Horizontales Balancing** zwischen DFDs und Minispezifikationen:
  - Jede Minispezifikation muß genau einem (Primitiv-)Knoten zuordenbar sein und umgekehrt
  - Alle Schnittstellen zu Knoten müssen in der MSpec referenziert sein und umgekehrt.
  - Alle Ausgaben jedes Prozesses müssen aus seinen Eingaben erzeugbar sein (korrekte Nutzung von Speichern!).
- **Balance von DFDs zum Data Dictionary:**
  - Zusammensetzung jedes Datenflusses und Speichers vollständig im DD beschrieben
  - Jedes Datenelement im DD muß in anderem Datenelement oder DFD vorkommen (Vollständigkeit)
- **Balance von ERD zu DFDs und Minispezifikationen:**
  - Jeder Speicher und Typ eines Kanals in einem DFDs muß einem Entitytyp des ERD entsprechen.

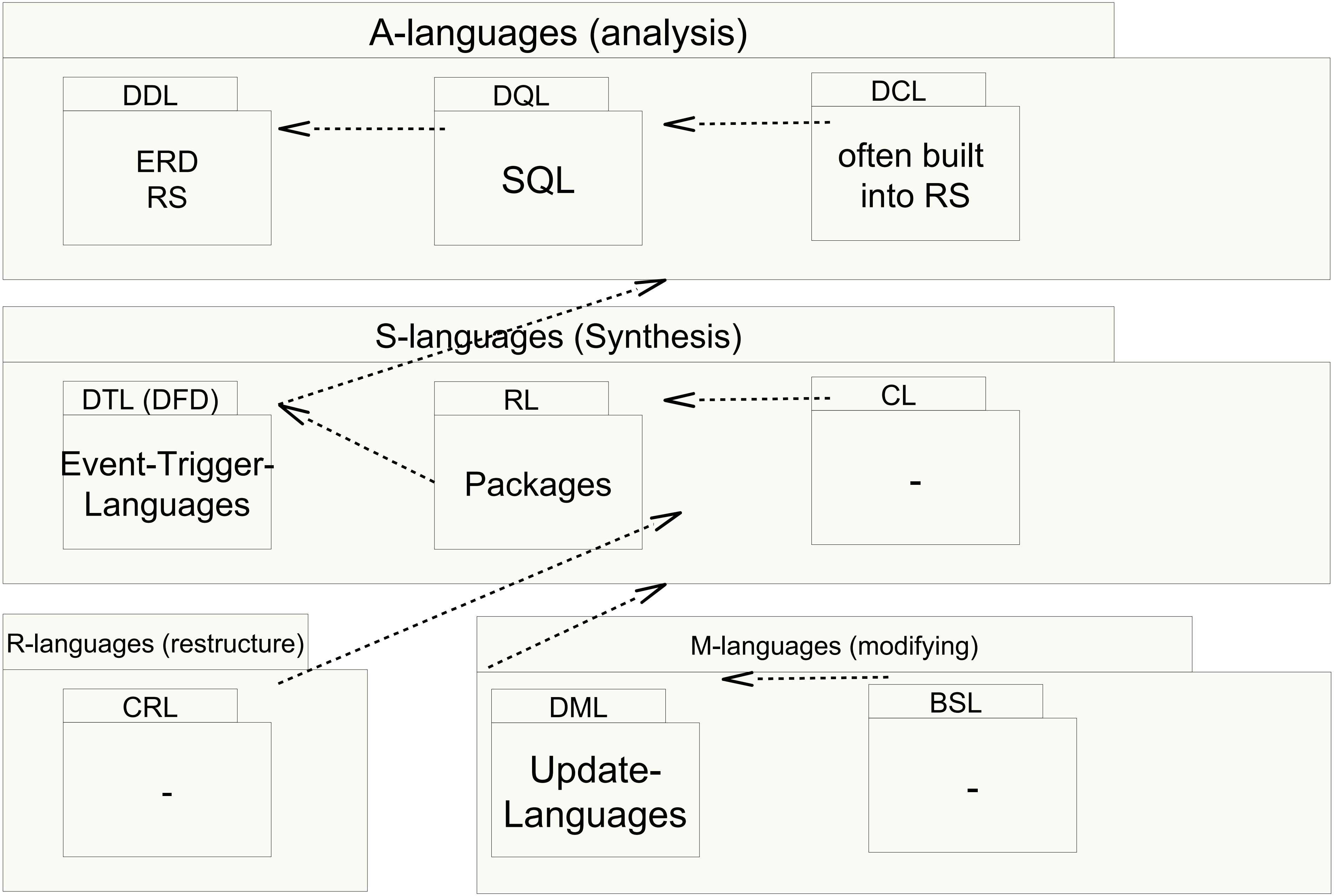
# DFD als BSL mit privaten Daten

- DFD verzichten auf ein globales Repositorium, sondern spalten die Daten in “private” Speicher auf,
  - für die explizit spezifiziert wird, wohin ihre Daten fließen
- DFD sind sehr gut geeignet für die Spezifikation von Werkzeugverhalten
  - Datenabhängigkeiten sind immer klar, da explizit spezifiziert
  - Natürliche Parallelität
  - Einfache Komposition durch Anfügen von weiteren Datenflüssen und Teilnetzen

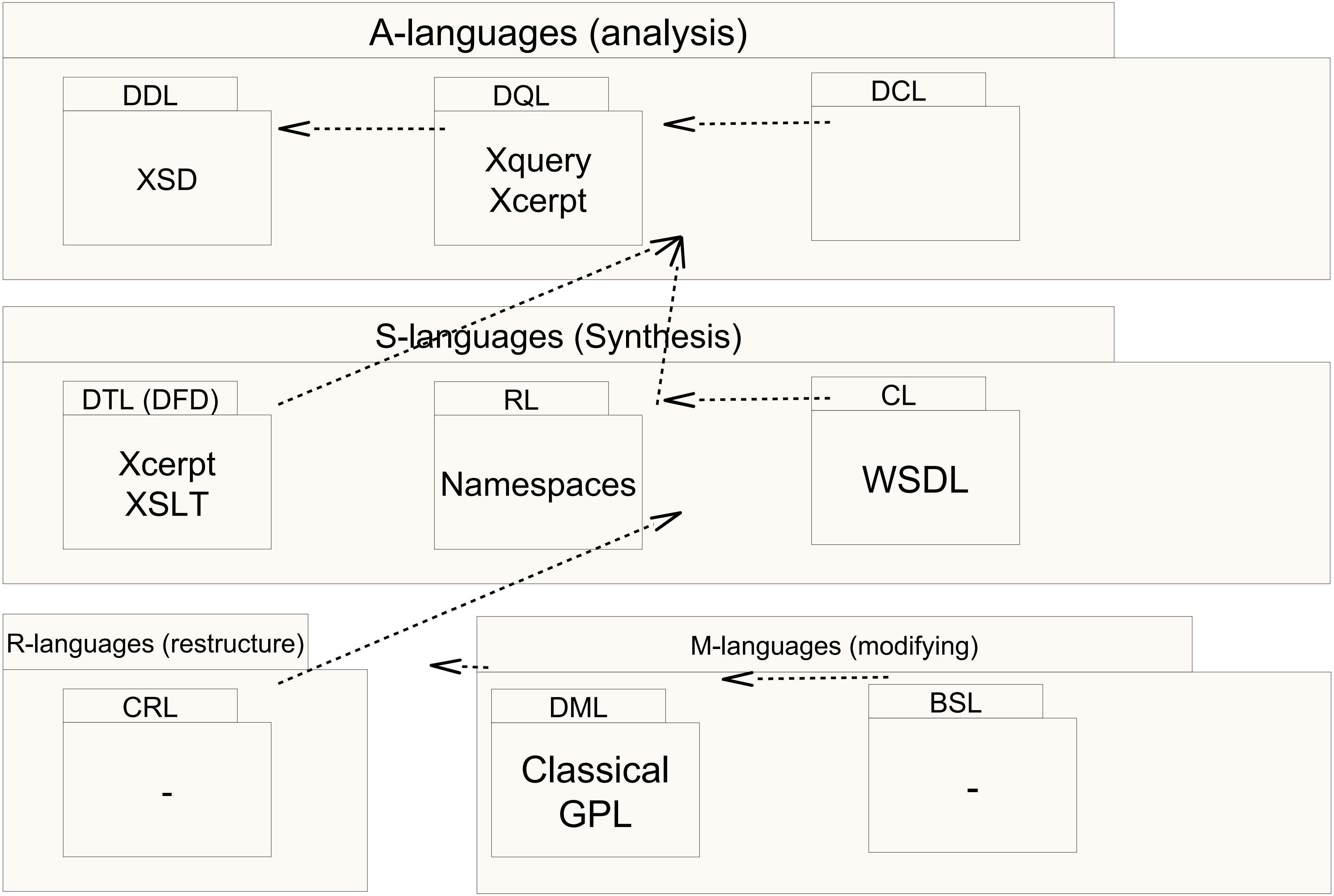
# UML Language Family



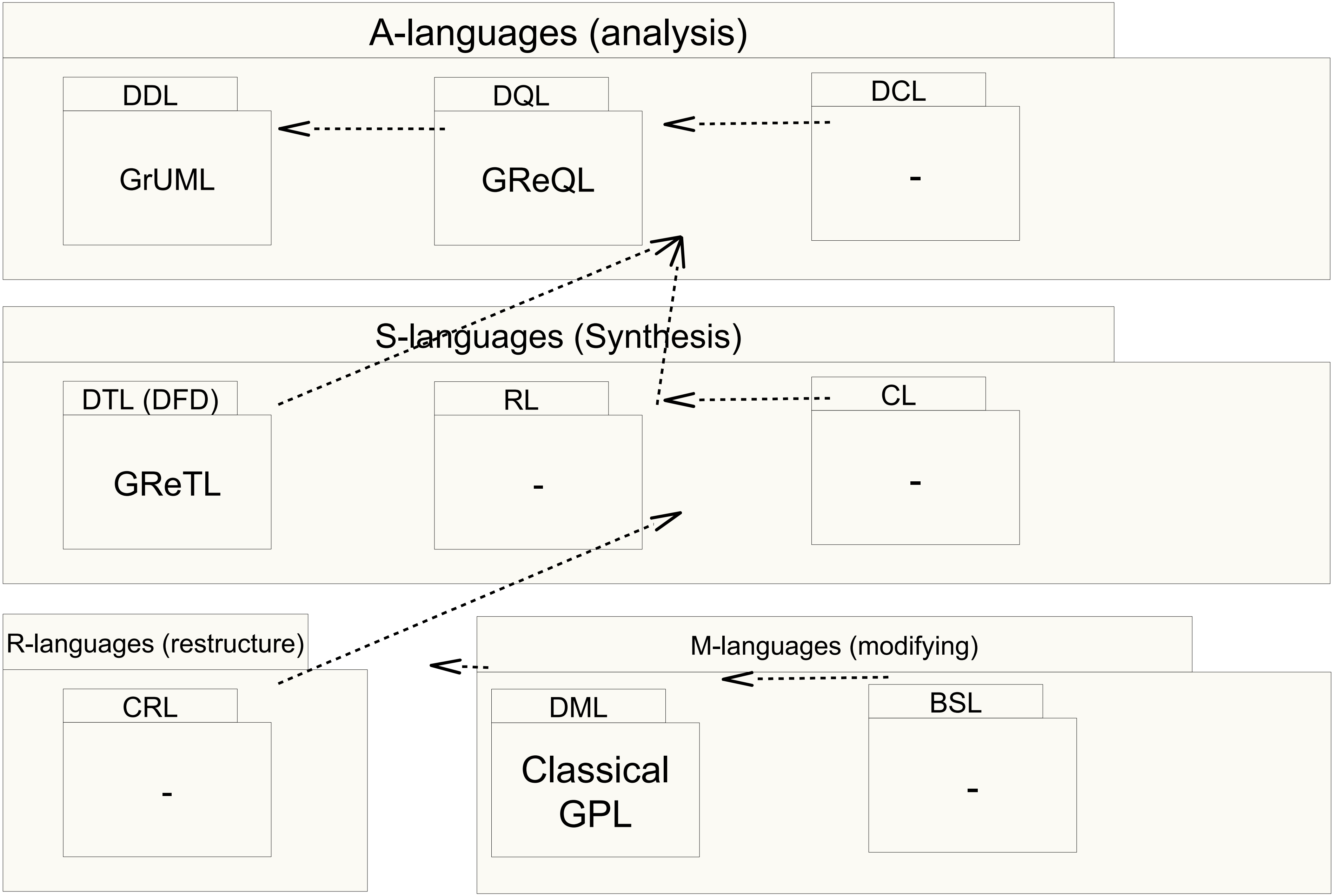
# ERD/RS Language Family



# XML Language Family



# GrUML Language Family


A-languages (analysis)
DDL
GrUML
DQL
GReQL
DCL
-
S-languages (Synthesis)
DTL (DFD)
GReTL
RL
-
CL
-
R-languages (restructure)
CRL
-
M-languages (modifying)
DML
Classical GPL
BSL
-