



All You Wanted To Know About MDD (But Were Afraid To Ask)

ProSoftwarica GmbH
Technoparkstrasse 1
8005 Zürich
www.prosoftwarica.com

Milan Ignjatovic
abstractionist, architect,
coach, trainer, consultant
mig@prosoftwarica.com

Agenda

- Introduction
 - Problem and Solution
 - MDD: What, Where, How
- MDD experience
 - Model compiler approach
 - Case Study: Transformation Chain approach
 - Architecting Domain Metamodels in real Projects
- Our Business Patterns / Antipatterns

Separation of Concerns

- The fundamental Principle of Separation is:

Never let a machine do a human job
Never let a human do a machine job

- We believe in:

- human-architected systems realised by automating repeatable manual tasks
- using **models** to deal with the world in a focused manner, providing means for **managing** enormous, **evolving complexity** and **irreversibility of reality**, "The Nature of Modeling", Jeff Rothenberg

Problem Statement 1

- Much of the inaccuracy of software is due to the extremely detailed and sensitive nature of current programming languages - "the brick" metaphor
 - Minor lapses and barely detectable coding errors, such as misaligned pointers or uninitialized variables, can have unpredictable consequences
- A true story says that a single missing 'break' statement in a C program resulted in the loss of long-distance telephone service for a large part of the US
 - The economic damage caused was estimated to be in the hundreds of millions of dollars

Problem Statement 2

- We no more want to pay such a high price for simply migrating our information system to a new platform e.g. Java **because** our business domain is (quasi) stable.
- We are prepared to pay a last price for building **abstract models** of our business as guarantee against technological obsolescence
- Additionally, any platform provider will also have to provide the mapping solutions from standard business models to his own platform, before we commit.



The "human" Solution

- This “vulnerable” nature of mainstream software technologies, where a seemingly minute defect can have major effects on the overall system, makes it very difficult for **humans** to model software systems completely and accurately
- The only weapon we have for conquering growing complexity is **abstraction** (removal of unessential detail)
- How do we make better systems?
 - remove all dependencies to irrelevant detail and chaotic sensitivity from our daily activities!

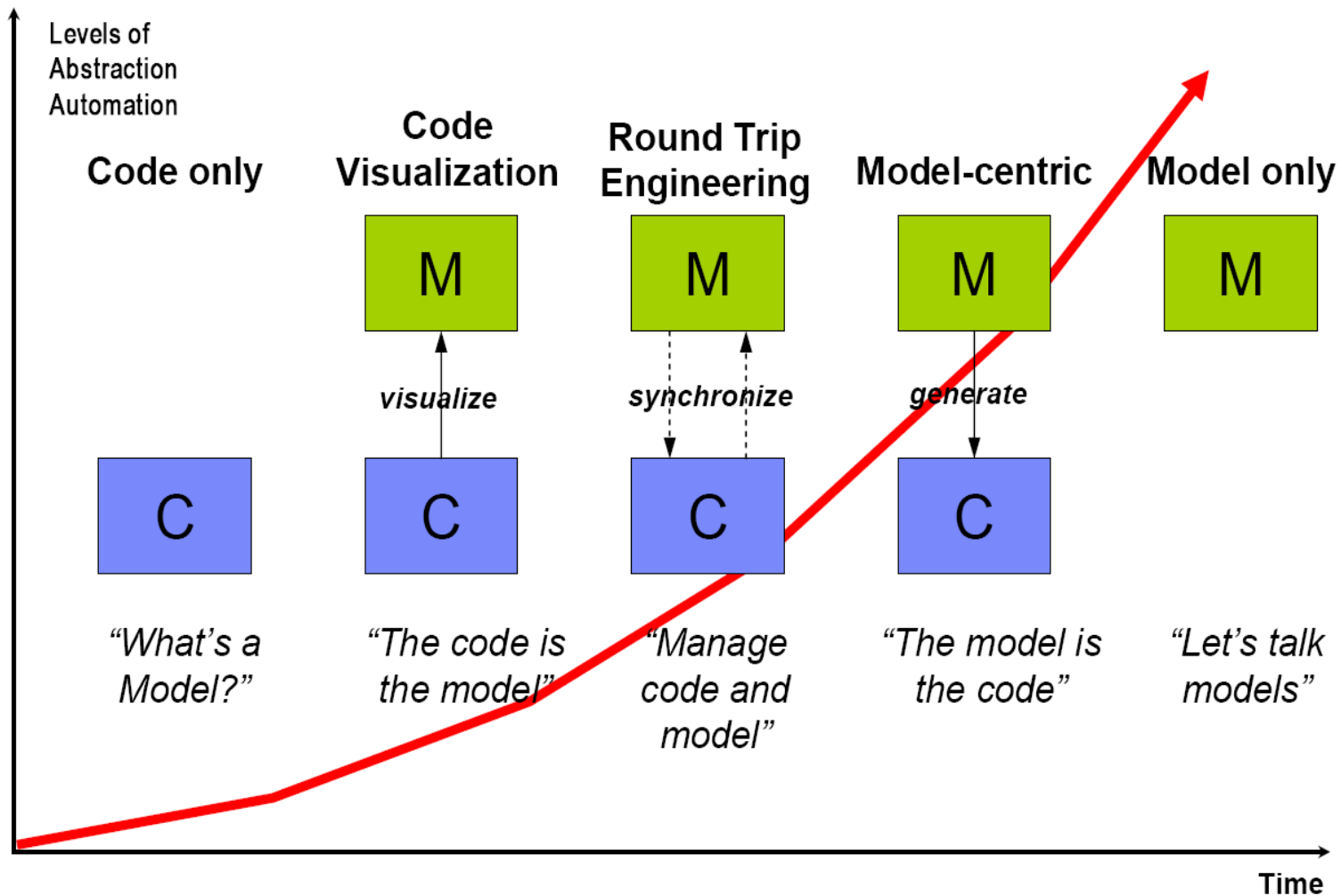
What is MDD? Human Software Development!

- Shifting of focus away from program code towards models as primary artifacts of software design
- By judicious selection of software abstractions and through the precise definition of their semantics, the **transition** from an **abstraction** into its **software realization** can be **automated** without loss of accuracy
- This combination of **abstraction** and **automation** has inspired a set of modelling technologies and corresponding development methods collectively referred to as **Model Driven Development**

Where is MDD today?

- Model-driven methods of software development are not particularly new and have been used in the past with varying degrees of success
- MDD is receiving more attention today because supporting technologies have matured to the point where **automation** can be used in practical applications
- ...and this is due to new technologies being much more scalable, more efficient, and easily integrated with existing tools and methods than was ever possible in the past

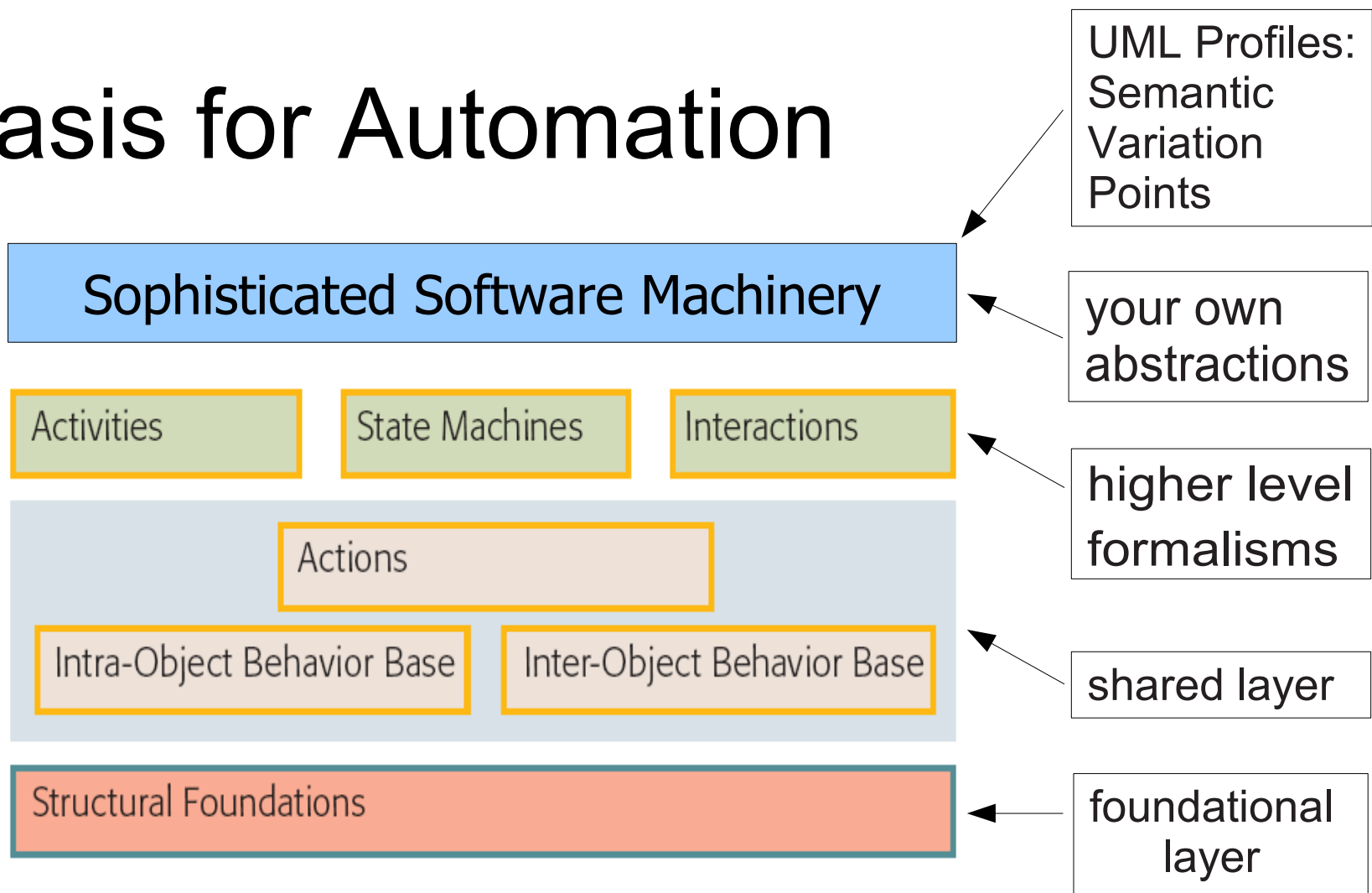
MDD Maturity Model and Styles



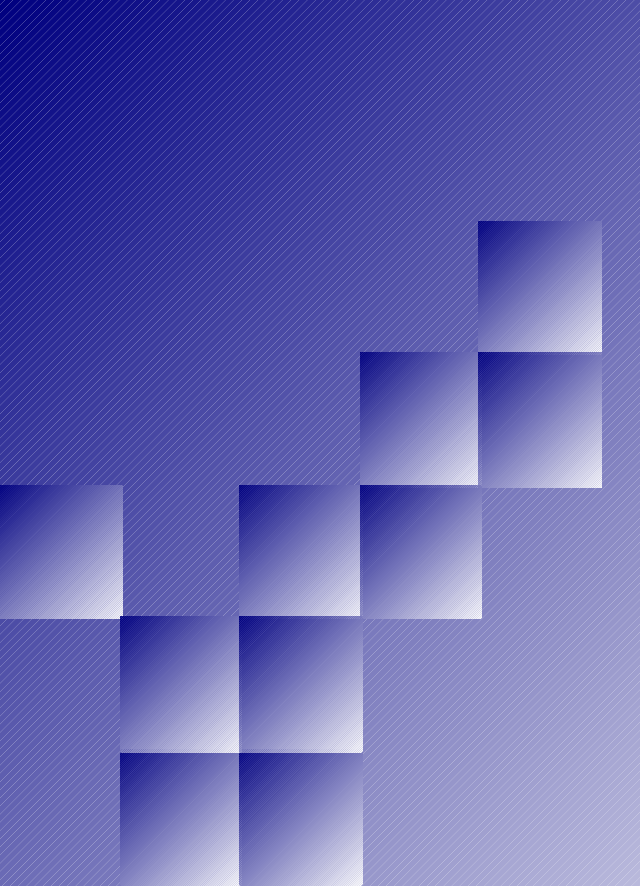
How to MDD? We need Automation!

- The more automation, the more accurate the models and the greater are the benefits of MDD
- One very effective solution is to formally link a model with its corresponding software implementation through **one or a series** of automated model transformations
- Perhaps the best and most successful exemplar of that approach is the compiler, which automatically translates a high-level language program into an equivalent machine language representation

Basis for Automation



The UML2 semantic framework

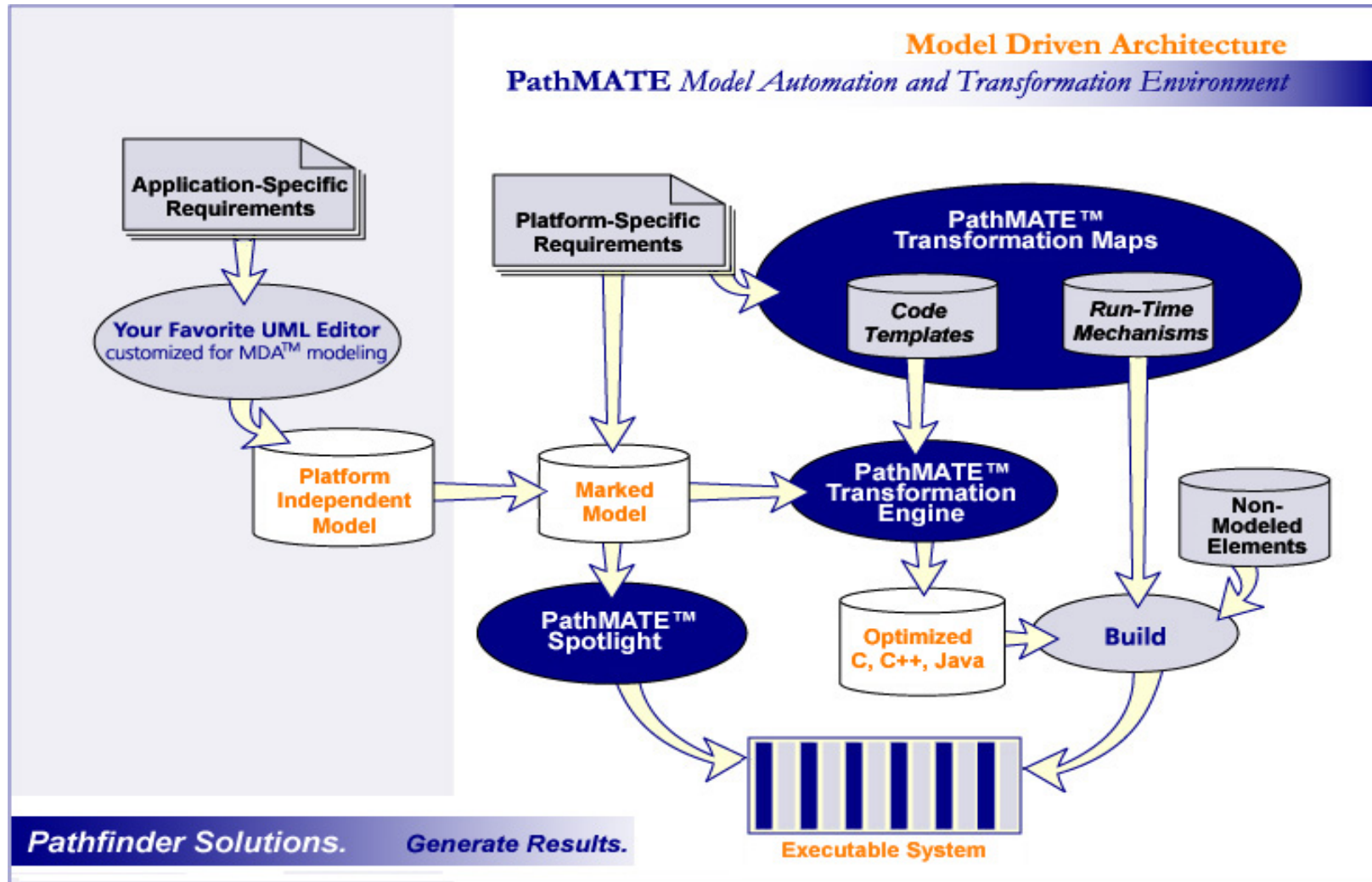


Model Compiler Approach

Pathfinder Solutions

- Pathfinder Solutions
 - Mission: *Transform expertise into a high performance software development advantage*
- Leading provider of MDD and MDA technology, training and expertise
 - Formed in 1995 to bring more complete solutions to customers
 - Exclusive focus is on applying model based techniques and technology to the development of high performance systems
 - Provide tools, components, training, and mentoring
- Active member of the OMG
- Offices in Foxboro MA (Boston)

PathMATE Environment



PathMATE: 3 top entities

- The three parts of the PathMATE toolsuite cooperate to turn your models into executable systems:
 - *Transformation Maps* – Generate C, C++, or Java software with off-the-shelf Transformation Maps, or create custom maps to drive output for other languages or specific platforms
 - *Transformation Engine* – The Engine transforms platform independent models into working, embedded software applications
 - *Spotlight* – Verify and debug your application logic with Spotlight, the most advanced model testing environment available.

Relationship with Pathfinder

- ProSoftwarica is the Pathfinder arm in Europe
 - We provide customers with PathMATE training and mentoring
 - We apply Pathfinder model-driven methods and tools to deliver key business results to our customers
- ProSoftwarica delivers on-site pre- and post-sales support for Pathfinder technologies throughout Western Europe
- We are branded: „Ready for Rational Software“
 - Tight integration with the IBM RSx family of products

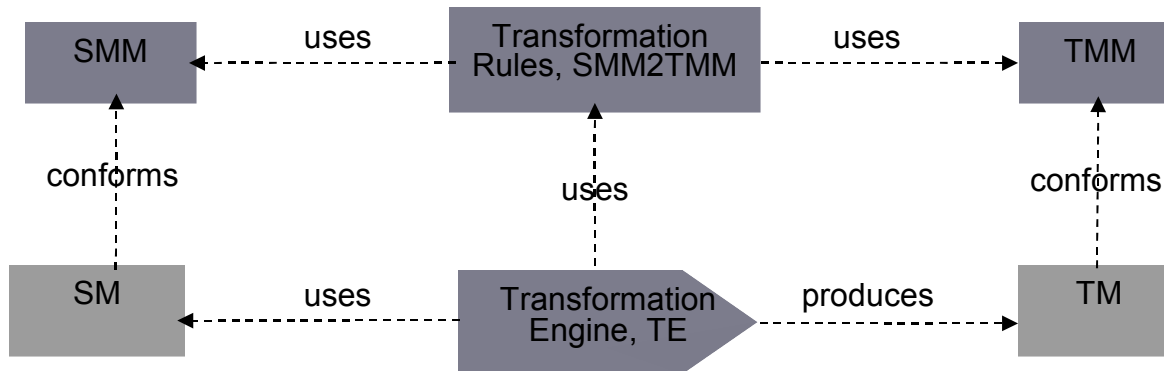


Case Study: Transformation Chain Approach

Case Study Objectives

- Analyse and design transformation chains based on the GTA Architecture using separation of concerns and other architectural principles
- Realise generic x2y, and specifically uml2uml, transformations using QVT or similar transformation languages
- Apply mof2text transformations
- Consider: reuse, scalability, programmatic application of UML profiles, automated transformation chain execution and timing thereof
- Automation for Visualisation, Debugging, Testing and Deployment of generated artifacts
- The objective is **not** to simply generate code by using UML profiles

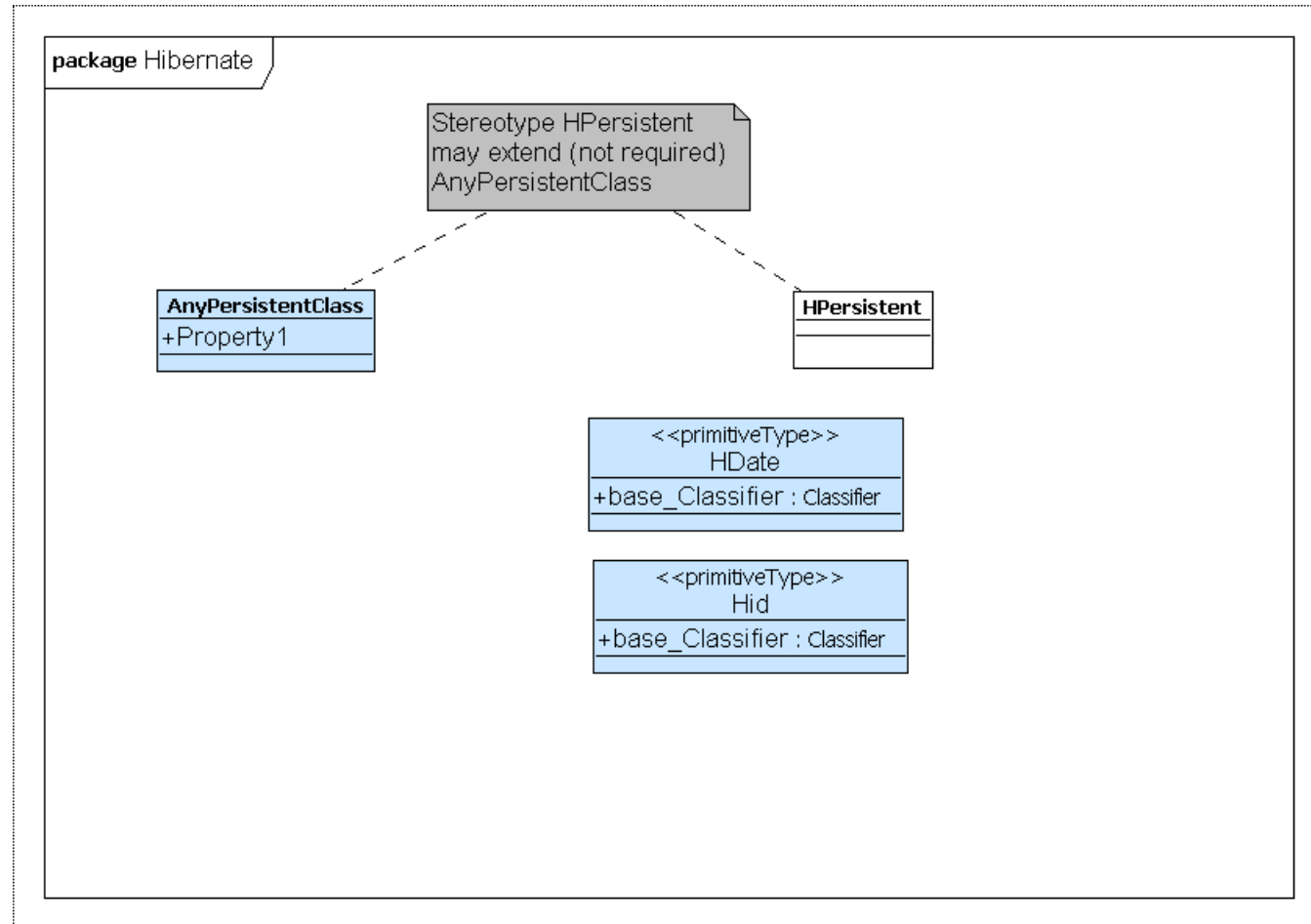
GTA - simplified



- SM: source model, conforming to the source meta model SMM
- TM: target model, conforming to the target meta model TMM
- SMM2TMM: set of transformation rules for transforming models of type SMM into models of type TMM
- TE: transformation engine uses a set of transformation rules and a source model SM to transform it and produce TM as output

Simple Hibernate UML Profile

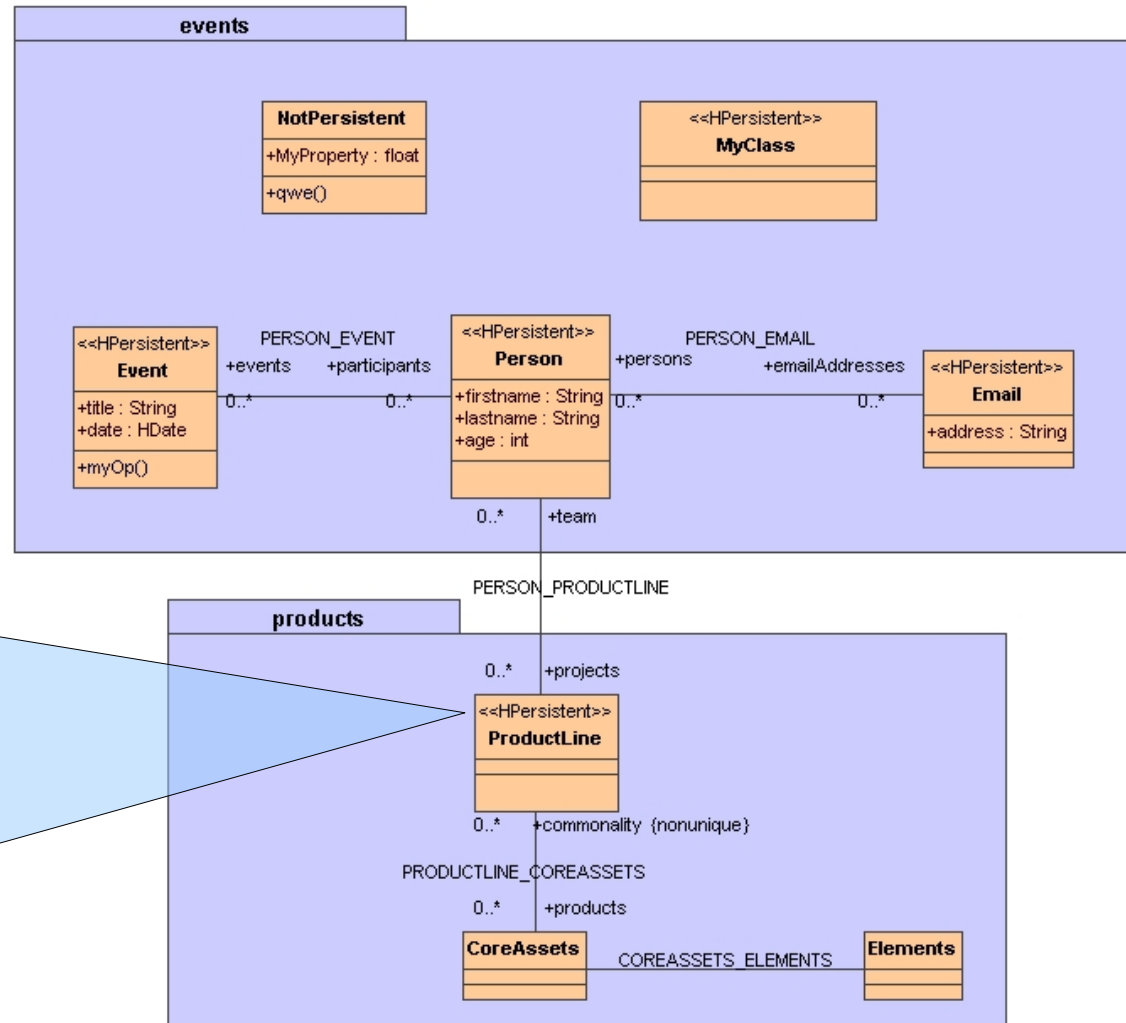
We designed a trivial UML profile to introduce persistence using the Hibernate framework.



Profile application to a PIM

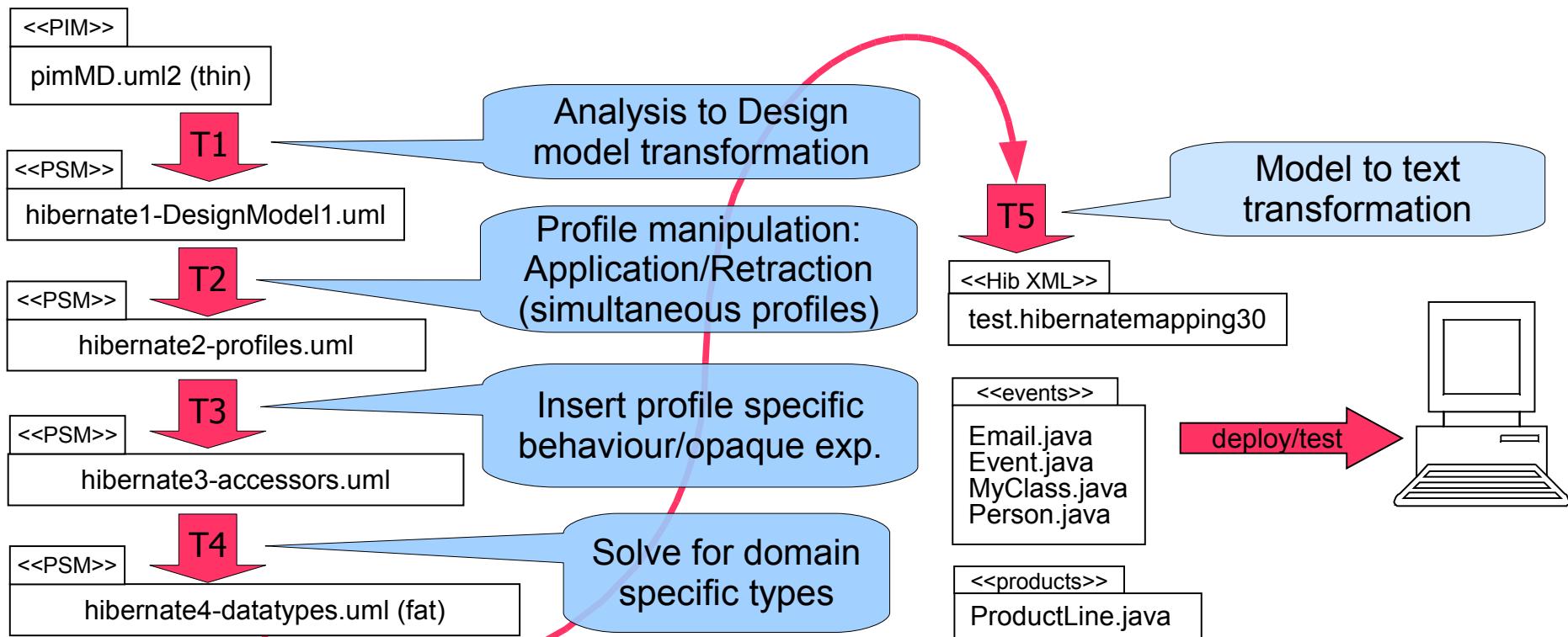
We apply the profile to the PIM and use the HPersistent stereotype where appropriate.

Using an automated transformation chain, we transition from this abstraction into its software realization




Transformation Chain

- A PIM is refined by a number of successive transformations
- PIM2PIM, PIM2PSM, PSM2PSM, ..., PSM2Text





Example: Architecting Domain Metamodels



Example:
embedded,
real-time, high
performance
system
architecture

For more info please contact the author



A partial view of the Domain Metamodel

For more info please contact the author



Our Business Patterns

A generic approach

- We presently use ATAM to do assessments and reviews with our clients and/or end customers, thereby:
 - We get to know the business domain of the end customer
 - We get to know the competence of end customer's development team
 - We identify shortages of the end customer's process and architecture with respect to MDSD and document pitfalls of the applied technologies
 - We point out to MDSD technologies to alleviate shortages
- We train our client and/or end customer with proposed technology by delivering training and on-the-job coaching
- We show our competence and are called back in on “as needed” basis

Business Pattern

- “All Software Aids are good”
 - There may be a “software aid” applied in a client's project
 - We think there are none-so-bad software tools on the market that we cannot enhance and win hearts in a project
- “Buy-in of the Team”

We evangelise a new technology, we demonstrate its feasibility using a prototype, we confirm applicability:

 - We realise it on condition of commitment by the CEO
 - E.g. we build a domain metamodel and provide commercial or custom tools to enable maximum automation of the life cycle
 - This pattern is usually associated with firing of the CTO and technology laggards in the team.
CTO is the person who uses vi to code sql statements. He has a team which is permanently coding, manually testing, bugfixing...



A Business Anti-Pattern

For more info please contact the author



For more info

- <http://www.ProSoftwarica.com>