

A swedish friend asked:

What is this "uml" that I see everywhere on the web?

Humla : Swedish for bumble-bee.

The old story about the



"Depending on its weight in relation to the size of its wings, it shouldn't be able to fly!!!"

The bumble-bee doesn't know that...

What's wrong with the flying ability of the BUMBLE-BEE?



- Confusing travel plan ?
- Too big, heavy and clumsy ?
- Inefficiently or low-powered wings ?
- Primitive or unreliable flight control system ?

Sometimes special purpose mutations are invented, like:

BUMBLE-BEE RT



EMBEDDED BUMBLE-BEE

But they are still based on the same old bumble-bee, (but in a new fancy costume) that can not fly safely and will eventually crash, if they ever manage to take off.

Then comes the more expressive:

BUML-B 2.0

…carrying a lot of new equipment!



The main question must be:



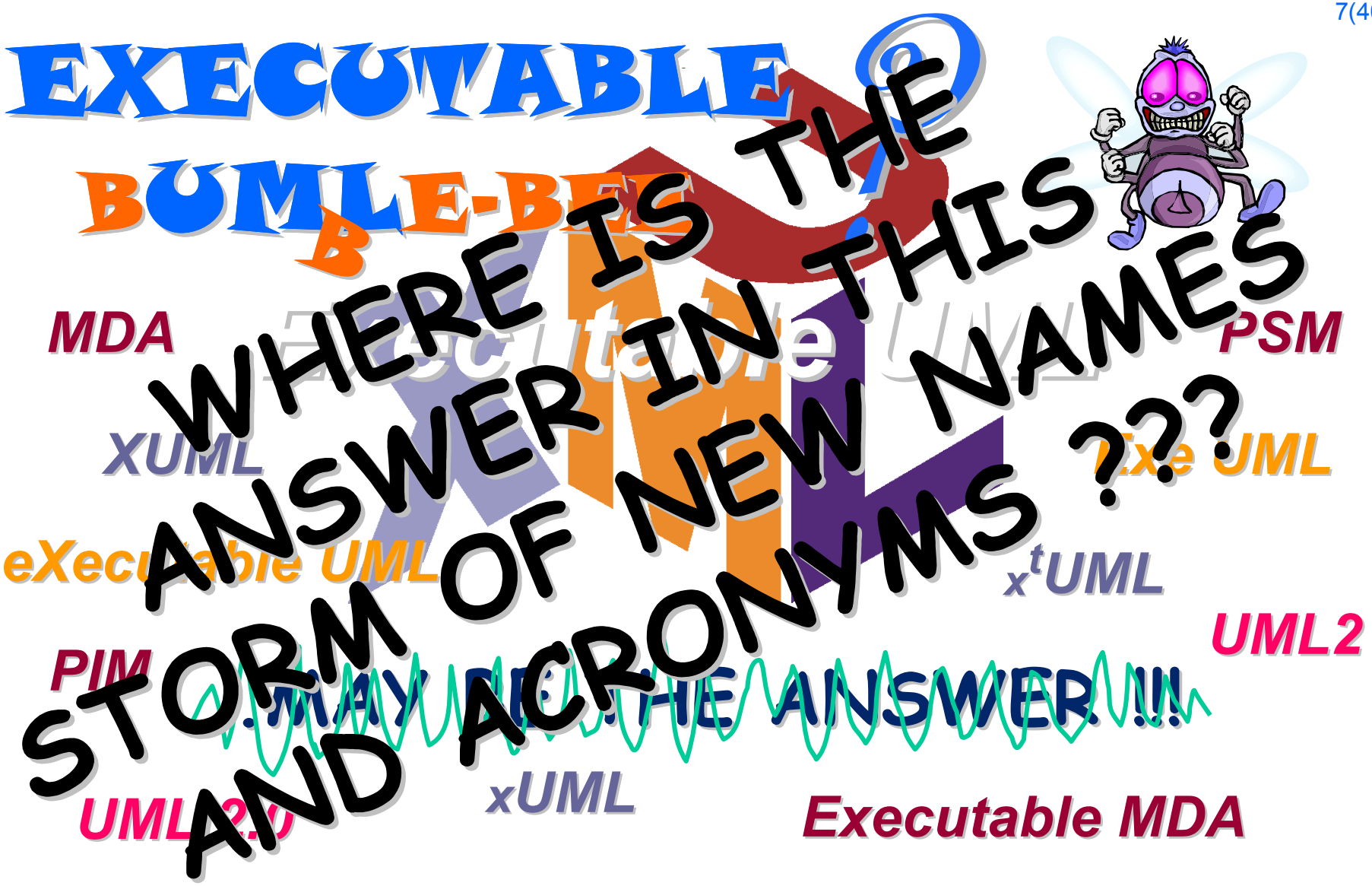
Does it really fly higher and/or safer with even more extra weight to carry?

…and without any fundamental improvements of its flying capabilities?

WHAT IS REQUIRED TO MAKE THE UML FLY REALLY HIGH ?



- Guidance by an easy to learn methodology !
- A reliable Specification Technology !
- An efficient Code Generation Technology !
- Fully automated tool support for all the above !


EXECUTABLE ?
BUMLE-BEE
WHERE IS THE ANSWER IN THIS STORM OF NEW NAMES AND ACRONYMS ???
MDA
PSM
XUML
eXecutable UML
PIM
UML2.0
xUML
Executable MDA
UML2



...MAY BE THE ANSWER !!!

WHAT IS THE QUESTION ???

HOW CAN SYSTEMS BE GENERATED FROM EXECUTABLE MODELS ?



BOTH SOFTWARE...

...AND HARDWARE



Content of this Presentation

- System Design Problem Descriptions
- Results from Pilot Projects
- The Model Compilers

The System Design Problems

- The cultural differences between the SW and HW disciplines. SW and HW engineers have different backgrounds, training and experience (tools).
- SW and HW engineers often have difficulties to communicate analysis and design decisions between each other, simply because...
- They speak different languages!
- This frequently results in lead time consuming inconsistency problems in the design.

The Partitioning Problems

- Partitioning into software and hardware is often (typically) made very early in a project.
- Decisions are most often based on local interests, old habits, current knowledge and experience.
- Other factors are e.g. reuse of existing legacy code and the human resources available to the project.
- The consequences in e.g. lead-time or product performance of a partitioning are rarely analysed.
- The early decision is often the final solution.

How can this be solved ?

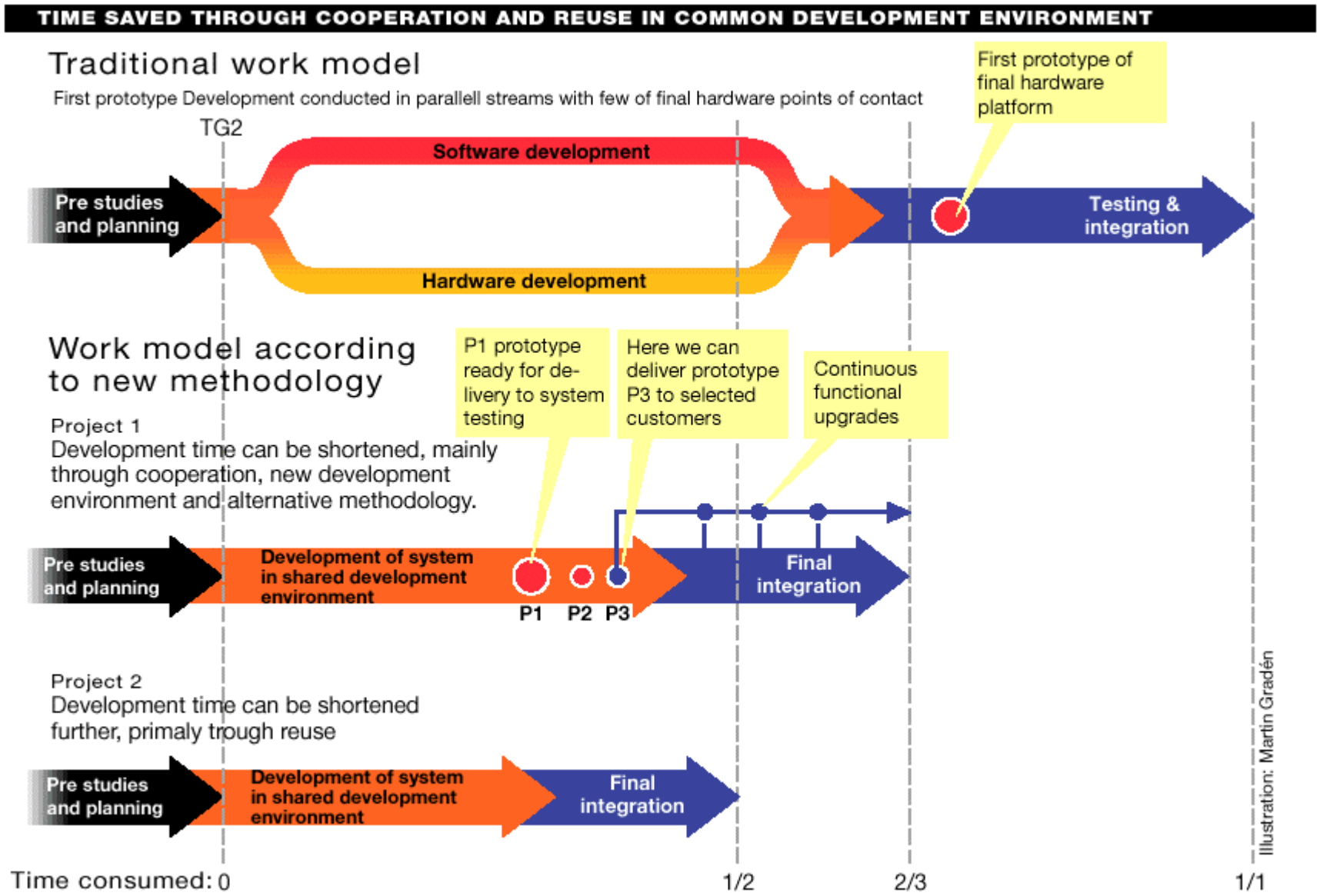
- An environment that allows us to prototype and develop executable and verifiable specifications on a system level (implementation independent).
- These specifications must be possible to variably partition and automatically translate into different alternative designs late in the project.
- But first, we need a common and implementation independent system specification language!
- Why not base this language on the UML?

Some of all the benefits: 1(2)

- This approach to design will bring the SW and HW engineers together already in the early phases of a project to analyse system requirements and specify system behaviour as a team.
- We get the possibility to evaluate different design alternatives late in projects, even after 1st delivery, with the purpose of finding the most efficient partitioning between software and hardware.
- We can ship a prototype early (perhaps SW only) and re-partition the design in the next release.

Some of all the benefits: 2(2)

- When translating *Executable Specifications* into design we are also reducing the hand-coding, design inconsistency and SW/FW/HW interface errors to a minimum, consequently also...
- …reducing the ***Time To Market*** to become ***First To Market*** with a higher quality product.
- This is of course compared to hand-coding and compared to the elaborative approach to design (= models not reusable across different platforms).


TIME SAVED THROUGH COOPERATION AND REUSE IN COMMON DEVELOPMENT ENVIRONMENT
Traditional work model
First prototype Development conducted in parallell streams with few of final hardware points of contact
TG2
Software development
Pre studies and planning
Hardware development
First prototype of final hardware platform
Testing & integration
Work model according to new methodology
P1 prototype ready for de-livery to system testing
Here we can deliver prototype P3 to selected customers
Continuous functional upgrades
Project 1
Development time can be shortened, mainly through cooperation, new development environment and alternative methodology.
Pre studies and planning
Development of system in shared development environment
P1
P2
P3
Final integration
Project 2
Development time can be shortened further, primaly trough reuse
Pre studies and planning
Development of system in shared development environment
Final integration
Time consumed: 0
1/2
2/3
1/1
Illustration: Martin Gradén



Many questions are important to us:

- How can we get truly *Executable Models* possible to execute and verify before we generate code?
- How can we generate both software and hardware?
- How can we get control of the code performance?
- How can we achieve large-scale reuse across different platforms and for alternative designs?
- **ANSWER:** Through ***Executable UML Specifications***

Executable UML Specifications



The learning steps from Structured Programming to Executable UML Translation

2000's

1990's

1980's

1970's

1960's

INTELLECTUAL PROPERTY INVESTMENTS

LEARN - IMPROVE - EVOLVE - GAIN

Translation into SW and HW

Executable UML Specifications

RUP

UML Oriented Project Management

UML Based Development

C++, Ada95, Java

Model Based Development



Object Oriented Methods



High Level Languages

Assembly Code

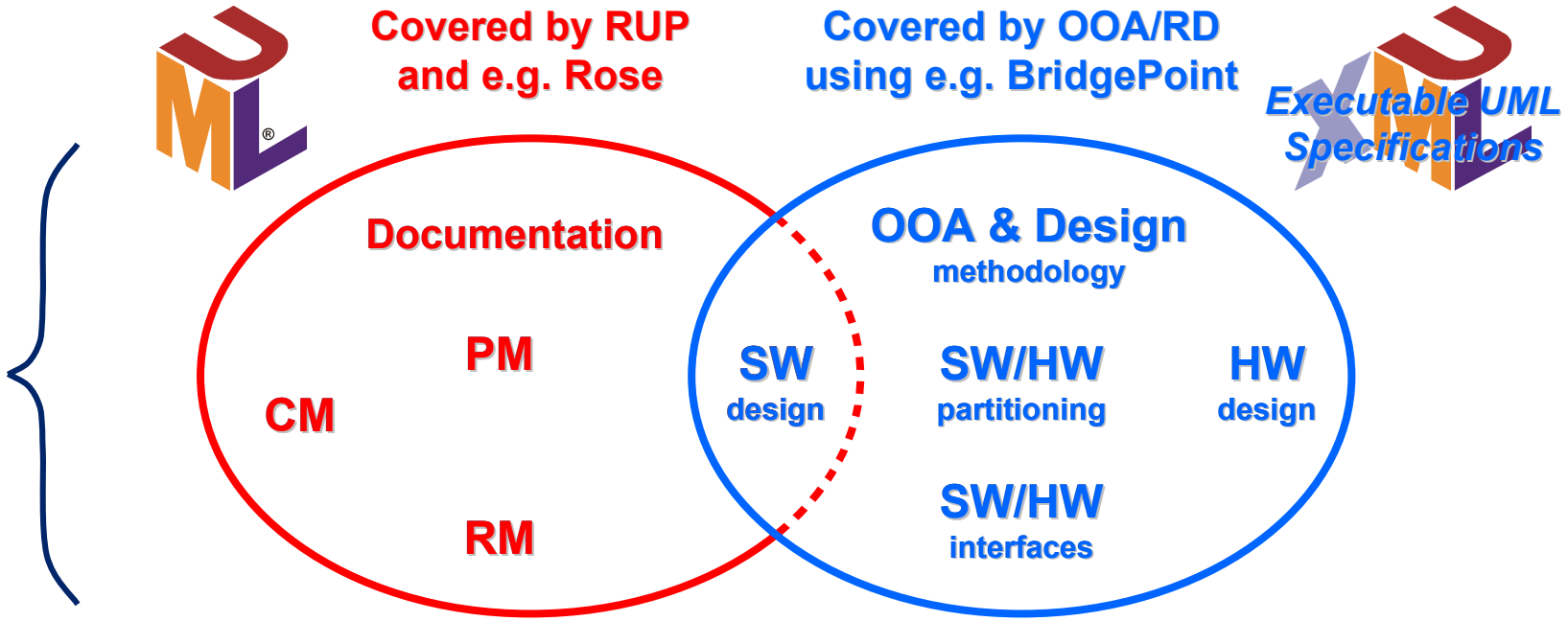
Executable UML Specifications **are implementation independent, can be run, tested and debugged much like a program before any code is generated.**

Executable UML Specifications **are translated into design by *Model Compilers* and provides large-scale reuse across different platforms and alternative designs, such as C++, Java, Plex-C, Erlang, VHDL, Verilog, SystemC...**

Is RUP good enough ?

(Rational Unified Process)

Necessary technologies to cover for Systems Development

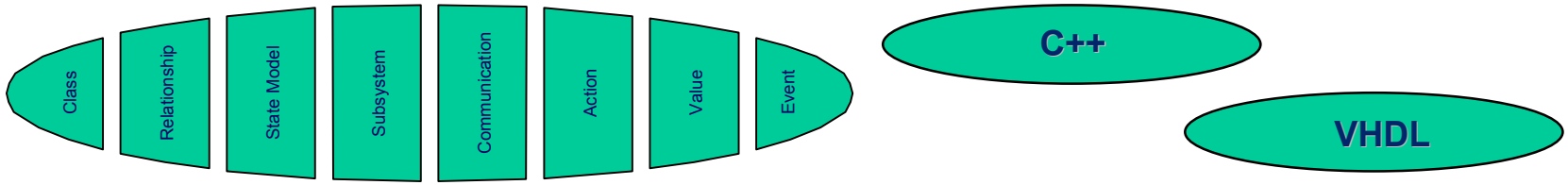


- Approach to design: *Elaboration*
- *Hacking With Pictures* (often C++)
- Platform dependent models
- Prevents reuse across platforms

- Approach to design: *Translation*
- *Executable UML Specifications*
- Platform independent models
- Large-scale reuse across platforms

Executable UML Specifications

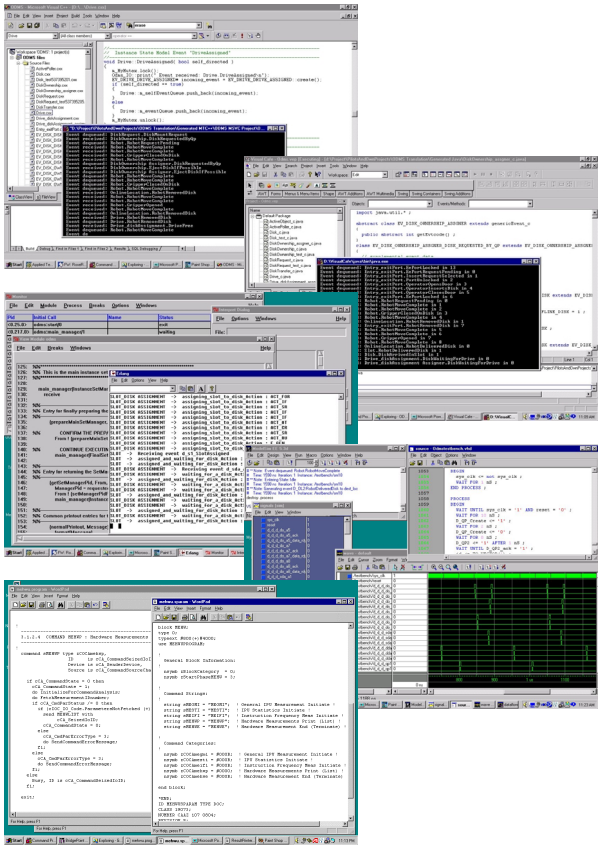
Open Code Generation using BridgePoint®



- The code generation technology is *Open*.
- Allows 1st class user-defined extensions of the metamodel for *Model Driven Architectures*.
- Code Generation and Competence are Intellectual Properties owned by Ericsson (we are the experts).

Executable UML Specifications

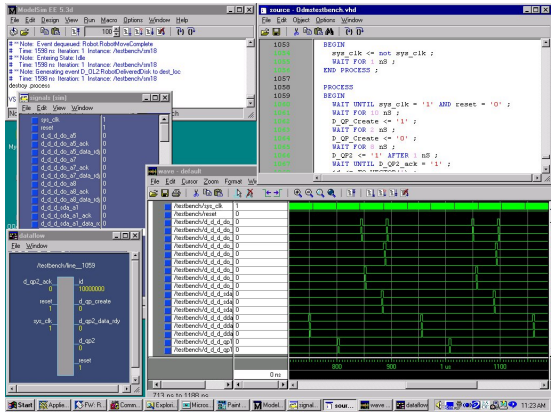
Open Translation of Executable UML *Projects*

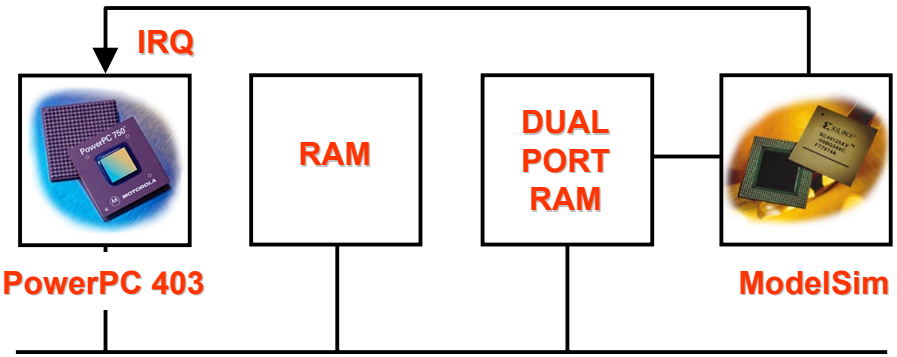


- During a number of pilot projects we have developed a number of different Model Compilers for translation of *Executable UML* models into:
- Erlang, C, Plex-C and C++ (VHDL)
- Both the generated Plex-C and C++ code for the AXE got as efficient as hand-written code at the first attempt.
- A new Model Compiler for going from UML-to-Silicon is being developed.

Executable UML Specifications

Translation into VHDL and C++ *(1st co-design pilot)*

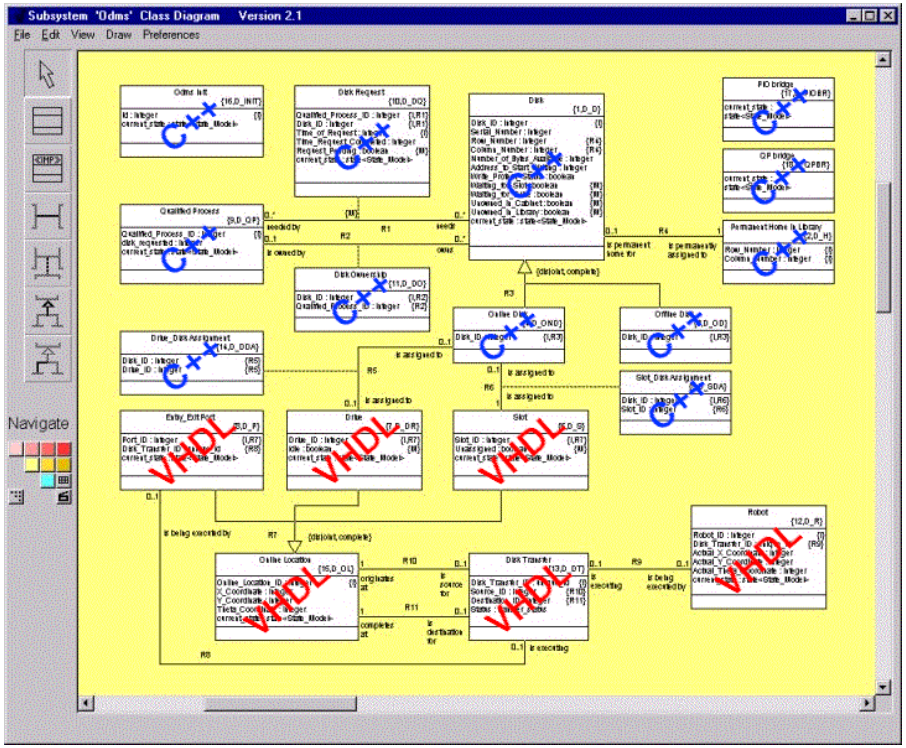




- Generate C++ and behavioural VHDL from the same model.
- New Model Compilers for C++ and VHDL developed (3 months)
- Generated VHDL executed and verified using ModelSim.
- Test & Integration in Seamless.
- DP RAM Interface Protocol fully managed by Model Compilers.

Executable UML Specifications

Translation into VHDL and C++ *(1st co-design pilot)*



- We apply a model-based partitioning between software and hardware.
- We have proved that partitioning no longer need to be an early decision. It can be made later.
- We can re-partition the modelled system at any time, even after the 1st delivery, and then simply re-translate the entire modelled system into a different design.



Executable UML to C++ for the AXE OS *(4th pilot)*

- Reverse engineering of a part of the AXE Operating System into an Executable UML model.
- Development of a Model Compiler that translates this model into C++ optimized for the AXE OS.
- The generated code must comply to the local design rules and guidelines for C++.
- Code optimized for memory cost and performance.



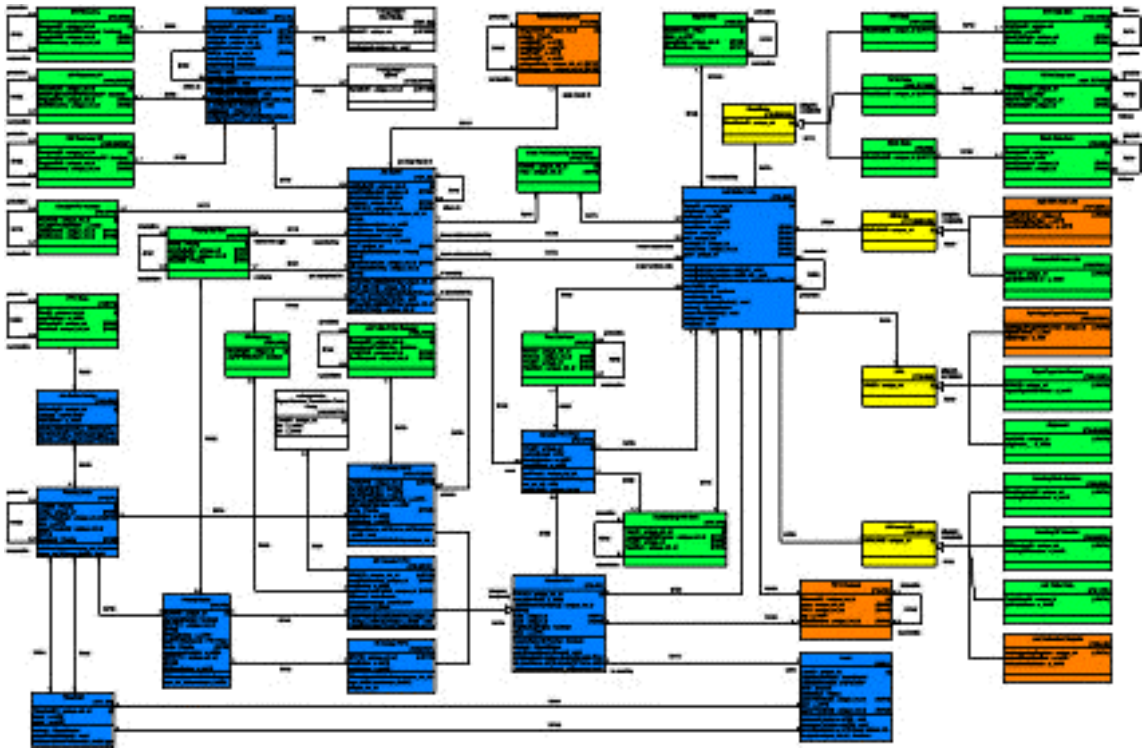
Executable UML to C++ for the AXE *Requirements*

- Seamless integration of the generated code.
- Small-scale introduction of translation technology (just a small part of a large-scale system: the AXE).
- Legacy system not allowed to be affected (at all).
- No introduction of new (unpredictable) risks.
- No more than 20% overhead in memory cost and performance compared to the hand-coded C++.

Executable UML Specifications

Executable UML to C++ for the AXE OS *(4th pilot)*

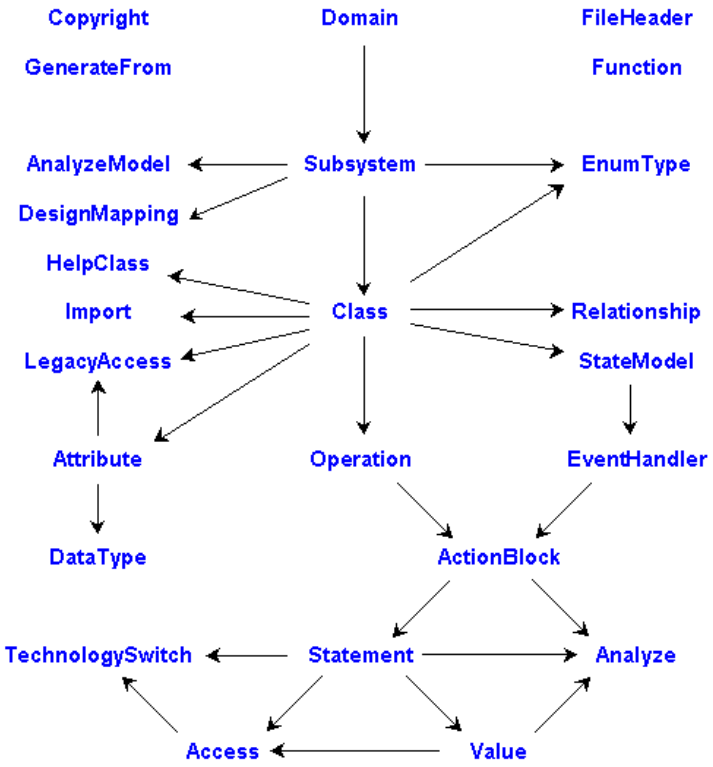
Reverse engineered Class Model of the JobBuffer.



- A primitive Model Compiler translates only into C++ classes.
- **THIS ONE DOES NOT**
- This Model Compiler translates a model into **classes**, **structures**, **unions** & **variables** and generates C++ code that is optimized for high performance.

Executable UML Specifications

Executable UML to C++ for the AXE OS *(4th pilot)*

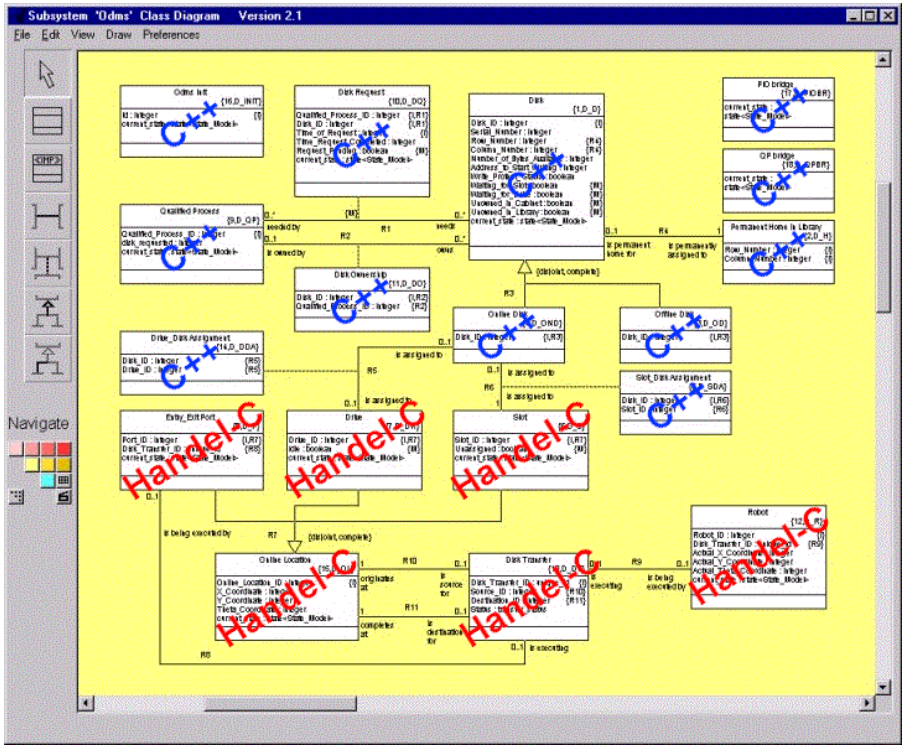


THE MODEL COMPILER:

- Generates standard C++ with switches for generating code that is memory & performance optimized for the VM in the AXE OS.
- Verified this far is that generated code require no additional data memory (0%) and no additional program memory (~0%) compared to hand-coded version.

Executable UML Specifications

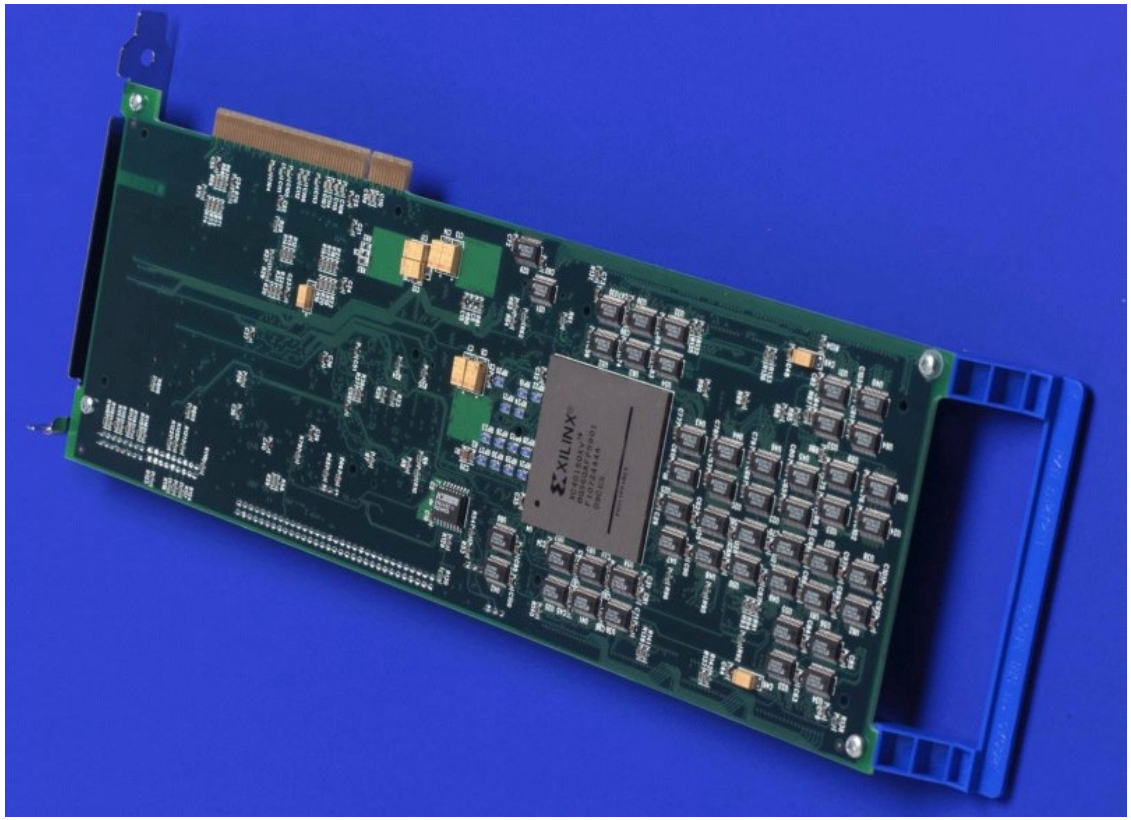
Translation to Handel-C and C/C++ *(ongoing project)*



- Develop a new Model Compiler that generates Handel-C (HW).
- Integrate the Model Compiler for Handel-C with our latest Model Compiler for C or C++.
- Model-based partitioning.
- Generate real silicon from UML.
- Run the synthesized Handel-C on In-System Reconfigurable Hardware (ISR) from Celoxica.

Executable UML Specifications

Translation to Handel-C and C/C++ *(ongoing project)*



RC1000-PP
from Celoxica

Single FPGA
(XILINX)

PCI-bus

2MB on-board
Memory



Translation to Handel-C

- Translation of Executable UML to hardware is not more difficult than translating to C++ optimized for performance for the AXE OS. **It's different.**
- One of the main differences is that hardware has the quality of being able to execute multiple *things* in parallel (during the very same clock cycle).
- So… Everyone who knows about Handel-C asks:



Translation to Handel-C

What about the `par` keyword?

Yeah… What about it?

Executable UML Specifications

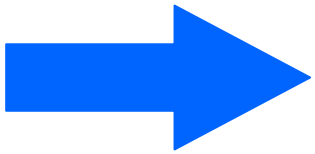
Translation to Handel-C

A SIMPLE EXAMPLE

TRANSLATION RULE: (applied automatically system wide)

Sequential assignment statements that does not share any left- and right-side variables can be enclosed in a **`par`** *block.*

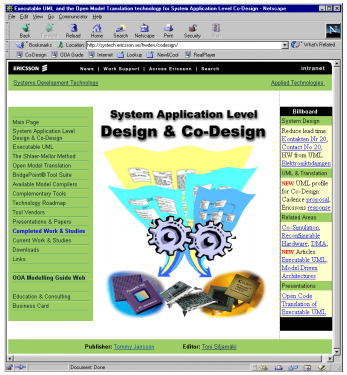
```
A = B
C = D + 3
B = C << 2
```

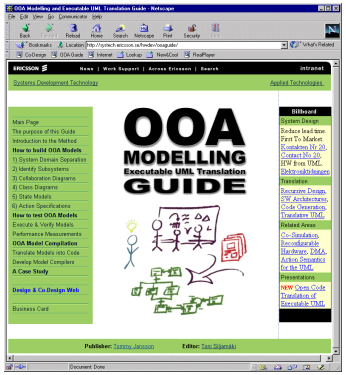


```
par
{
   A = B
   C = D + 3
}
B = C << 2
```

Model Compilers and the Target Platform

A proven and successful OOA Concept Mapping Strategy in 10 steps
- Initial considerations
- Step 1: Class
- Step 2: Class and Instance memory allocation
- Step 3: Instance Reference
- Step 4: Instance Reference Set
- Step 5: Class and Instance Access
- Step 6: Passing of Arguments and Return Values
- Step 7: Methods and Implicit Service Functions
- Step 8: State Model
- Step 9: Atomic Accesses and Actions
- Step 10: Model Compiler Structure



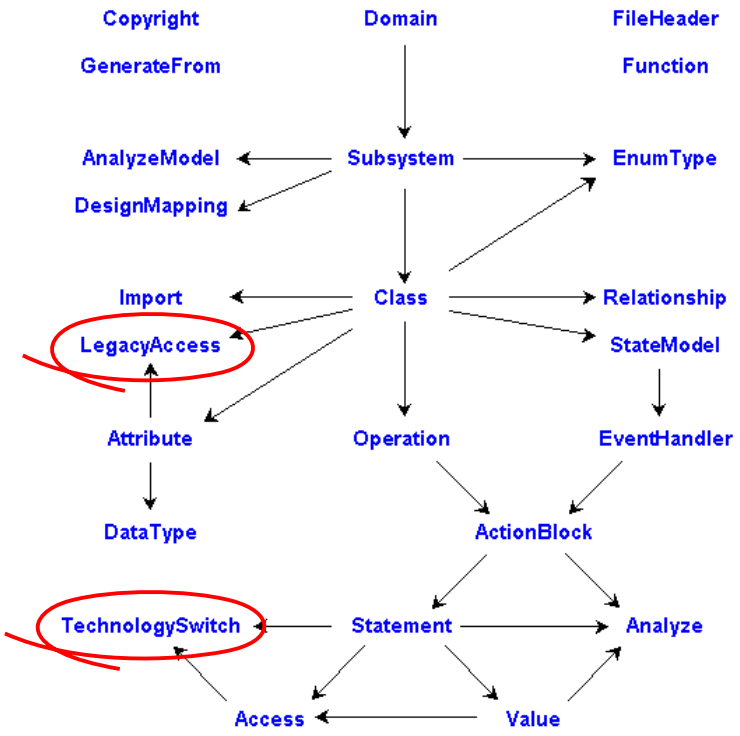


COMPILER DEVELOPMENT:

- We have developed 5 different Model Compilers in 3 years for translation of Executable UML to Erlang, C, Plex-C, C++ & Handel-C.
- The Model Compilers are based on a well documented and proven Mapping Strategy for the transition from analysis to implementation.
- The Model Compiler Structure is one of the most important parts.

Executable UML Specifications

Model Compilers and the Target Platform



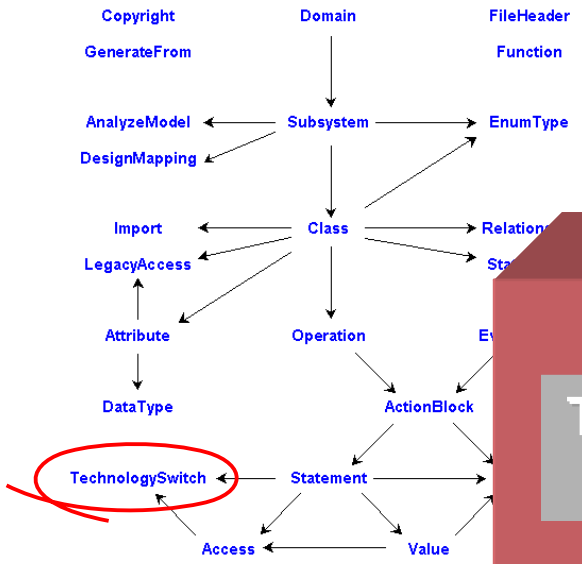
MODEL COMPILER STRUCTURE:

- This is a start-up structure on which all our Model Compilers are based.
- It has 2 important Archetypes:
- The **LegacyAccess** Archetype provides seamless integration of generated code with legacy code.
- The **TechnologySwitch** Archetype handels integration of different implementation technologies.

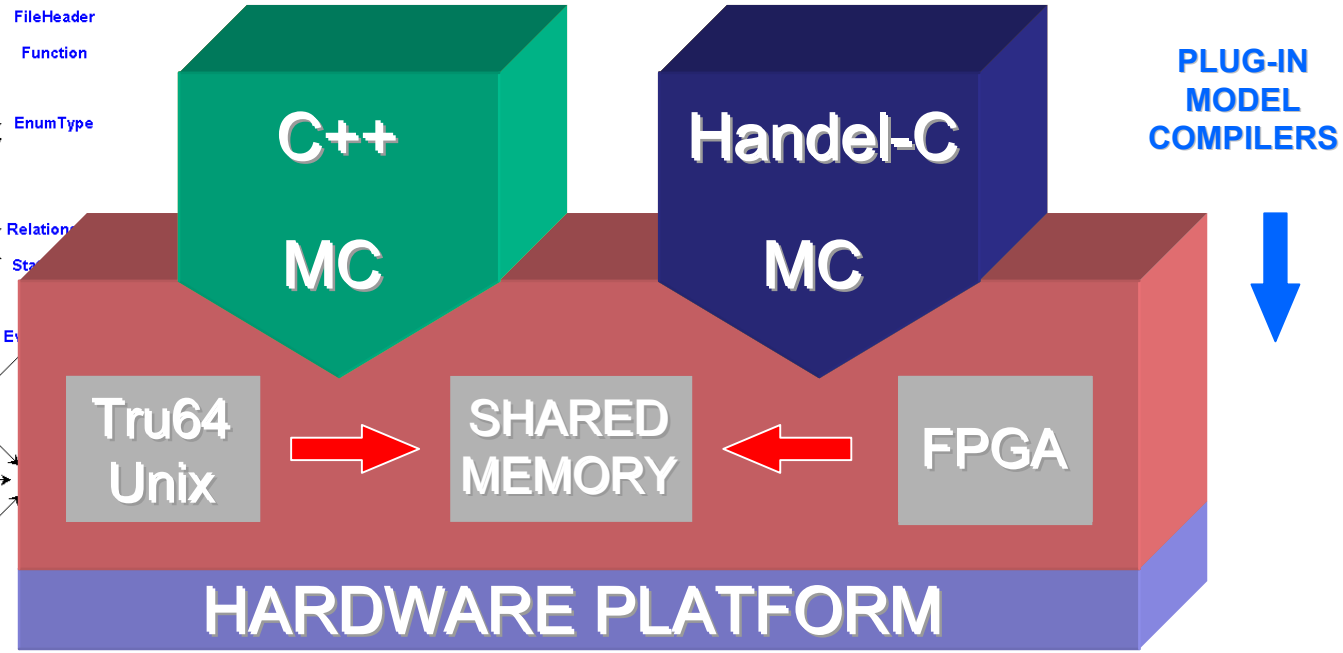
Executable UML Specifications

Model Compilers and the Target Platform

MODEL COMPILER STRUCTURE



Integrated Model Compilers



SOFTWARE PLATFORM : Legacy Code, Operating System etc.

Executable UML Specifications

Model Compilers for download at Ericsson





Open Translation of Executable UML

Proven Benefits (this far) 1(2)

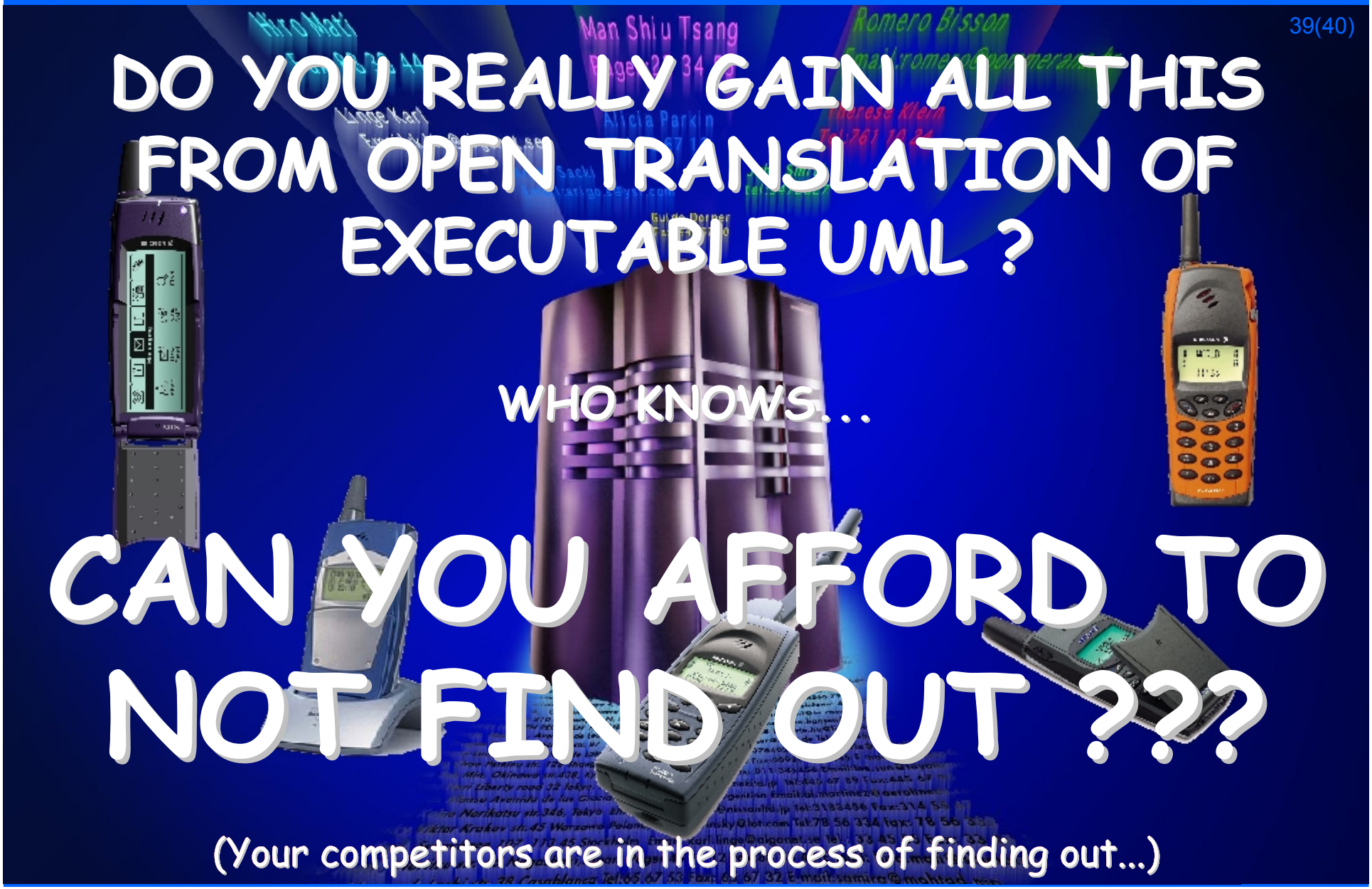
- Always and proven first time success.
- SW and HW engineers develop systems as teams.
- Analysis: *Executable UML Specifications.*
- Analysis Models are 100% reusable across different standard or proprietary SW/HW platforms.
- Design: Model Compilers (reusable expert systems).
- We can generate as efficient software code as hand-written code whenever we need to.

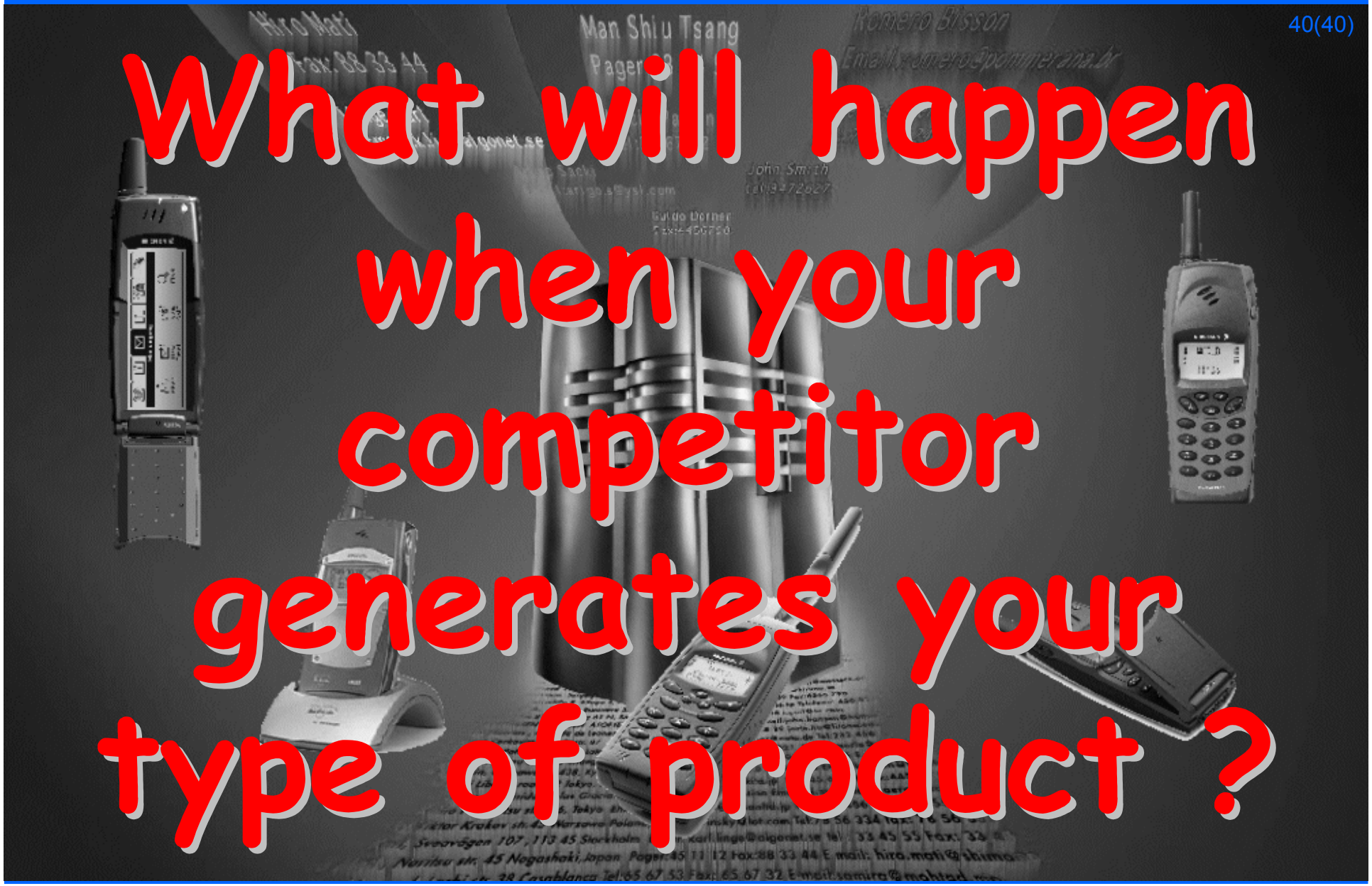


Open Translation of Executable UML

Proven Benefits (this far) 2(2)

- Small-scale introduction in a large-scale system.
- Seamless integration with existing legacy code and platforms (both software and hardware).
- Generation of Software, Firmware & Hardare with automated handling of all SW/FW/HW interfaces.
- System performance controlled solely by Ericsson.
- Competence & control stays within Ericsson, i.e. not available to tool vendors and competitors.


DO YOU REALLY GAIN ALL THIS FROM OPEN TRANSLATION OF EXECUTABLE UML ?
WHO KNOWS...
CAN YOU AFFORD TO NOT FIND OUT ???
(Your competitors are in the process of finding out...)


What will happen
when your
competitor
generates your
type of product ?