



## Model Based Development of Large Embedded Systems in Distributed Organisations

Peter Cigéhn

Senior Specialist UMTS RBS SW Architecture

[peter.cigehn@tietoerator.com](mailto:peter.cigehn@tietoerator.com)

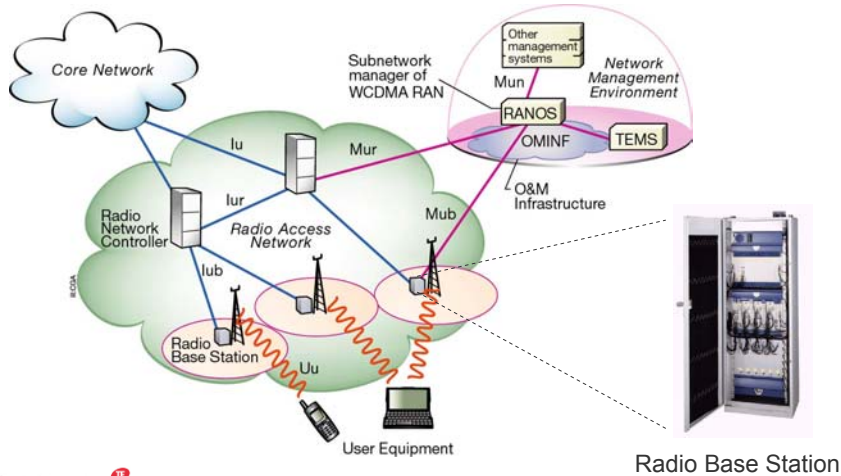
**TietoEnator**<sup>TE</sup>  
Building the Information Society

## My Background

- Master of Science in Computer Science and Engineering, 1992, Luleå University of Technology, with specialization in Software Engineering.
- Started working 1993 at Ericsson Erisoft in Umeå. Been working there since then although the company has changed names and owners over time and today we are a part of TietoEnator Telecom & Media.
- Working with development of control software for 3G WCDMA Radio Base Stations since 1996.
  - 1996 - 1998: Experimental 3G WCDMA system to NTT DoCoMo, Japan
  - 1998 -: Development of commercial 3G WCDMA/UMTS systems
- In 2002: Appointed as Senior Specialist UMTS RBS SW Architecture.

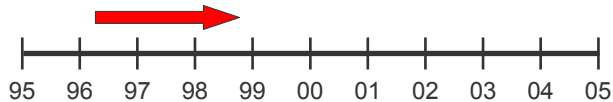
**TietoEnator**<sup>TE</sup>  
Building the Information Society

## Our Domain: The Radio Access Network



**TietoEnator**<sup>TE</sup>  
Building the Information Society

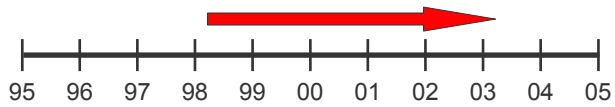
## Our Past



- During the fall of 1996 we started the development of an experimental 3G WCDMA system to NTT DoCoMo, Japan
- We were responsible for developing the control software running on the main processor in the radio base stations.
- The control software has the overall control and supervision responsibility for all HW in the radio base station
- The system was implemented in ordinary C running on top of the RTOS OSE Delta
- All systemisation and design was made document based in the "old fashioned" way

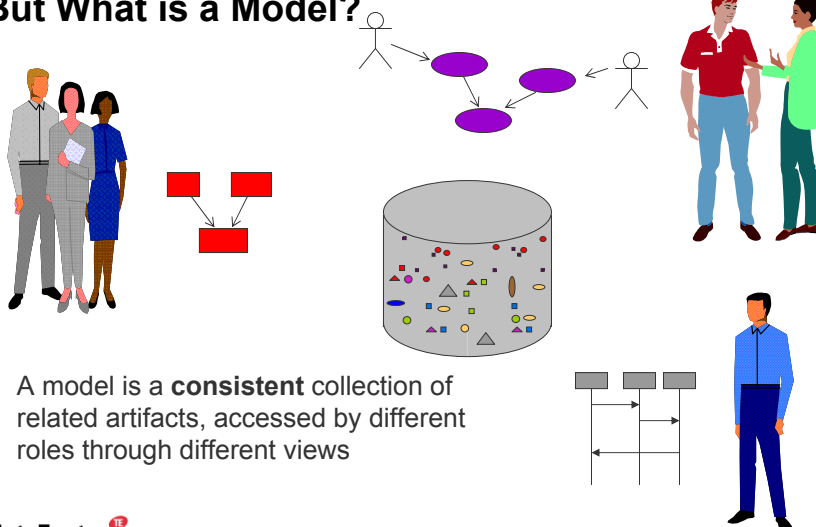
**TietoEnator**<sup>TE</sup>  
Building the Information Society

## The Transition



- During 1998 the development of commercial 3G WCDMA/UMTS systems took off
- We then felt that we wanted to improve our ways of working to:
  - improve our efficiency and time to market
  - improve the quality of our work
  - find a more “state-of-the-art” way of working
  - get a more homogenous development environment
  - basically make it more fun to work and not getting stuck in hacking C code in Emacs...
- We started to look into and evaluate different tools for model based development

## But What is a Model?



A model is a **consistent** collection of related artifacts, accessed by different roles through different views

## Document Based vs. Model Based Development

- What is Document Based Development?
  - System and design information are kept in documents in an informal way readable only by humans, not machines
  - Information often needs to be duplicated and manually updated and reviewed to be kept consistent between different documents
- What is Model Based Development?
  - The biggest difference is that in a model you put in your information only once – one source of information
  - Consistency between different views are kept automatically
  - Information is stored in a formal way which makes it possible to verify automatically to a larger extent, e.g. in a Daily Build process
  - Verifications of the model can be made earlier in the development cycle to avoid a “big bang” integration at the end of it

## Our Model Based Development Environment

- We are using Rational RoseRT which is based on the UML-RT profile
- Two major concepts are used in UML-RT:
  - Specifying the structure of the system graphically using capsules, ports and bindings
  - Specifying the behaviour of the system using graphical state machines
- All information, including low level behaviour on state transitions, are stored in the model
- Complete code generation from the model with “one push of a button” to get the final executable which gets loaded into the radio base station

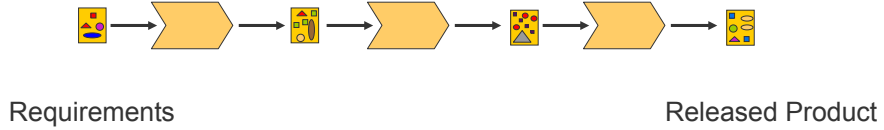
## So where is the Complexity and the Obstacles then?

- Yes, building multi-threaded real-time systems are hard
- Yes, building large embedded systems are even more complex
- Using model based development can reduce some complexity, by for example allowing the behaviour to be specified graphically
- But the real problems are found in these areas instead:
  - How do you get an architecture and organisation of your system so that you can get a lot of people working in parallel?
  - How do you keep the coordination between people on different sites and in different teams to be as limited as possible?
  - How can you work in different increments in parallel and minimize branching in your configuration management system?

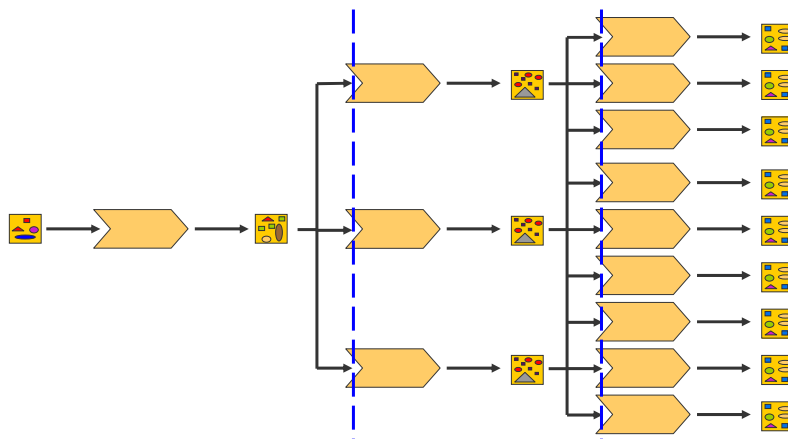
## A Large Software System

- Developed on 3 sites in 2 countries: Umeå and Kista in Sweden and Enschede in Holland
- The organisation has in total around 180 people
- Around 80 people working with design and implementation of software
- Around 40 people working with integration & verification of software
- Around 5000 model elements (700 capsules, 700 protocols and 3600 classes) in our model
- In total over 2 million lines including generated code from code generators, out of which around 300 000 lines of code are written by “hand”

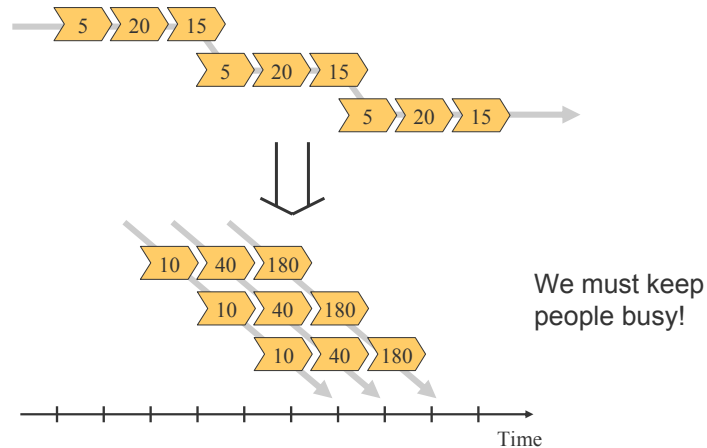
## The Basic Development Cycle



## Scaling Up a Large Development Project



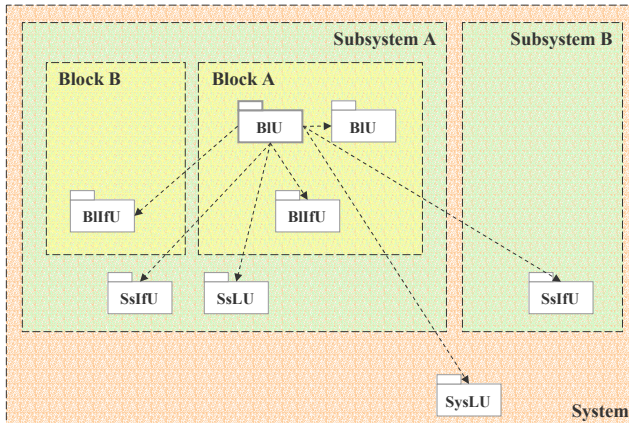
## Small vs. Large Scale Incremental Development



## Dependency Management

- One of the key aspects to be successful developing a large system
- Keeping track of your dependencies
  - Partition your system into highly cohesive part, e.g. separating interfaces of a component and the actual implementation of it
  - Setting up rules on what parts are allowed to depending on each other, and then verify continuously that you follow the rules
- Improves possibilities to:
  - Reuse your components
  - Appoint responsibility in the organisation, especially if you are distributed on multiple sites
  - Knowing who is doing what and also knowing who is depending on a certain change

## Example of Dependency Management



- BIU = Block Implementation
- BIIFU = Block Interface
- SsIFU = Sub-system Interface
- SsLU = Sub-system Library
- SysLU = System Library

## But What About Analysing Real-Time Characteristics Then?

- We have made some initial trials with RMA, Rate Monotonic Analysis where we had a Master Thesis has been performed
- The result and findings:
  - Scaling up in large models and organisation becomes hard
  - The cost compared to the benefit is hard to justify
  - Complete process from early systemisation through design and implementation must be considered which can be hard in a large organisation.
  - Big risk of ending up with a model of the model which quickly will become inconsistent
- But if the usage of RMA failed how do we know if our system is schedulable? Do we know if it meet all its dead lines?

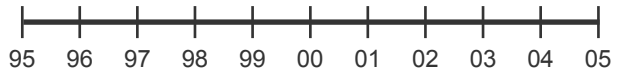
## Keeping Things Simple

- The part of the system we are building is a soft real-time system
- No hard dead lines that needs to be met
- We use a simple capsule thread mapping to meet our response times:
  - Two threads are used: One “fast” and one “slow”
  - Thread priorities: Few priorities used, basically just two
  - Signal priorities: No signal priorities used in RoseRT
- Basically we try to keep things as simple as possible to make the design of the system as easy as possible:
  - No sharing of data and usage of semaphores, only strict ownership of data
  - Simple layered architecture and clear dependency chains to reduce risk of dead lock situations

## Some Thoughts on Formal Methods and Tools

- Yes, the ideas in the area of formal analysis of the characteristics of a real-time system is great
- But to be able to be successful there are things that must be considered, both by tool vendors and academic research:
  - The methods and tools must be able to be scaled up
    - Tools must be able to handle very large models
    - Methods and tools must consider which artifacts are updated when. Just consider that different parts of the model needs to be updated, reviewed and frozen at different milestones in the project.
  - The methods and tools must consider that you may be forced in developing the system in parallel tracks
- If all this is not fulfilled the deployment in large projects and systems will utterly fail!

## Our Future



- Today we are focusing our model based development in the later design and implementation phases
- We believe in model based development and its increase in productivity and quality so now we are aiming for having model based systemisation where we want to keep our system information in a consistent model as well
- In this way we can get consistent information, e.g. with respect to signal names, all the way from early specification and systemisation, via design and implementation all the way to code generation and the final executing system
- The upcoming UML 2.0 also includes lots of improvements in the area of modelling complex embedded systems, e.g. drawing sequence diagrams. The concepts from with capsules, port and bindings UML-RT are making its way formally into the UML standard.

## Finally some words of wisdom: How to succeed with model based development

- Get commitment in the organization: From line managers, project managers, key SW architects, system engineers and designers.
- A true dedicated Methods & Tools team, with the responsibility of:
  - Tool support
  - Daily Build of the model
- Do not believe that using a new tool or a new method will be the “silver bullet” that solves all your problems
  - New problems and aspect will arise, e.g. how to perform reviews
  - You still need to get a good architecture and do your intellectual engineering work. The tool will not solve that for you...