Model driven Engineering & Model driven Architecture

Prof. Dr. Mark van den Brand

Software Engineering and Technology
Faculteit Wiskunde en Informatica
Technische Universiteit Eindhoven

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Model driven software engineering

- **Models**
  - **often used for both hardware and software design**
  - **probably manually translated into design documents and code**
  - **no guarantee of consistency between model, design and resulting code**

Model driven software engineering

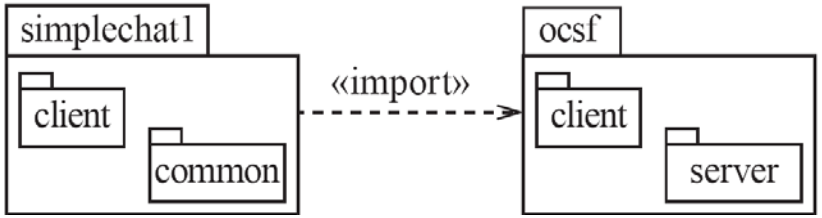
- **Modeling languages**
  - **whole range over the years:**
    - **data oriented, e.g., E/R models, class diagrams**
    - **behaviour oriented, e.g., use cases, state machines, sequence diagrams, activity diagrams**
    - **architecture oriented, e.g., package diagrams, component diagrams**
  - **standardization initiative of OMG:**
    - **Unified Modeling Language**

Model driven software engineering

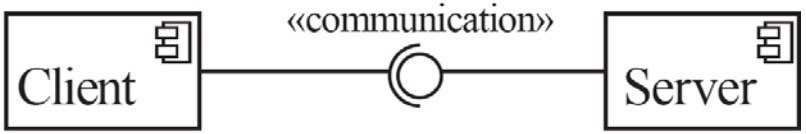
- **UML diagrams for architectural models:**
  - **All UML diagrams can be useful to describe aspects of the architectural model**
  - **Four UML diagrams are particularly suitable for architecture modelling:**
    - **Package diagrams**
    - **Subsystem diagrams**
    - **Component diagrams**
    - **Deployment diagrams**

Model driven software engineering

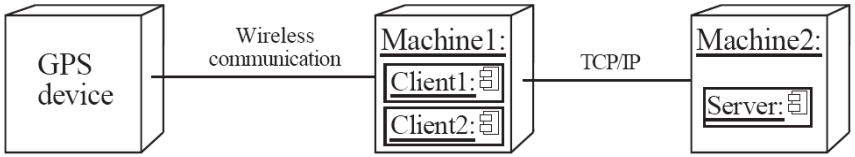
- **Package diagram:**



- **Component diagram:**



- **Deployment diagram:**



Model driven software engineering

- **The following list is a minimal set of requirements for a language to be an ADL. The languages must :**
  - **Be suitable for communicating an architecture to all interested parties**
  - **Support the tasks of architecture creation, refinement and validation**
  - **Provide a basis for further implementation, so it must be able to add information to the ADL specification to enable the final system specification to be derived from the ADL**
  - **Provide the ability to represent most of the common architectural styles**
  - **Support analytical capabilities or provide quick generating prototype implementations**

Model driven software engineering

- **ADLs have in common:**
  - **Graphical syntax with often a textual form and a formally defined syntax and semantics**
  - **Features for modeling distributed systems**
  - **Little support for capturing design information, except through general purpose annotation mechanisms**
  - **Ability to represent hierarchical levels of detail including the creation of substructures by instantiating templates**

Model driven software engineering

- **ADLs differ in their ability to:**
  - **Handle real-time constructs, such as deadlines and task priorities, at the architectural level**
  - **Support the specification of different architectural styles. Few handle object oriented class inheritance or dynamic architectures**
  - **Support analysis**
  - **Handle different instantiations of the same architecture, in relation to product line architectures**

Model driven software engineering

- **Positive elements of ADL**
  - **ADLs represent a formal way of representing architecture**
  - **ADLs are intended to be both human and machine readable**
  - **ADLs support describing a system at a higher level than previously possible**
  - **ADLs permit analysis of architectures – completeness, consistency, ambiguity, and performance**
  - **ADLs can support automatic generation of software systems**

Model driven software engineering

- **Negative elements of ADL**
  - **There is not universal agreement on what ADLs should represent, particularly as regards the behavior of the architecture**
  - **Representations currently in use are relatively difficult to parse and are not supported by commercial tools**
  - **Most ADLs tend to be very vertically optimized toward a particular kind of analysis**

Model driven software engineering

- **Automatic transformation of domain specific models into software models**
- **Automatic translation from software models into executable code**
- **Ingredients:**
  - **syntax and semantics of modeling formalisms should be described**
  - **correctness preserving transformation steps should be defined**
  - **code generators should be developed**

Model driven software engineering

- **Software generation:**
  - **Increase in productivity**
  - **Increase in quality**
  - **Based on existing formalisms**
    - **UML**
    - **domain specific extensions**
  - **Prototyping of tooling**
    - **model transformations**
    - **code generation**

Model driven software engineering

- **Domain specific languages**
  - **Little language for specific application domains**
    - **Terminology of application domain**
      - **domain concepts**
    - **Restricted number of language constructs**
    - **Easy to learn for domain engineers**
    - **Examples:**
      - **SQL**
      - **YACC (compilers)**
      - **Risla (modeling of financial products)**
      - **WebDSL**

Model driven software engineering

- **“State-of-the-art” technology: component based software development**
  - **Code Generation**
  - **Aspect Oriented Programming**
  - **Coordination Architectures**
  - **Design Patterns**

Model driven software engineering

- **Program Generators**
- **Automatic production of programs by means of other programs**
  - **A program generator reads meta-data and produces well-formed source code**
    - **Grammars**
    - **Database model**
    - **UML diagrams**
  - **A program generators makes your project “agile”**

Model driven software engineering

- **Advantages**
  - ***Increase in productivity*:**
    - **generating tedious and boring parts of the code**
    - **code generators produce thousands of lines of code in seconds**
    - **changes are quickly propagated**
    - **agile development**
  - ***Increase of quality*:**
    - **bulky handwritten code tend to have inconsistent quality because increase of knowledge during development**
    - **bug fixes and code improvements can be consistently roled out using a generator**

Model driven software engineering

- **Advantages**
  - ***Increase of Consistency*:**
    - **in API design and naming convention**
    - **single point of definition**
    - **explicit documented design decisions**
  - ***Architectural consistency*:**
    - **Programmers work within the architecture**
    - **Well-documented and -maintained code generator provides a consistent structure and approach**
  - ***Abstraction: language-independent definition***
    - **Lifting problem description to a higher level**
    - **Easier porting to different languages and platforms**
    - **Design can be validate on an abstract level**

Model driven software engineering

Models of program generators:

- Code munging: given information in some input code, one or more output files are generated, e.g., scanner or parser
- Inline-code expander: take source code with special mark-up code as input and creates production code in separate output file, e.g., imbedded SQL is replaced by C
- Mixed-code generation: take source code with special mark-up code as input and replace this inline

Model driven software engineering

Examples of program generators:

- Programming environment generators
- API generators
- UML based generators
- Domain Specific Languages

Domain specific languages

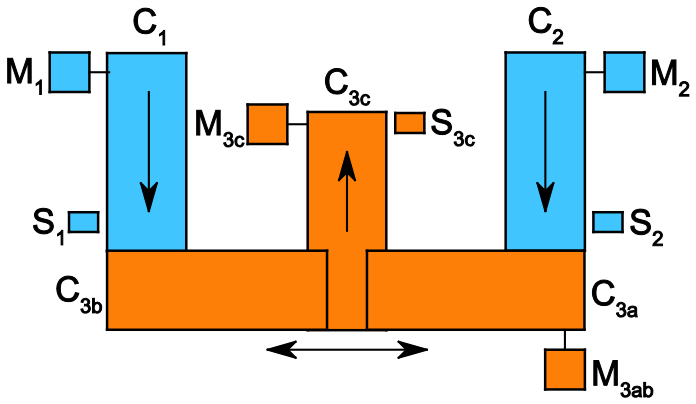
- **DSLs & Model driven software engineering:**
  - **Domain specific variants of UML**
    - **profiles: extending/adapting existing UML diagrams, e.g., SysML**
    - **meta modeling: entirely new diagrams**

Domain specific languages

- SysML offers improvements over UML, which tends to be software-centric:
  - SysML's semantics are more flexible and expressive
    - two new diagram types:
      - requirement and
      - parametric diagrams for performance analysis and quantitative analysis
  - SysML removes many of UML's software-centric constructs
  - SysML has allocation tables for
    - requirements allocation,
    - functional allocation, and
    - structural allocation
  - This capability facilitates automated verification and validation and gap analysis

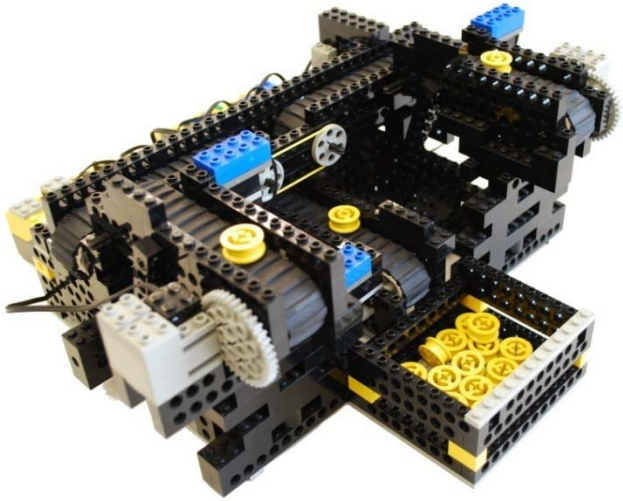
Domain specific languages

- **Example of a newly developed DSL**
  - **Specifying, Simulating, Verifying and Implementing the Controllers of a Conveyor Belt using model transformations**
    - **We defined a domain specific language (DSL) for modeling communicating systems**
    - **Simulation via transformation from our DSL to POOSL**
    - **Stepwise refinement to adapt the characteristics of the communication channels to the Lego Mindstorms platform**



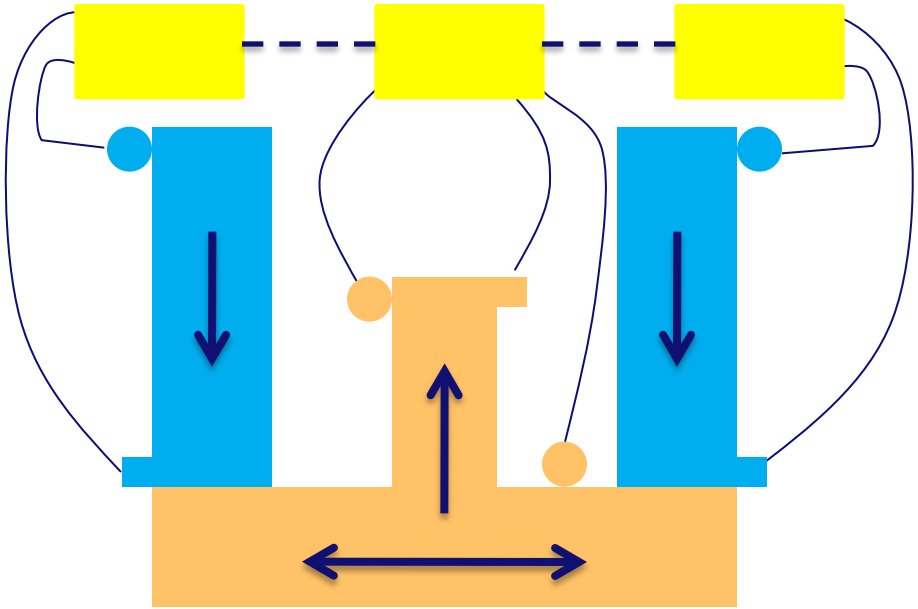
Domain specific languages

- **Specifying, Simulating, Verifying and Implementing the Controllers of a Conveyor Belt using model transformations**
  - **To verify the system, we implemented a transformation from our DSL to Promela, the language used by the model checker Spin**
  - **We implemented a transformation from our DSL into Not Quite C (NQC), a programming language for Lego Mindstorms' controllers**



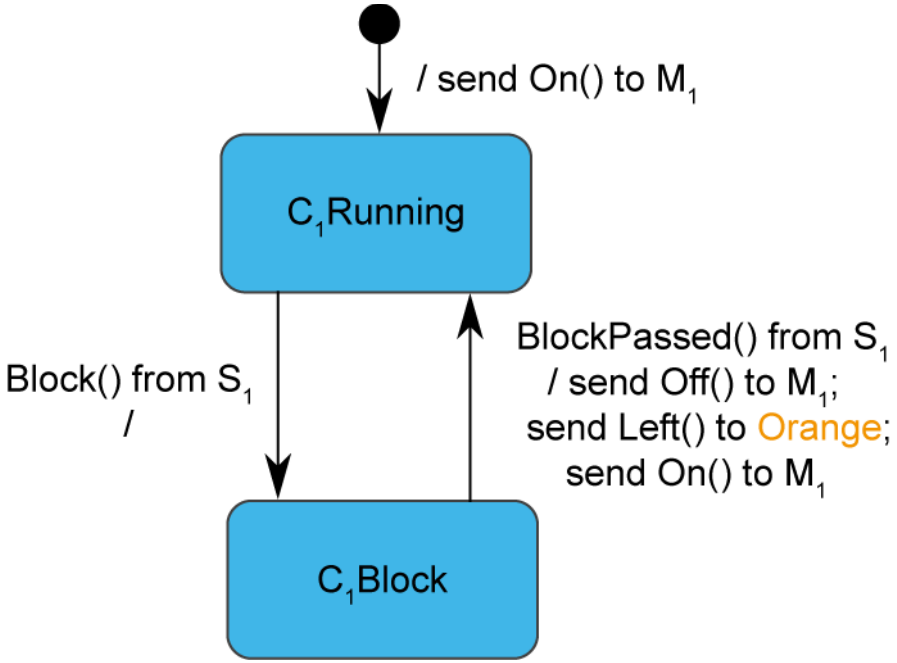
Domain specific languages

- **Concurrent objects**
  - **Controllers**
  - **Hardware**
    - **Conveyors**
    - **Motors**
    - **Sensors**
- **Communication**
  - **Wireless**
  - **Wired**

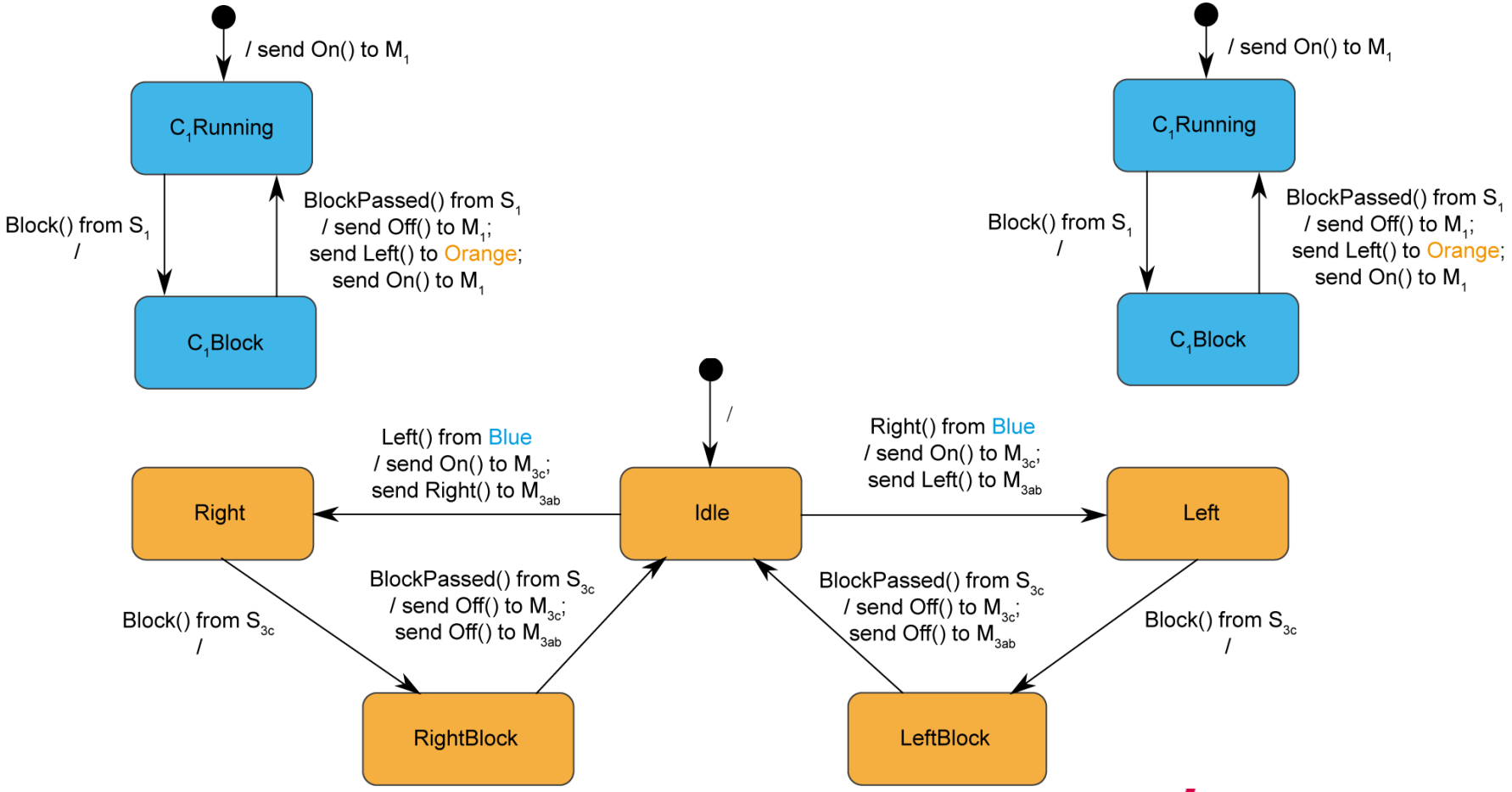


Domain specific languages

- **State machine based**
  - **Combination of:**
    - **graphical models and**
    - **textual models**
  - **Conditional message exchange**
    - **plus activities**
  - **No data yet**
  - **No timing yet**



Domain specific Languages



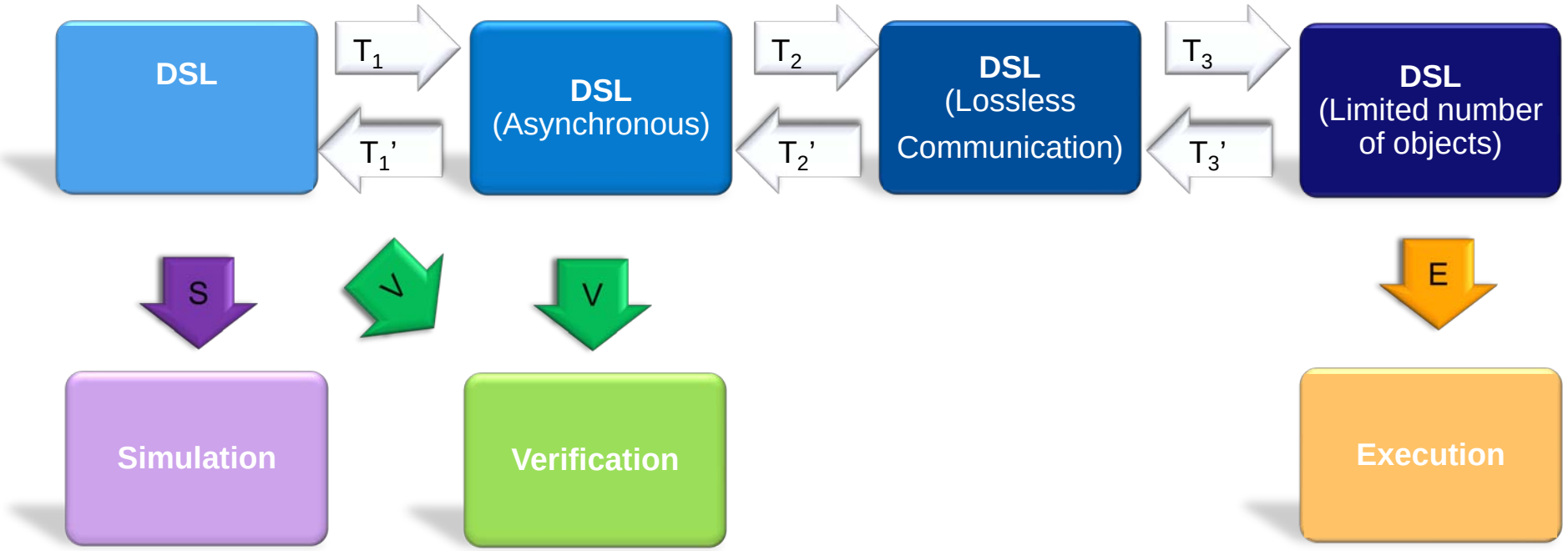
Domain specific languages

- **Platforms**
  - **Simulation**
    - **POOSL**
  - **Execution**
    - **NQC**
  - **Verification**
    - **PROMELA/SPIN**

	Synchronous/ Asynchronous	Lossless/ Lossy	#Objects
DSL	Both	Both	Unlimited
POOSL	Synchronous	Lossless	Unlimited
NQC	Asynchronous	Lossy	Limited
PROMELA	Both	Lossless	Unlimited

Domain specific languages

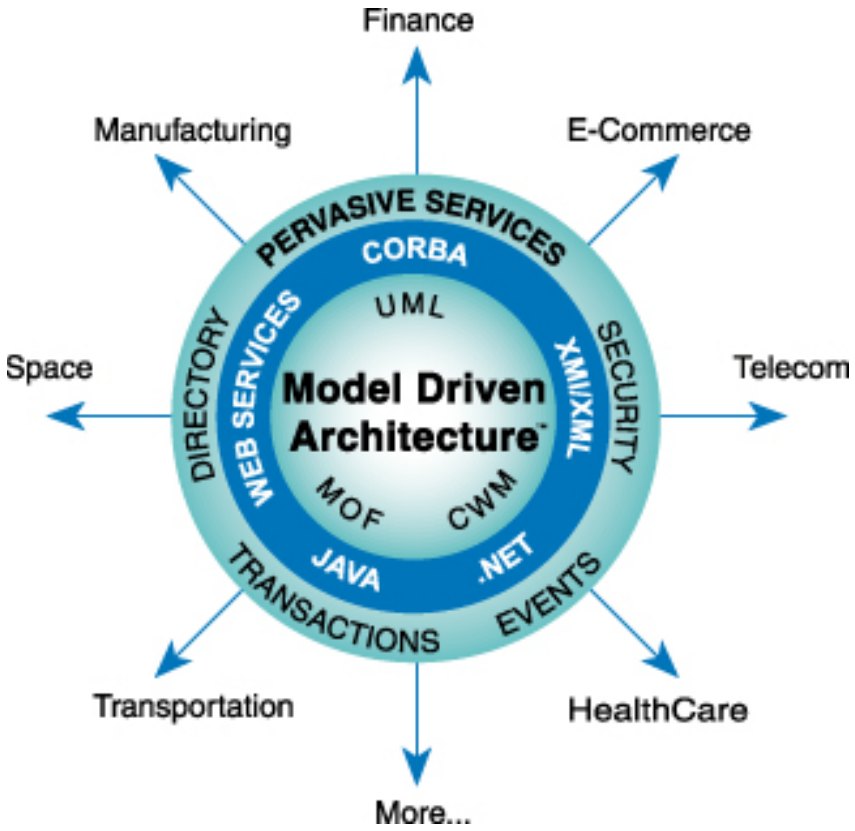
- **Model transformations**



Domain specific languages

- **Requirements for a DSL:**
  - **Unambiguous semantics**
  - **Formal composition rules**
  - **Unambiguous transformability**
  - **Strongly-typed and deterministic IO**
  - **Encapsulation**
  - **Expressiveness**
  - **Readability**

Model Driven Architecture



MDA

- **Today's Software Environment:**
  - **Worldwide distributed systems**
  - **Heterogeneous platforms, languages, and applications**
  - **Increasing interconnectivity within and between companies**
  - **New technologies: XML, .NET and web services**

MDA

- **Heterogeneous platforms and languages**
  - **Programming languages**
    - **~3 million COBOL programmers**
    - **~1.6 million VB programmers**
    - **~1.1 million C/C++ programmers**
  - **Operating systems**
    - **Unix, MVS, VMS, MacOS, Windows (about 10), PalmOS…**
    - **Windows 3.1: it's still out there!**
    - **Embedded devices (mobile, set-top, etc.)**

MDA

- Good News
  - Increased standarization
    - Internet protocols, SQL, UML
  - Increased openness
    - Linux, apache, etc.
  - Less custom specific development
    - Component reuse, ERP applications

MDA

- **Bad news**
  - **Legacy applications and databases**
  - **ERP applications that are difficult to adapt**
  - **Multiple, competing middleware**
  - **Develop software for the future: adaptable to future modifications**

MDA

- MDA is a more sophisticated way of using UML
- Raising level of abstraction:
  - General trend
  - Already well-established for front and back ends
    - WYSIWYG GUI modeling and data modeling
    - Hand coding no longer predominates
    - Tuning allowed

MDA: what is it?

- **The Model-Driven Architecture approach defines system functionality using a platform-independent model (PIM) using an appropriate domain-specific language:**
  - **Then, given a platform definition model (PDM) corresponding to CORBA, .NET, the Web, etc., the PIM is translated to one or more platform-specific models (PSMs) that computers can run**
  - **The PSM may use different Domain Specific Languages, or a General Purpose Language**
- **Automated tools generally perform this translation or mapping**

MDA: what is it?

- **MDA is related to multiple standards:**
  - **Unified Modeling Language (UML),**
  - **Meta-Object Facility (MOF),**
  - **XML Metadata Interchange (XMI),**
  - **Enterprise Distributed Object Computing (EDOC),**
  - **Software Process Engineering Metamodel (SPEM), and**
  - **Common Warehouse Metamodel (CWM)**
- **The term “architecture” in Model-driven architecture does not refer to the architecture of the system being modeled, but to the architecture of the various standards and model forms that serve as the technology basis for MDA**

MDA

- **Informal UML models provide**
  - **Informal modeling**
  - **Used to sketch out basic concepts**
  - **Advantages over other informal diagram techniques: it has some form of semantics**
  - **Not suited for code generators and interpretation**
    - **Analogously informal text can not be compiled and executed as 3GLs**

MDA

- **Formal UML models provide**
  - **Precise:**
    - **Precision and details are *not* the same**
  - **Computationally complete**
    - **Missing properties and unresolved references are not acceptable**
    - **3GL analogy ...**
      - **Incomplete programs can not be compiled**
  - **executable UML**

MDA: how does it work?

- **Platform Independent Model (PIM) in UML is developed by architect, no assumptions**
  - **on platform**
  - **on programming language**
  - **on databases**
  - **on architecture (2-tier vs 3-tier)**
- **High level of abstraction**

MDA: how does it work?

- PIM model is mapped to XMI, XML representation of UML
- PIM model is transformed into Platform Specific Model (PSM)
- Architectural decisions are resolved/instantiated
  - 2 tier vs 3 tier
  - CORBA
  - .NET

MDA: how does it work?

- The architecture implementation contains a series of declarative XML-based templates that generate “PSM” code
- Template resolves architectural issues for a certain layer
- Layers can be exchanged with other ones
- Mechanisms to provide architecture code extensions

MDA: does it work?

- **Important to model-driven architecture is the notion of model transformations:**
  - **QVT (Query/View/Transformation)**
  - **Xtend (openArchitectureWare)**
  - **ATL (ATLAS Transformation Language)**
  - **Plain Java**
  - **XSLT**

MDA

- Various MDA implementations:
  - Commercial:
    - OptimalJ from CompuWare (dead as far as I know)
    - Rational Rose (IBM)
    - OSLO (Microsoft) DSL based
  - Open Source Eclipse based:
    - OAW
    - ATLAS

Architecture-Driven Modernization

- ADM is the inverse of MDA
- Process of understanding and evolving existing software assets for the purpose of
  - Software improvement
  - Modifications
  - Interoperability
  - Refactoring
  - Restructuring
  - Reuse
  - Porting
  - Migration
  - Translation into another language
  - Enterprise application integration
  - Service-oriented architecture
  - MDA migration