DAIMLERCHRYSLER

PROACTIVE INFRASTRUCTURE

„Modellierung operationaler Aspekte von Systemarchitekturen“

Master Thesis presentation

October 2005 – March 2006

Mirko Bleyh - Medieninformatik

Agenda

- Goals
- Model-Driven Software Development
- Pro-active Infrastructure (PAI)
- Operational Aspects
- PAI Operational Model
- DSL for the Operational Model
- Model Transformations
- Implementation with Eclipse Tools
- Demo
- Conclusion
- References

Goals

- Analyse modeling approaches for operational aspects
- Evaluate existing technology for domain-specific modeling
- Implement prototype modeling solution

DaimlerChrysler

Agenda

- Goals
- Model-Driven Software Development
- Pro-active Infrastructure (PAI)
- Operational Aspects
- PAI Operational Model
- DSL for the Operational Model
- Model Transformations
- Implementation with Eclipse Tools
- Demo
- Conclusion
- References

DaimlerChrysler

Model-Driven Software Development

Goals:

- Reduce software development time
- Reduce software complexity
- Increase software quality
- Increase software reusability

Key aspects:

- Use models as primary development artifacts based on DSL
- Transform abstract models into less abstract models (or source code)
- Provide Infrastructure (tools, processes, components)

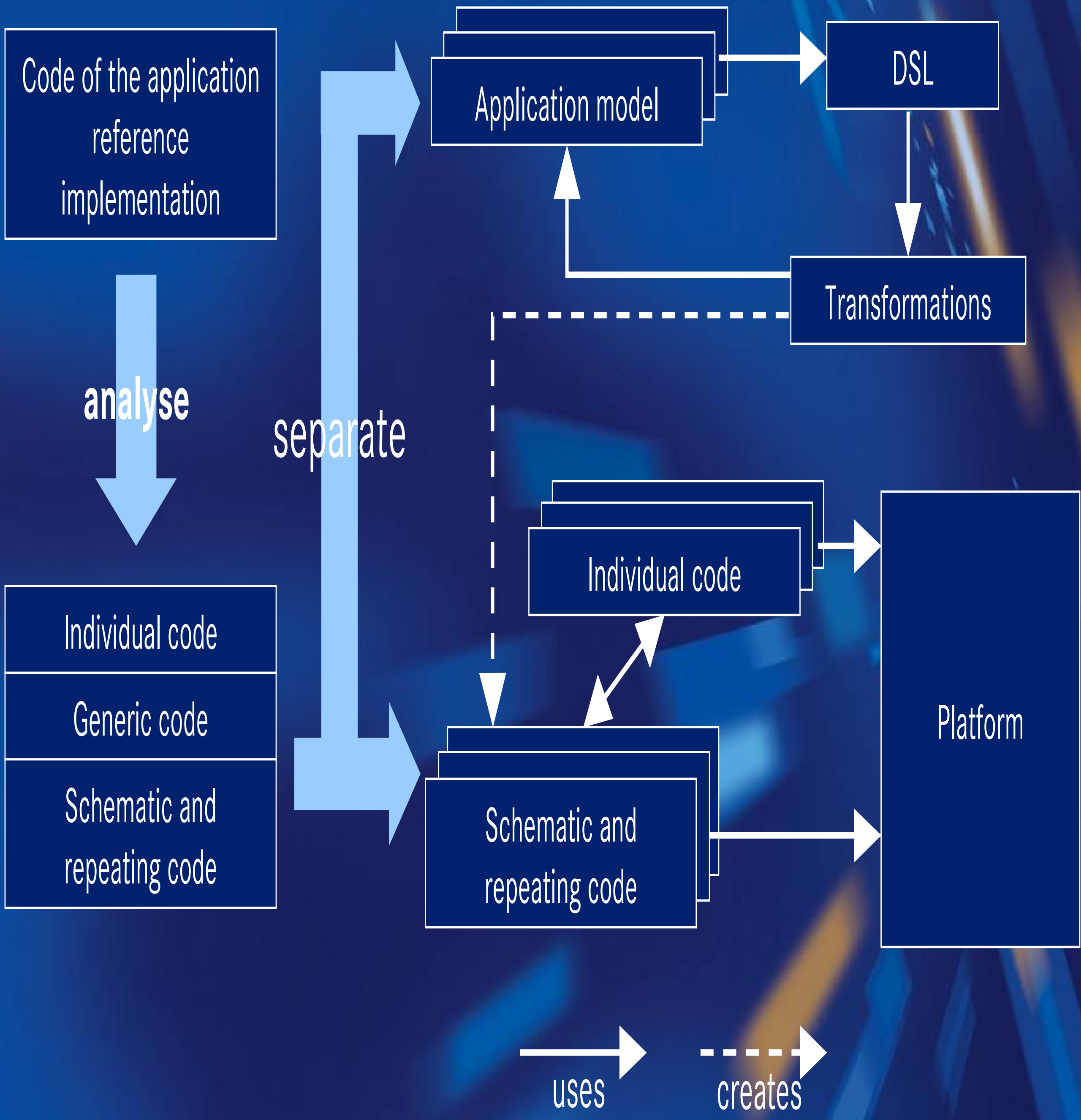
Model-Driven Software Development

Main paradigms:

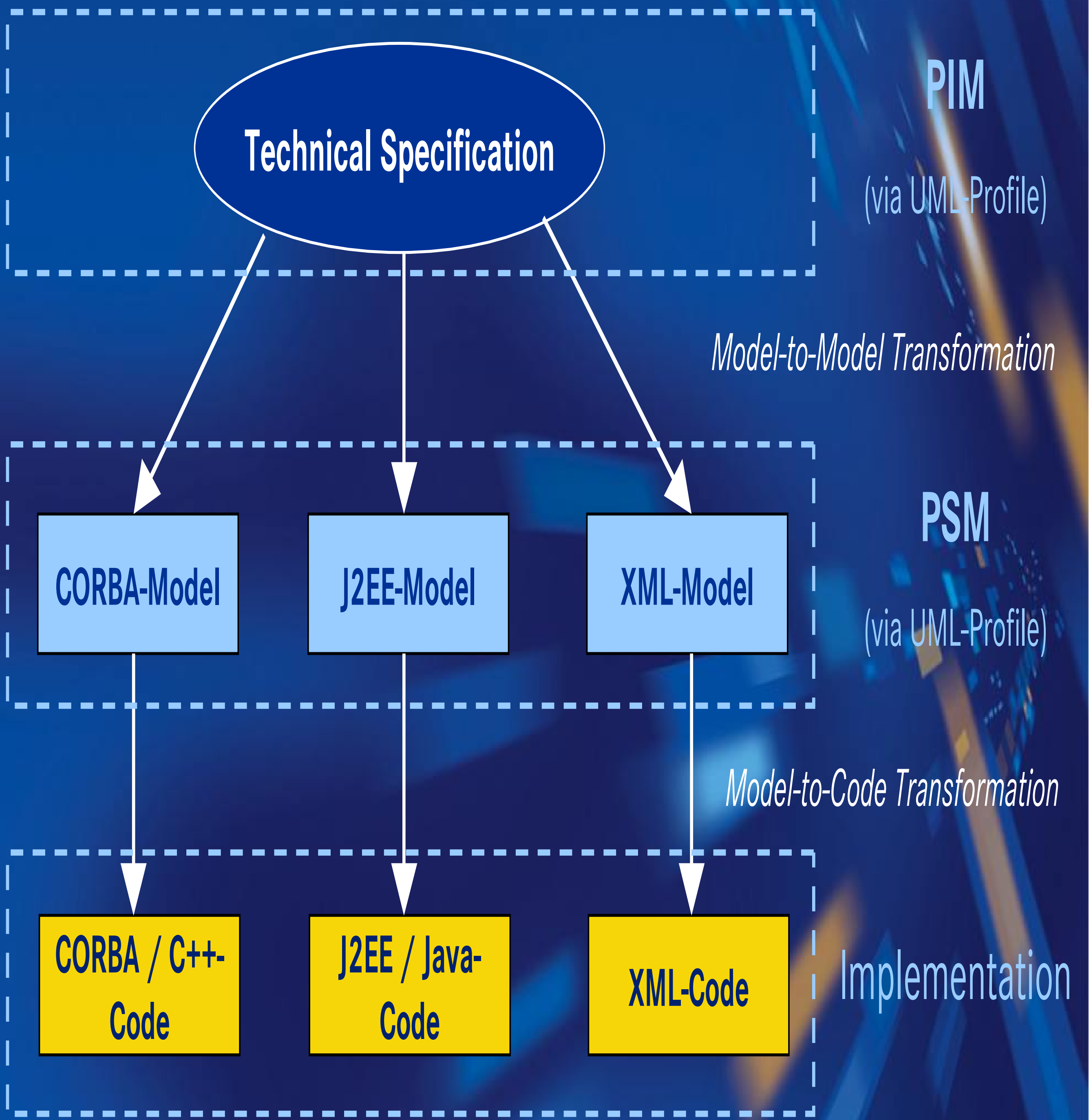
- Model-Driven Software Development
  - Use abstract but formal models based on DSLs
  - Use transformations to generate less abstract models or code
- Model-Driven Architecture (MDA)
  - Standardization of Model-Driven Software Development by OMG
  - Usage of OMG standards (MOF, UML, XMI, OCL, QVT)
  - Focus on interoperability and portability
- Software Factories
  - Microsofts vision of Model-Driven Software Development
  - Rejects OMG standards, uses own DSL Metamodel
  - Focus on tooling support

DaimlerChrysler

Model-Driven Software Development



Model-Driven Architecture



Agenda

- Goals
- Model-Driven Software Development
- Pro-active Infrastructure (PAI)
- Operational Aspects
- PAI Operational Model
- DSL for the Operational Model
- Model Transformations
- Implementation with Eclipse Tools
- Demo
- Conclusion
- References

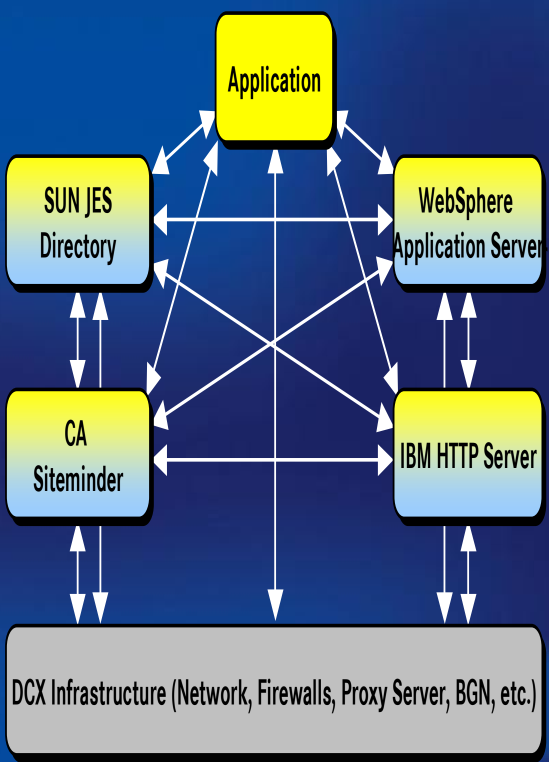
Pro-active Infrastructure

"Pro-active Infrastructure is a DCX standardized IT infrastructure foundation to optimize the development and in particular the operations of custom applications within the DaimlerChrysler group."

DaimlerChrysler

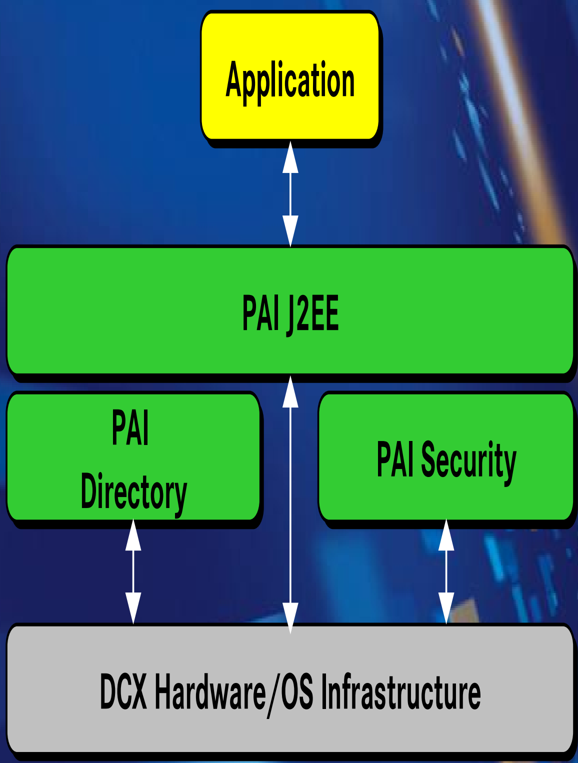
Pro-active Infrastructure

Before the usage of PAI



Infrastructure & Middleware integration issues need to be addressed on an application project level

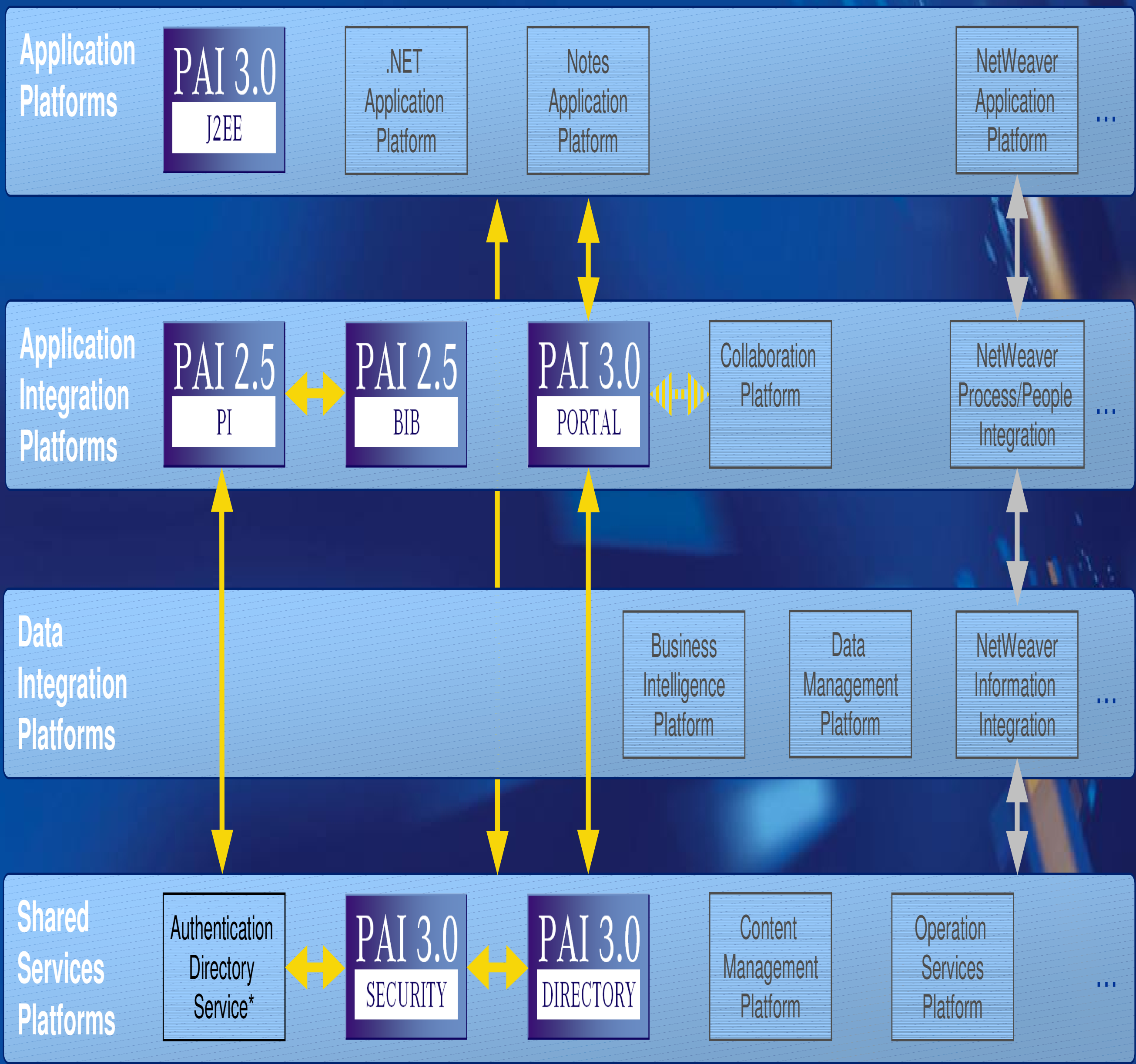
After the usage of PAI



Standardized, Integrated & Release Managed Platforms for all application projects to minimize complexity and provide standardized solutions.

DaimlerChrysler

Pro-active Infrastructure



*provided by PAI Security and Directory Providers

DaimlerChrysler

Agenda

- Goals
- Model-Driven Software Development
- **Pro-active Infrastructure (PAI)**
- Operational Aspects
- PAI Operational Model
- DSL for the Operational Model
- Model Transformations
- Implementation with Eclipse Tools
- Demo
- Conclusion
- References

Mirko Bleyh - Medieninformatik

DaimlerChrysler

Operational Aspects

Software Architecture is devided into two categories:

Functional Aspects

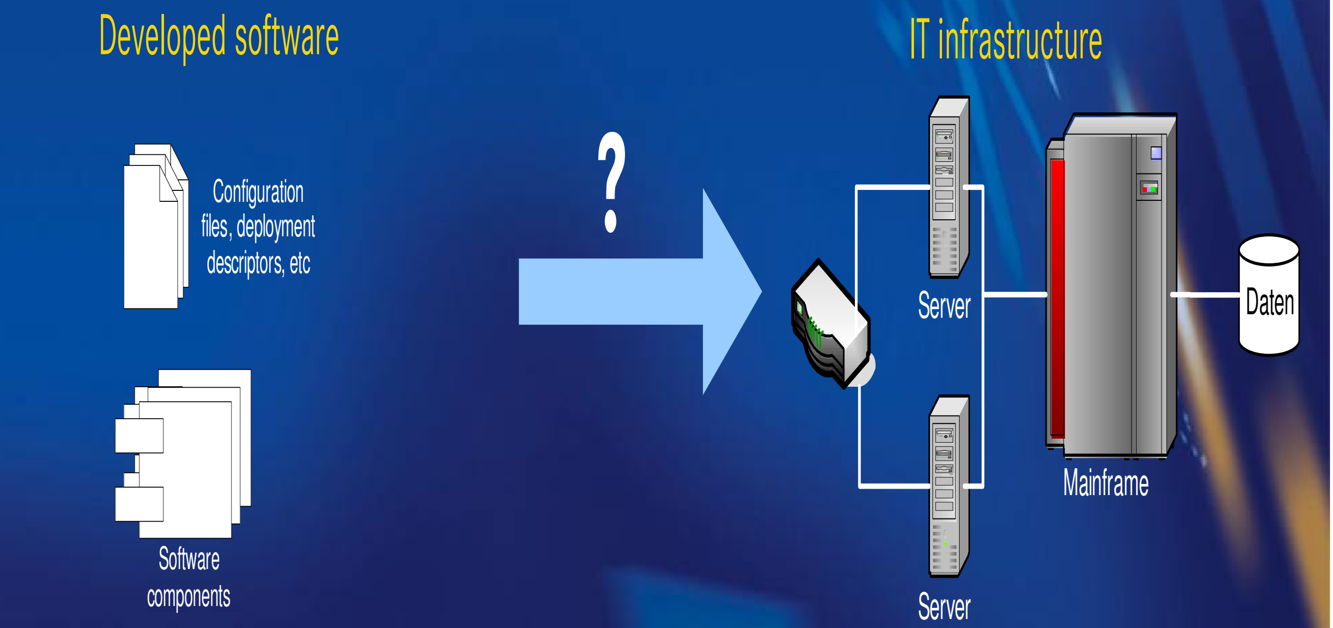
- Structures of software components
- Interaction between components
- Definition of interfaces
- Dynamic behaviour of components

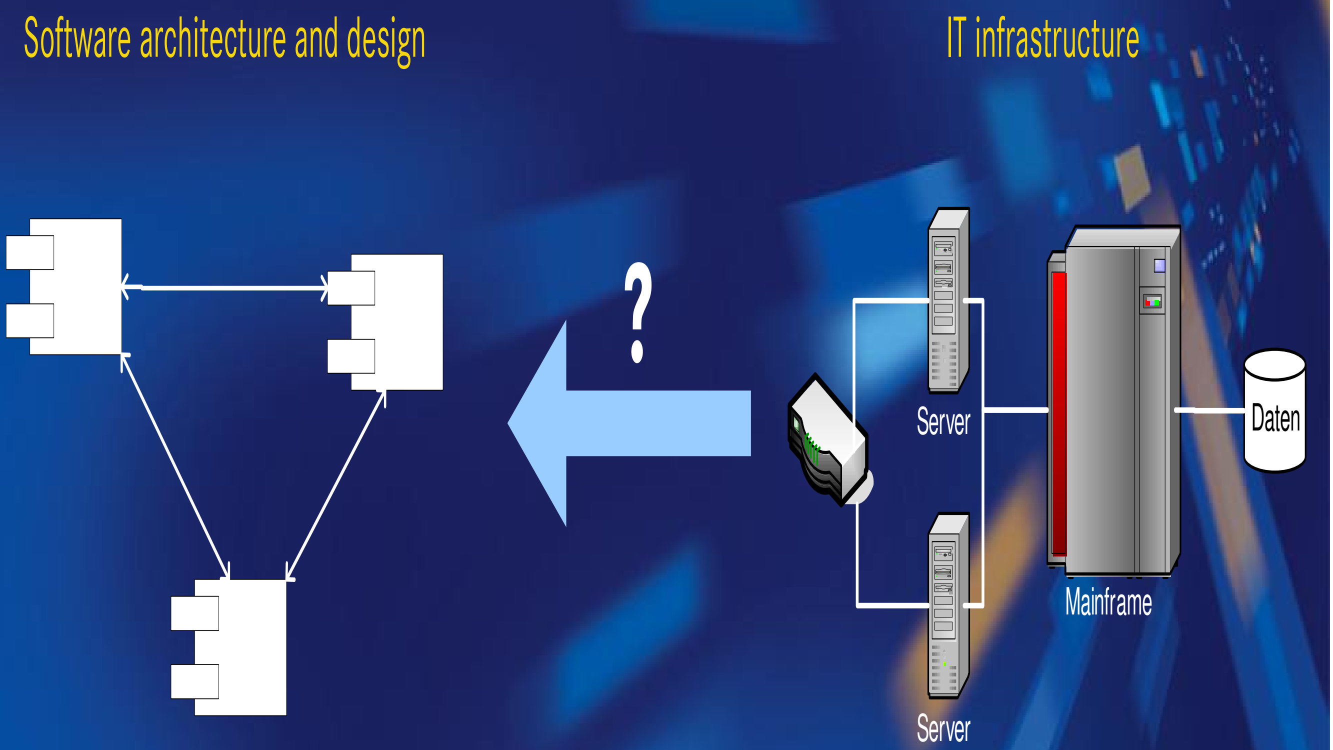
Operational Aspects

- Network organisation
- Distribution of components
- Service level requirements
- Systems Management

IBM Architecture Description Standard defines conventions for notation, terminology and semantics for the architecture of an IT system

Operational Aspects





Operational Aspects

- Vital for the developement and operation of large scale applications
- Need to be addressed in all phases of software development:

Requirements:
- Operational requirements
- Non-functional requirements

Design:
- Distribution of components
- Design for non-functional requirements

Implementation:
- Implement for non-functional requirements
- Implement operational Features

Test:
- Set up test environment
- Test functional and non-functional Requirements

Operation:
- Deploy and configure software
- Maintain and manage software

DAIMLERCHRYSLER

Agenda

- Goals
- Model-Driven Software Development
- Pro-active Infrastructure (PAI)
- **Operational Aspects**
- PAI Operational Model
- DSL for the Operational Model
- Model Transformations
- Implementation with Eclipse Tools
- Demo
- Conclusion
- References

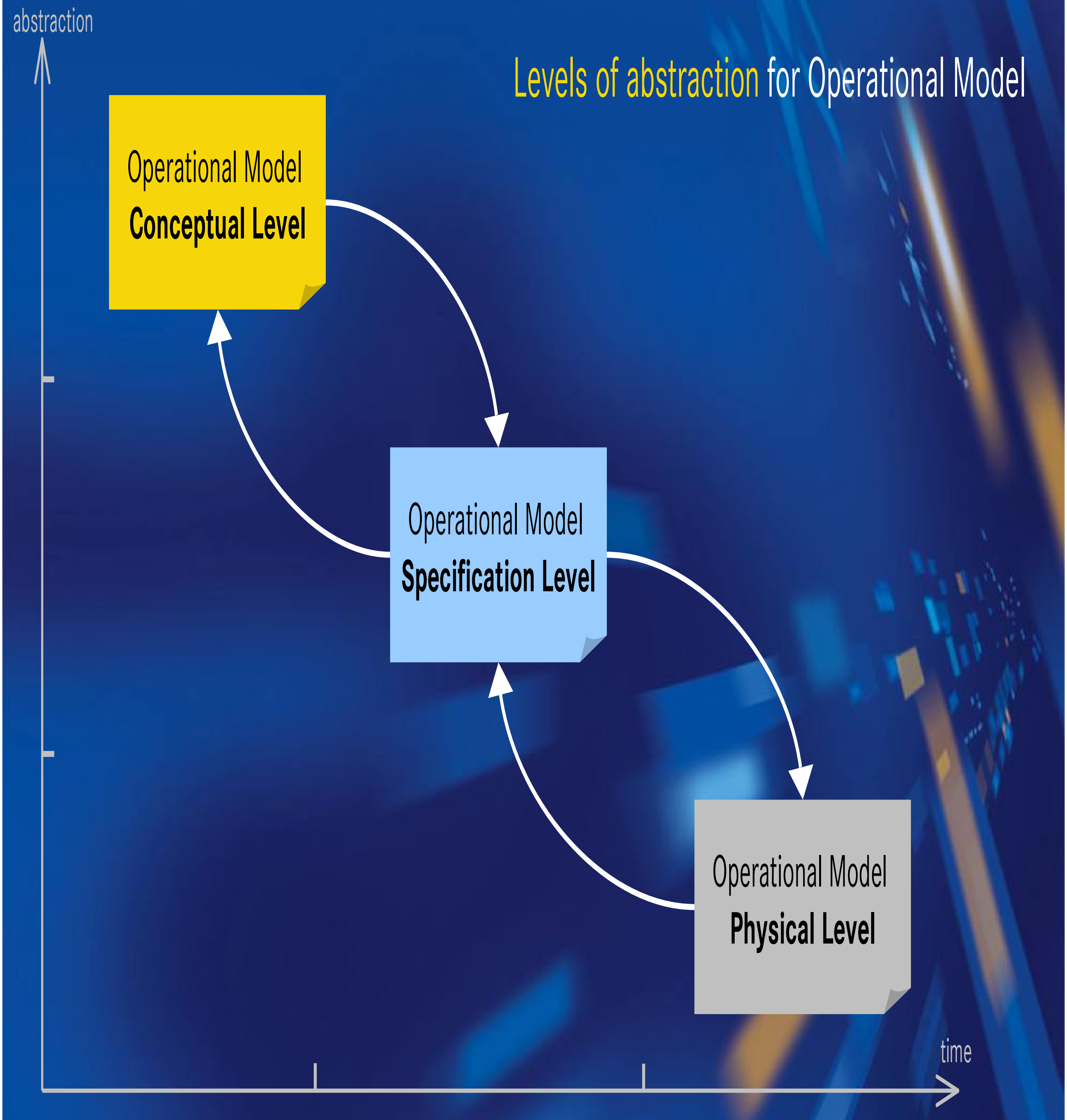
DAIMLERCHRYSLER

PAI Operational Model

Operational Model (OM) used for operational aspects within PAI:

- Part of IBM Global Services Method
- Defines:
  - **Distribution of components** over
  - **Nodes of the IT infrastructure** and the
  - **Connections required for the interactions of the components** in order to archieve
  - **Functional and non-funtional requirements**
- Contains:
  - One or more relationship diagrams
  - One or more walktrough diagrams
  - Detailed description of nodes and connections
  - Description of how functional and non-functional requirements will be met
  - Description of the systems management strategy
- Devided into two / three different levels of abstraction...

PAI Operational Model



PAI Operational Model

Operational Model
Conceptual Level

Operational Model Conceptual Level (CL)

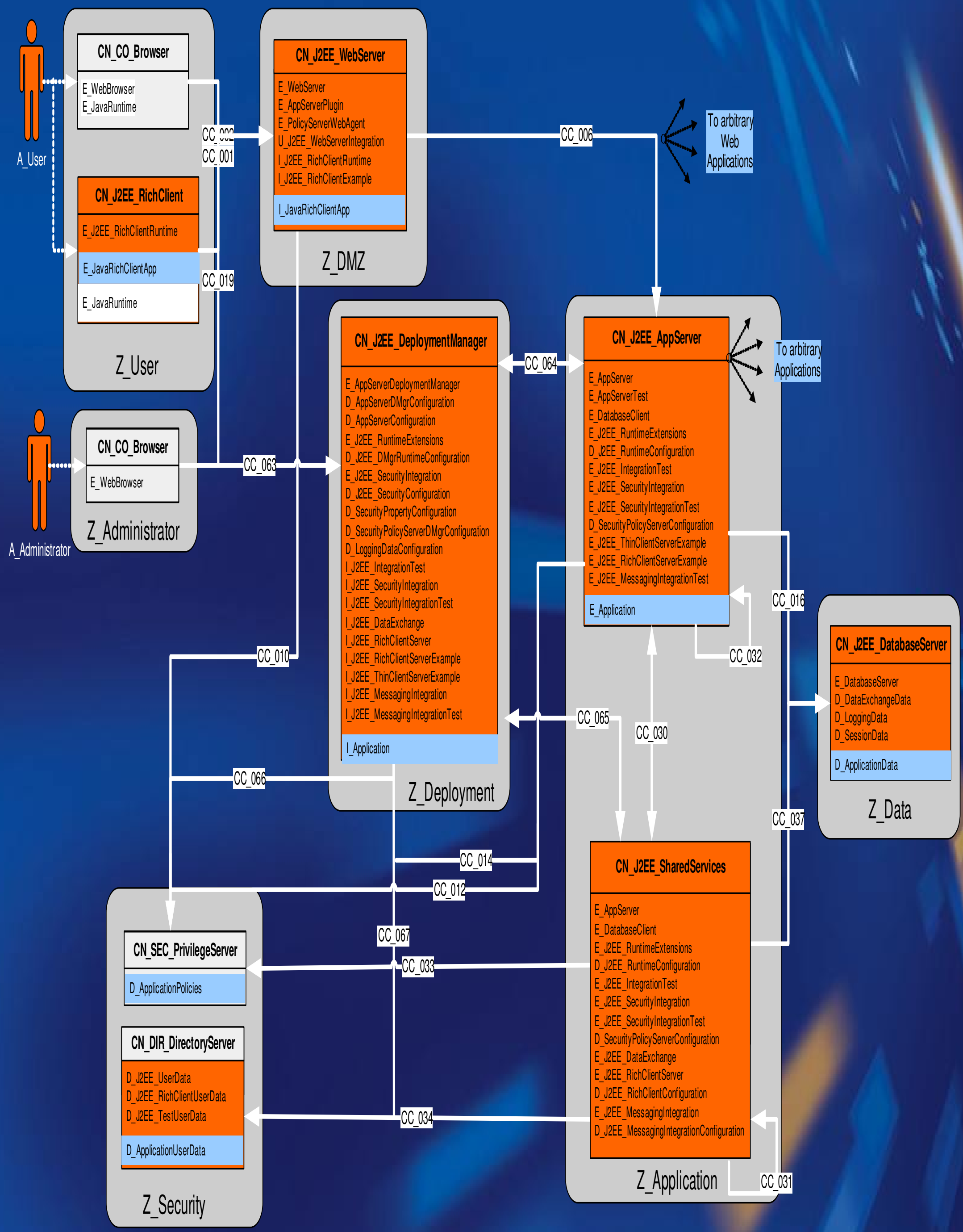
- defines the set of conceptual nodes (CN) and functional relations between them
- specifies the zoning
- defines deployment units (DU) for each CN
- no product information, no physical specifications

Operational Model
Specification Level

Operational Model
Physical Level

DaimlerChrysler

PAI Operational Model



DaimlerChrysler

PAI Operational Model

Operational Model
Conceptual Level

Operational Model
Specification Level

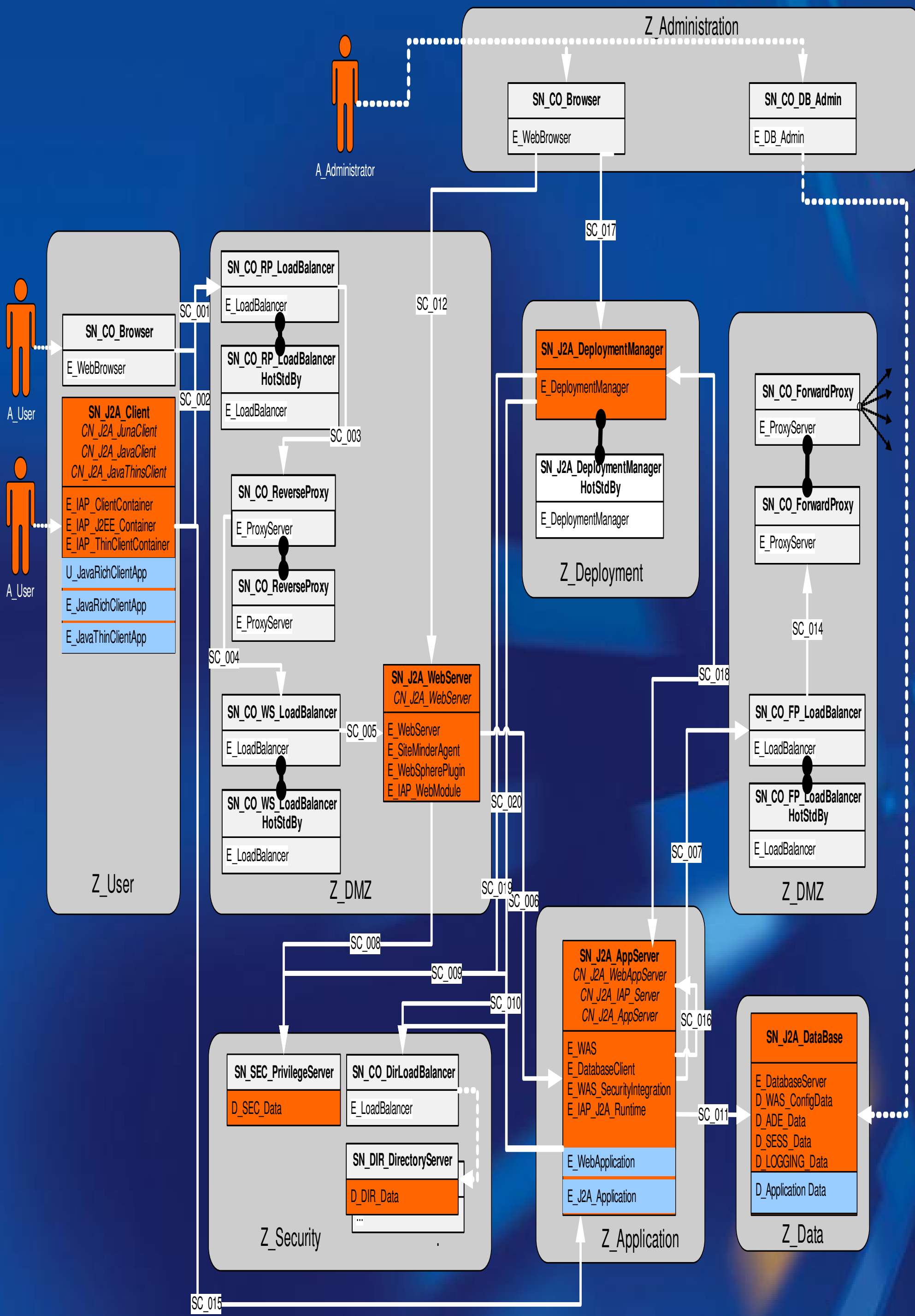
Operational Model
Physical Level

Operational Model Specification Level (SL)

- Specific instance of conceptual level
- Defines the products and major versions to use
- Specifies the type of CN's (cluster, HW pattern, ..)
- No hostnames, no information about real instances

DaimlerChrysler

PAI Operational Model



DAIMLERCHRYSLER

PAI Operational Model

Operational Model
Conceptual Level

Operational Model
Specification Level

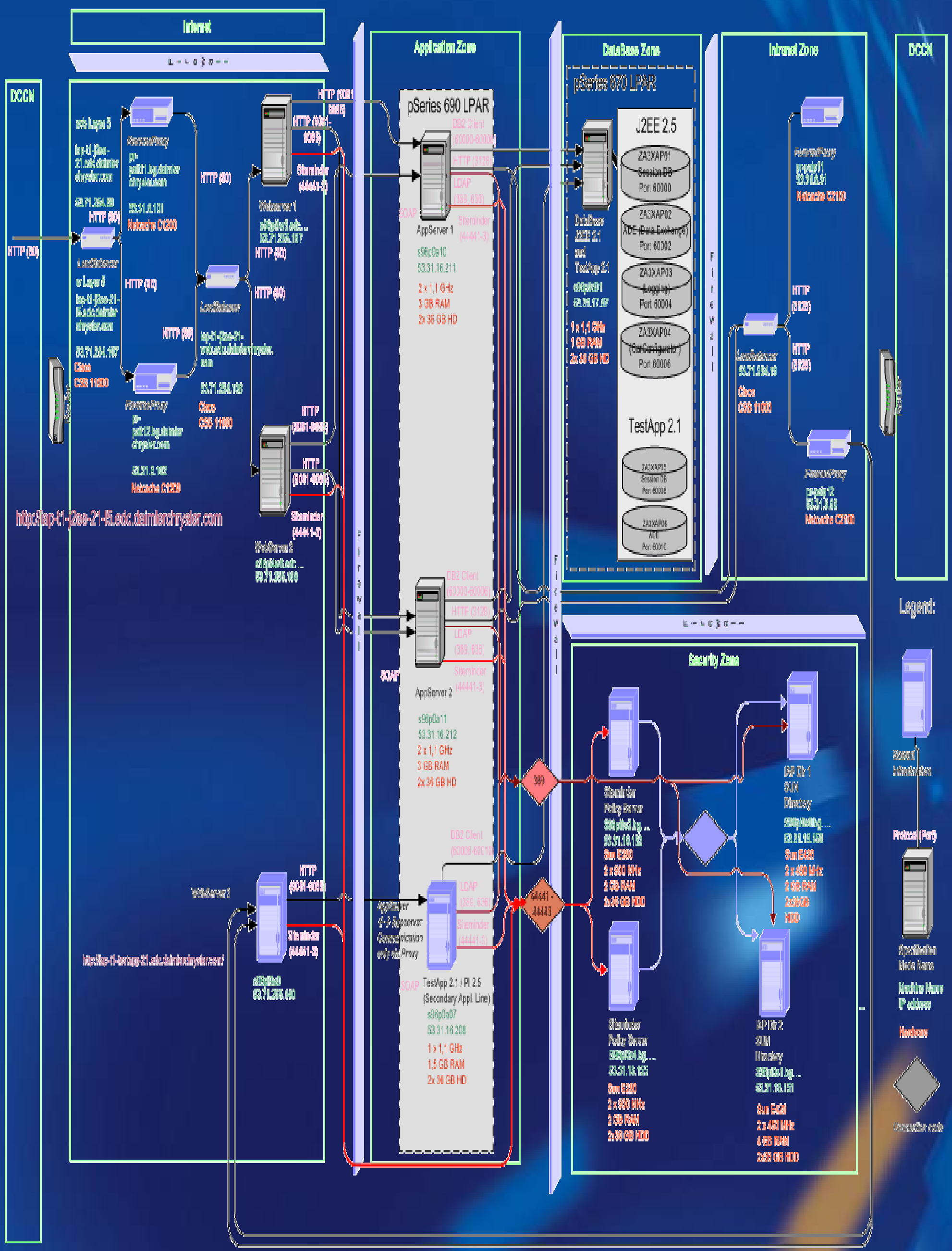
Operational Model
Physical Level

Operational Model Physical Level (PL)

- Defines all aspects required for setting up the environment in a real hosting environment
- IP's, hostnames, ports, FW
- Machine specification, references to asset management

DaimlerChrysler

PAI Operational Model



PAI Operational Model

Other operational artifacts:

- Operational Description (OD):
  - XML-based document
  - Central repository for operational information
  - Contains data from Operational Model of all levels
  - Detailed configuration parameters for base products and PAI components
  - Used for PICS
- Platform Installation and Configuration Solution (PICS)
  - Automated installation and configuration of PAI J2EE Platform
  - Based on predefined solutions and user-guided wizard
  - Uses OD as major input

PAI Operational Model

Current state:

- Operational Model only used in informal way (Visio diagrams, Word files...)
- No consistency checks possible
- No standard notation so far for Operational Model
- Only rarely used by PAI projects (due to lack of tools?)

- Operational Description for J2EE has around 3000 lines of XML
- Complex and not human readable
- Difficult to maintain

→ Modeling approach could solve some problems here!

DaimlerChrysler

Agenda

- Goals
- Model-Driven Software Development
- Pro-active Infrastructure (PAI)
- Operational Aspects
- **PAI Operational Model**
- DSL for the Operational Model
- Model Transformations
- Implementation with Eclipse Tools
- Demo
- Conclusion
- References

DaimlerChrysler

DSL for the Operational Model

Domain Specific Languages:

- Used to define the key aspects of a specific domain
- Enriches models with semantic
- Captures the knowledge of the domain expert
- Already widely used (SQL, FORTRAN, etc.)
- Key item for model-driven software development

Ingredients:

- Abstract Syntax
- Static Semantic
- Dynamic Semantic
- Concrete Syntax

Abstract Syntax, Static Semantic → Metamodel

Dynamic Semantic → Model transformations

DSL for the Operational Model

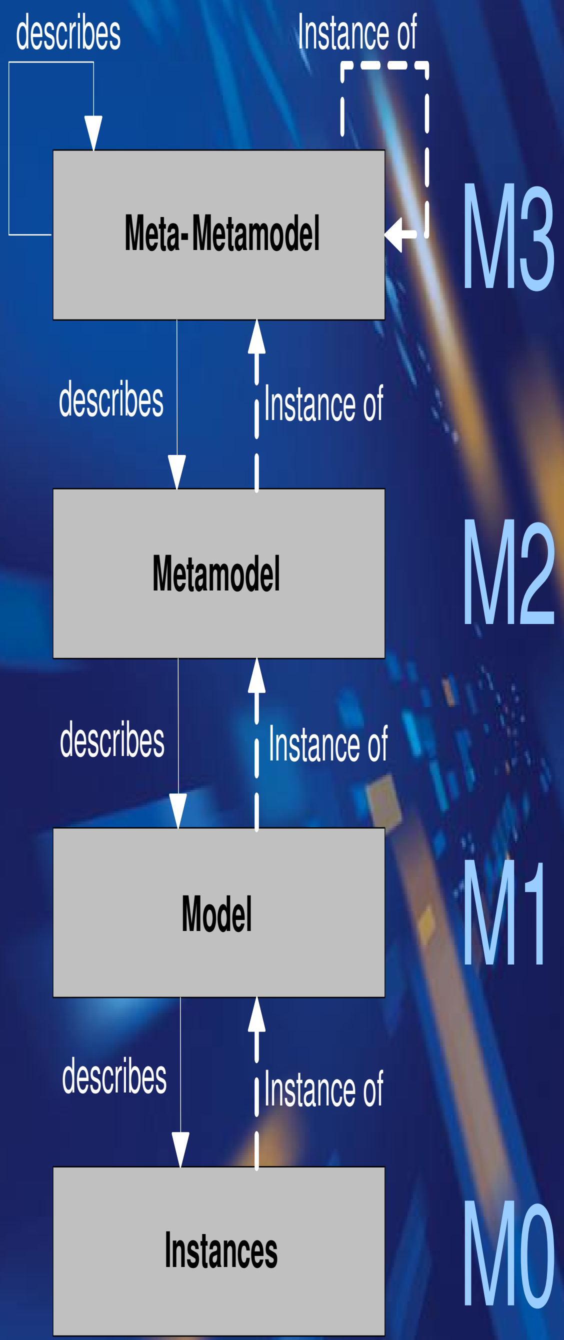
Metamodeling

- 4 layers defined by Meta-object Facility (MOF)
- Java Code on M0 (as instance of UML-model)
- UML-Models on M1
- UML-Metamodel on M2
- MOF on M3

Possible metamodels for DSL:

1. Extend UML-Metamodel in M2 with profiles (stereotypes, tagged values)
2. Create new M2 metamodel based on MOF
3. Create new M2 metamodel based on other M3 metametamodel

→ DSL as new M2 metamodel based on MOF



DSL for the Operational Model

Metamodeling approach:

- Analyse existing models
- Extract key elements
- Model key elements in metamodel
- Use modeling techniques known from UML modeling

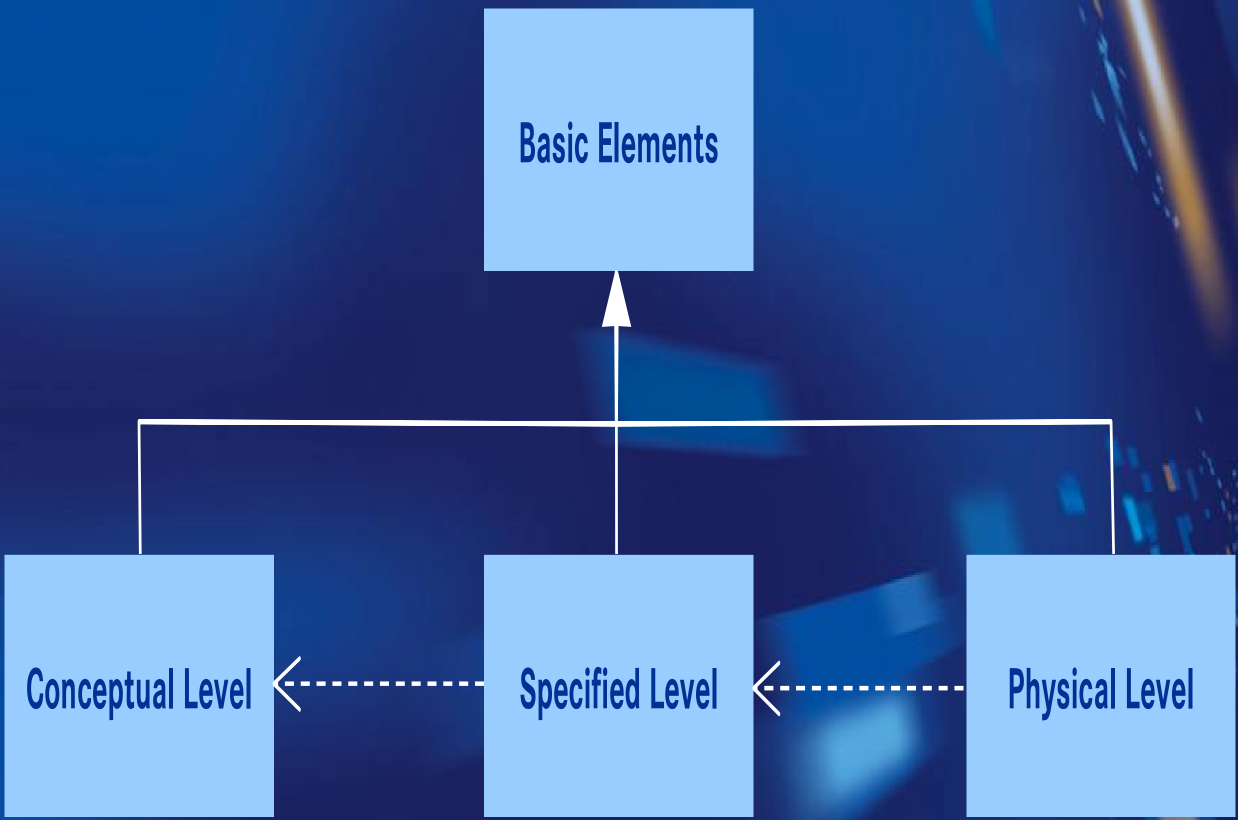
Best-practices:

- Keep it simple
- Interatively check and extend metamodel against model
- Model containment in one single element (direct or indirect)

DAIMLERCHRYSLER

DSL for the Operational Model

Structure of the Metamodel:

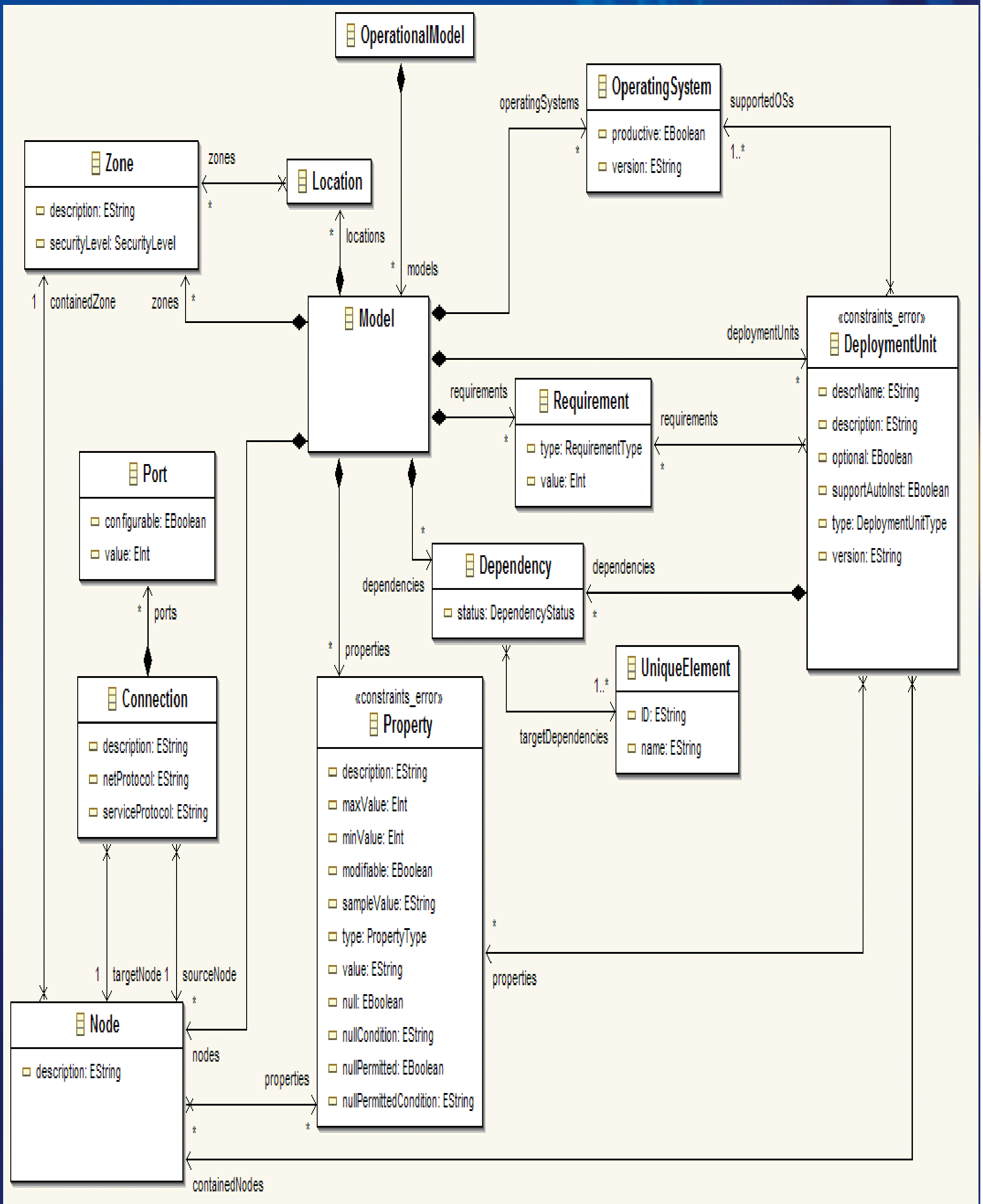


DAIMLERCHRYSLER

DSL for the Operational Model

Metamodel

Basic elements

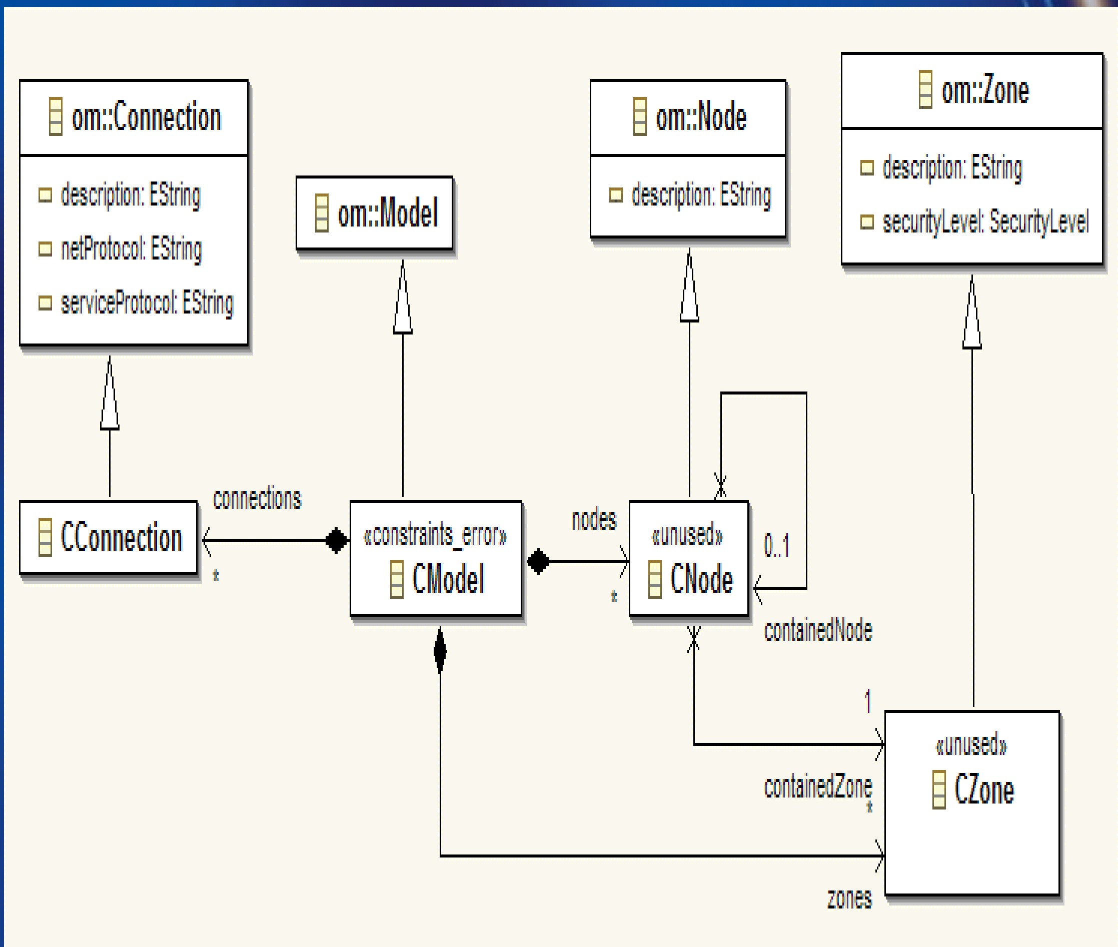


DaimlerChrysler

DSL for the Operational Model

Metamodel

Elements of conceptual level

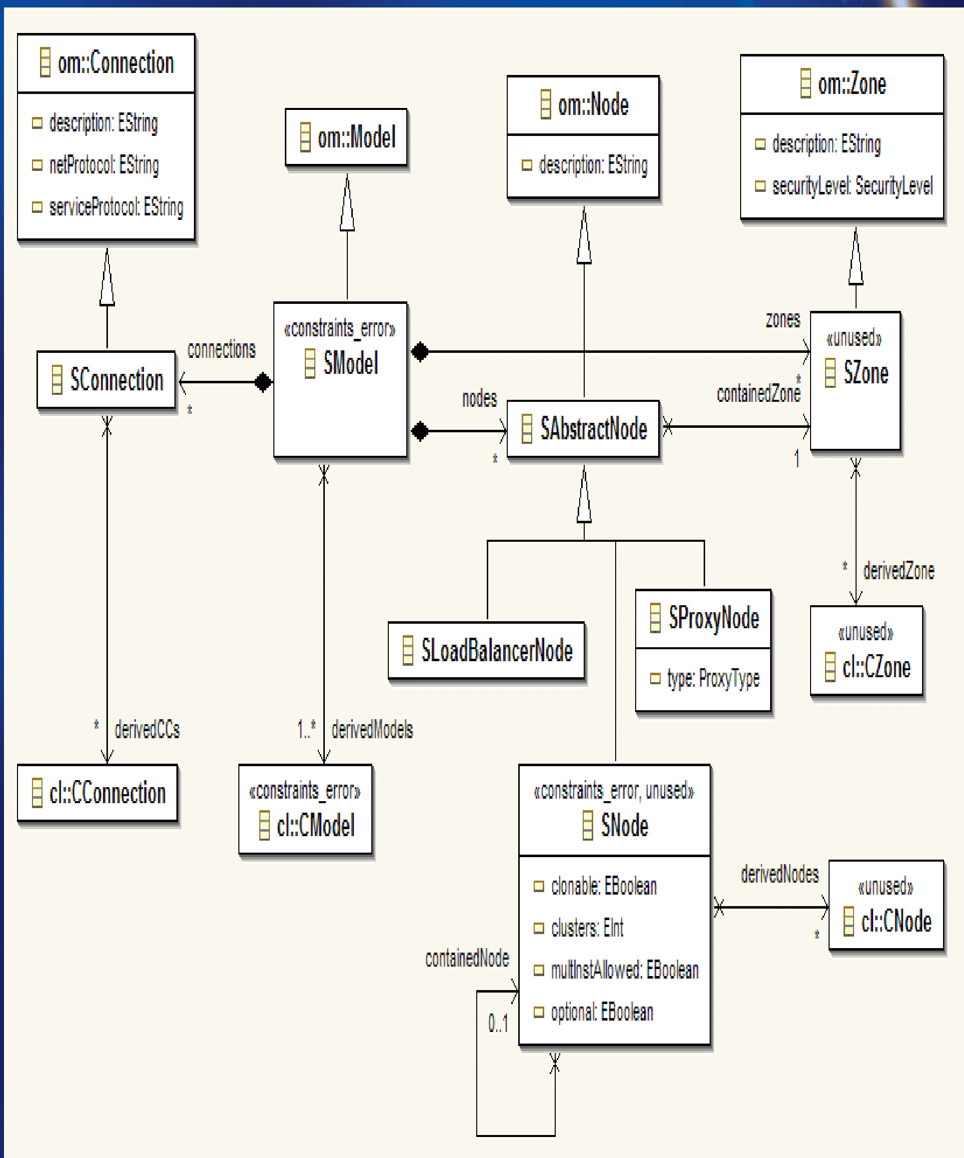


DaimlerChrysler

DSL for the Operational Model

Metamodel

Elements of specification level

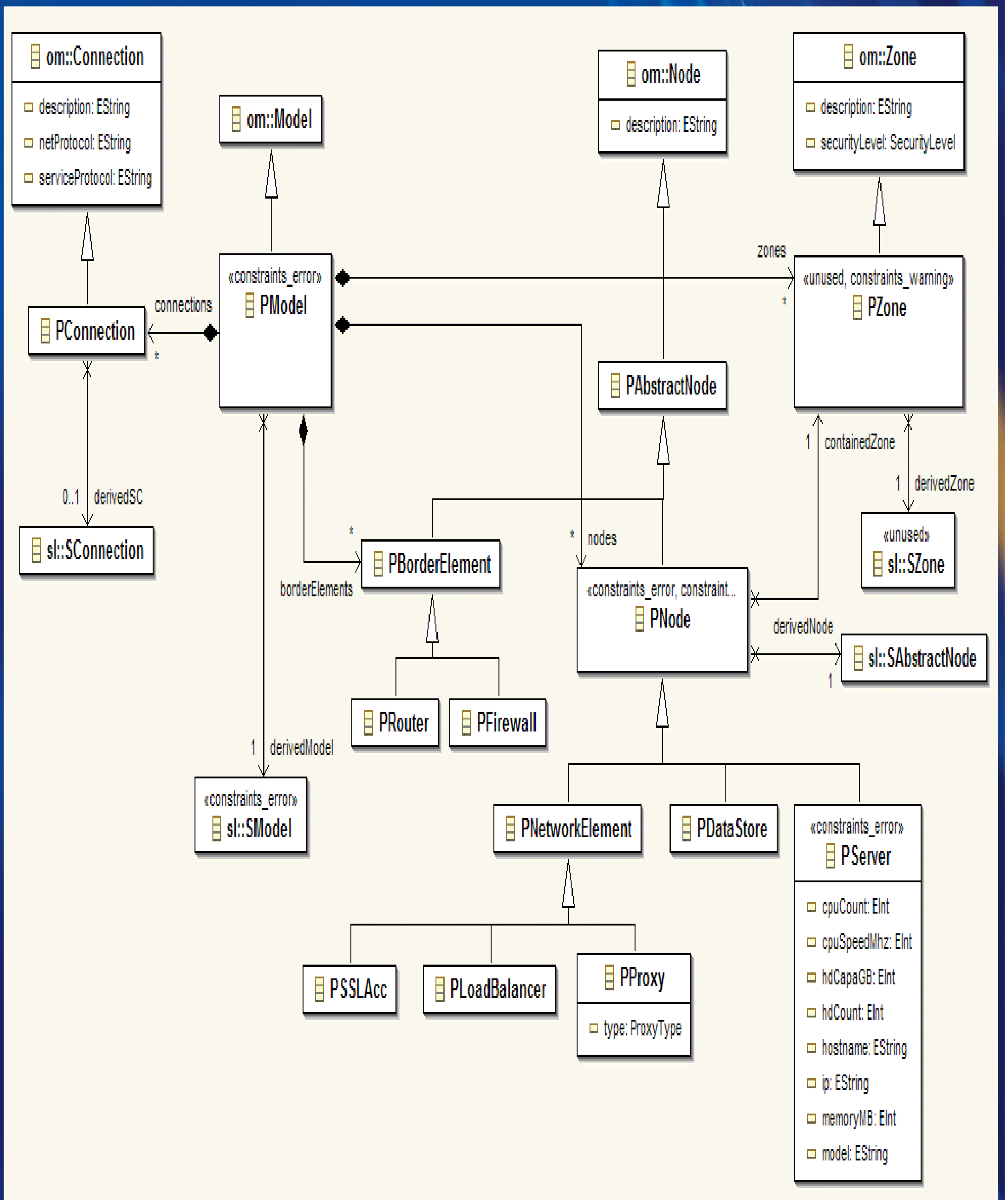


DaimlerChrysler

DSL for the Operational Model

Metamodel

Elements of physical level



DSL for the Operational Model

Semantics:

- Semantics define the meaning of a language
- Static semantics define the well-formedness of a model
- Static semantics can be expressed as contraints in the metamodel
- Dynamic semantics define the meaning of elements of the metamodel
- Dynamic semantics expressed on forms of transformations

DaimlerChrysler

DSL for the Operational Model

Static Semantic with OCL

- Object Constraint Language (OCL) is a declarative, side-effect free language for the definition of constraints on a model (or metamodel)
- Can be applied on M1, M2 or M3

Example for the Operational Model metamodel:

```
context sl::SNode
inv:
   self.deploymentUnits->select(du | not du.supportedOSs
      ->exists(os | os.ID = self.operatingSystem.ID))
      ->union(
         self.derivedDUs->select(du | not du.supportedOSs
            ->exists(os | os.ID = self.operatingSystem.ID))
      )
```

DaimlerChrysler

Agenda

- Goals
- Model-Driven Software Development
- Pro-active Infrastructure (PAI)
- Operational Aspects
- PAI Operational Model
- **DSL for the Operational Model**
- Model Transformations
- Implementation with Eclipse Tools
- Demo
- Conclusion
- References

Model Transformations

Why model transformations?

- Capture the semantics of the metamodel
- Reduce modeling complexity and effort
- Ensure consistency between models

Model Transformations vs. Text Transformations (XSLT)

- Validation of transformation rules based on metamodel
- Only valid models are generated
- Reduced complexity
- Support for synchronisation of models

Model Transformations

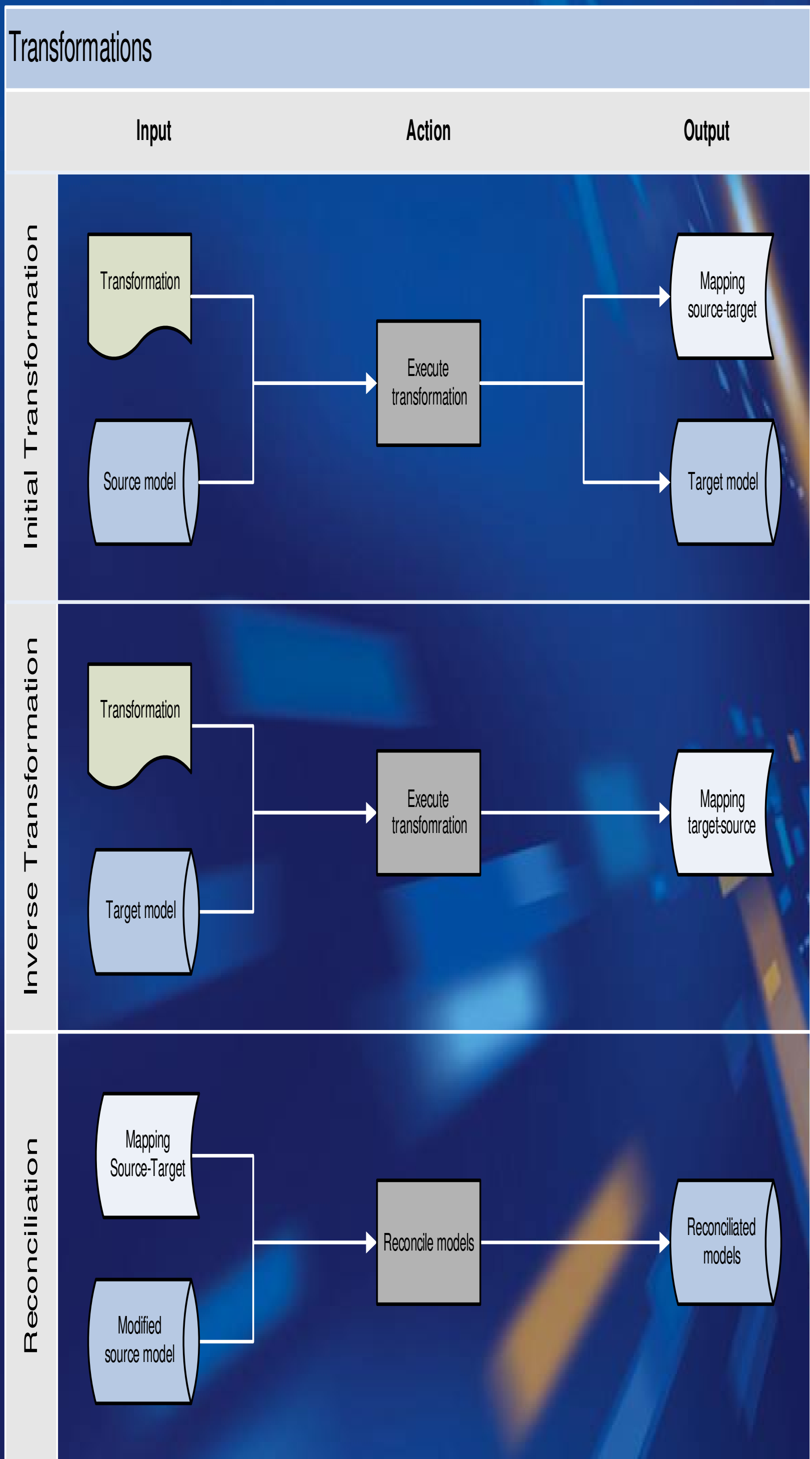
IBM Model Transformation Framework

- Based on EMF metamodels
- Bidirectional Transformations
- Reconciliation of transformed models
- Based on RFP on QVT (Query, View, Transformation)
- Available as Eclipse Plugin

DAIMLERCHRYSLER

Model Transformations

Workflow model transformations



Agenda

- Goals
- Model-Driven Software Development
- Pro-active Infrastructure (PAI)
- Operational Aspects
- PAI Operational Model
- DSL for the Operational Model
- **Model Transformations**
- Implementation with Eclipse Tools
- Demo
- Conclusion
- References

DaimlerChrysler

Implementation with Eclipse Tools

Eclipse Tooling Landscape:

- Eclipse:
  - extensible Rich-Client Framework and IDE
- Eclipse Modeling Framework (EMF):
  - modeling framework and code generation facility for building tools and other applications based on a structured data model
- Eclipse Graphical Editing Framework (GEF):
  - framework for creating rich graphical editors based on existing application model
- Eclipse Graphical Modeling Framework (GMF):
  - provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF

DAIMLERCHRYSLER

Implementation with Eclipse Tools

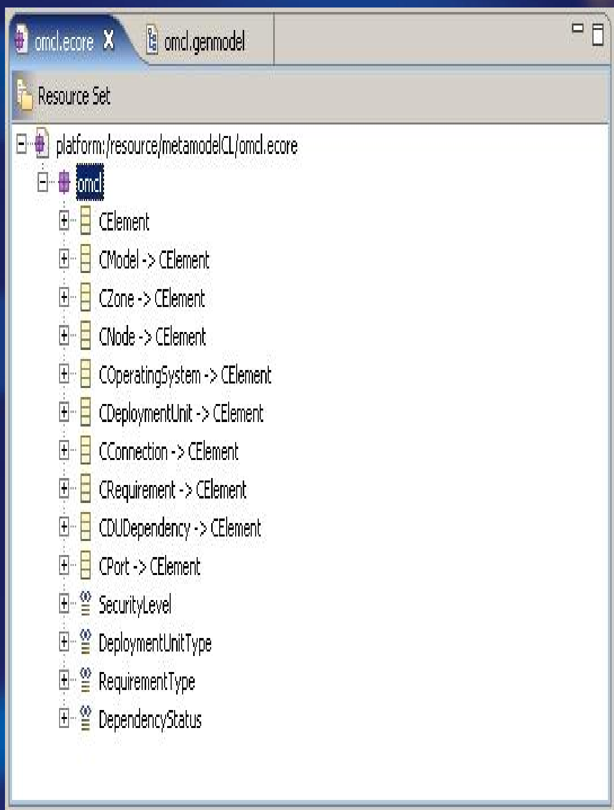
Tasks:

1. Create metamodel in EMF *(abstract syntax)*
2. Add constraints in OCL to the metamodel *(static semantic)*
3. Create GMF editor definition from metamodel (*concrete syntax)*
4. Generate metamodel and editor code
5. Adjust generated code
6. Run editor in Eclipse

Implementation with Eclipse Tools

1. Create metamodel in EMF

- EMF metametamodel is Ecore → similar to EMOF or UML class diagram
- Eclipse EMF provides simple Ecore editor
- EMF metamodel can be imported from Rational UML model, annotated Java classes, or XMI
- Graphical Editor can be used from GMF or e.g. Omondo EclipseUML



DaimlerChrysler

Implementation with Eclipse Tools

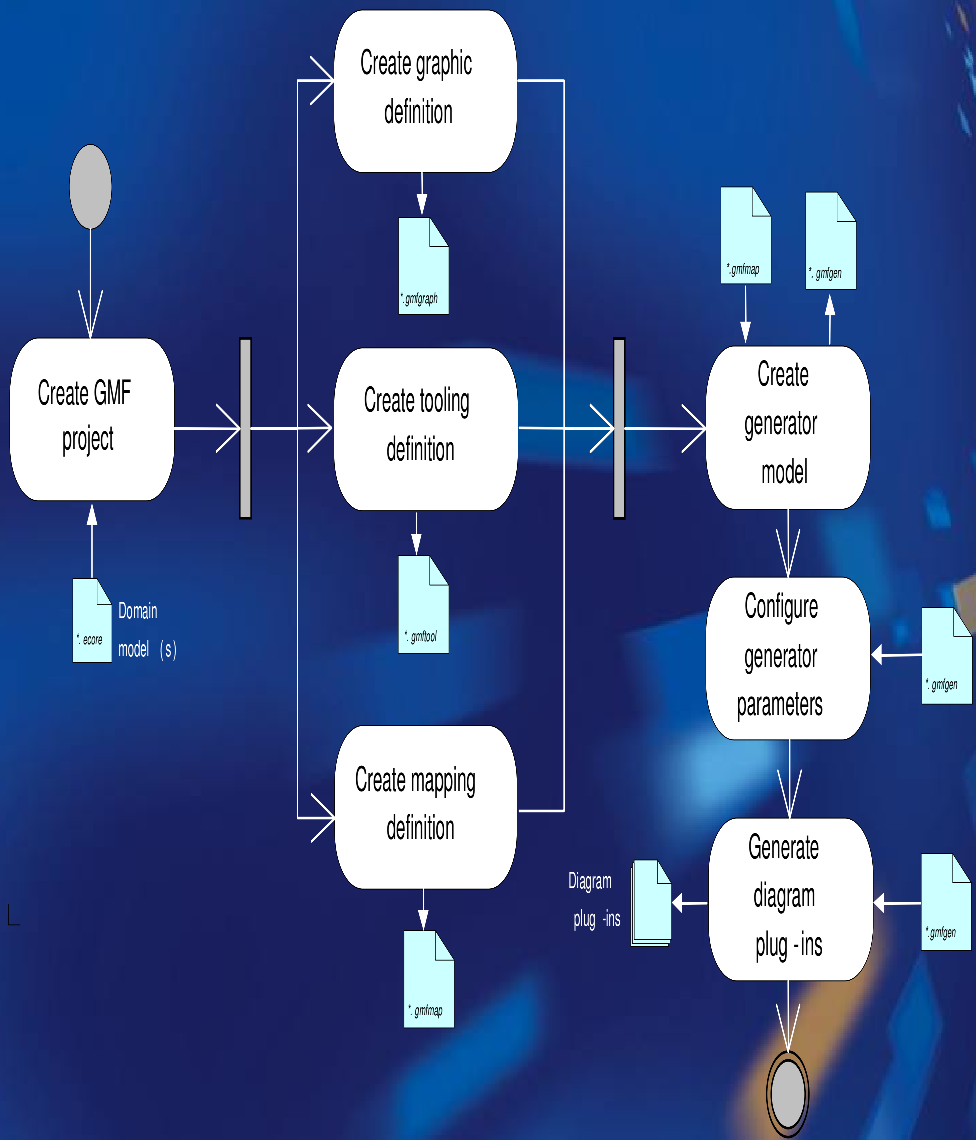
2. Add constraints to metamodel

- No native „constraint“ element in Ecore metamodel!
- In UML, annotations are used to visualize constraints
- *EAnnotation* elements can be used to add constraints to metamodel
- Constraints expressed in OCL
- Validation of constraints by external tool (e.g. Kent OCL Library)

DaimlerChrysler

Implementation with Eclipse Tools

3. Create GMF editor from metamodel



Implementation with Eclipse Tools

3. Create GMF editor from metamodel

A) Graphical definition (gmfgraph):

- Define Figure Gallery based on simple shapes (rectangle, rounded rectangle, polygon, ellipse, polyline, etc.) or custom shapes based on programmatic GEF figures
- Define graphical nodes for the specific editor
- Map graphical nodes to elements of the figure gallery (can be external figure gallery as well)

→ No direct relation to metamodel

→ Can be reused for different editors

DaimlerChrysler

Implementation with Eclipse Tools

3. Create GMF editor from metamodel

B) Tooling definition (gmftool):

- Define tools required for editor:
  - Menu contributions
  - Context menu
  - Toolbar
  - ...
- Minimum tooling definition contains creation tools for the toolbar for each metamodel element

→ No direct relation to metamodel

→ Can be reused for different editors

Implementation with Eclipse Tools

3. Create GMF editor from metamodel

C) Mapping definition (gmfmap):

- Connect all created models (metamodel, gmfgraph, gmftool)
- Map metamodel elements to corresponding graphical element and creation tool
- Define root diagram element

→ Direct relation to metamodel
→ Can use multiple models

→ Create Generator Model from gmfmap

DAIMLERCHRYSLER

Implementation with Eclipse Tools

4. Generate metamodel and editor code

- Generate Java representation of metamodel from EMF generator model
- Generate editor code from GMF generator model

→ resulting projects:

1. *Metamodel project* - contains models and metamodel code
2. *Edit project* - contains model editing code (properties, etc.)
3. *Editor project* - contains editor code (wizards, file extension, etc.)
4. *Diagram project* - contains GEF code for diagram editor

All projects are Eclipse Plug-ins and can be launched!

DaimlerChrysler

Implementation with Eclipse Tools

5. Adjust generated code

- Source code of plugins is available
- JavaDoc-tags mark generated code parts (`@generated`)
- Changes of code required for special use-cases
- Mark manually changed code parts with `@generated NOT`
- Code generation will not override changed parts

DaimlerChrysler

Implementation with Eclipse Tools

6. Run editor in Eclipse

- Generated projects all Eclipse Plugins
- Run plugins withing runtime workbench or export as feature
- Plugins include:
  - Creation wizards
  - Menu extensions
  - Simple model editor
  - GMF graphical editor
  - File extension registration

Agenda

- Goals
- Model-Driven Software Development
- Pro-active Infrastructure (PAI)
- Operational Aspects
- PAI Operational Model
- DSL for the Operational Model
- Model Transformations
- **Implementation with Eclipse Tools**
- Demo
- Conclusion
- References

Agenda

- Goals
- Model-Driven Software Development
- Pro-active Infrastructure (PAI)
- Operational Aspects
- PAI Operational Model
- DSL for the Operational Model
- Model Transformations
- Implementation with Eclipse Tools
- **Demo**
- Conclusion
- References

Conclusion

- Formal modeling is essential for managing complexity
- Operational aspects are too komplex NOT to be modeled
- Metamodeling approaches based on MOF / Ecore provide solid foundation for the creation of custom DSLs
- Eclipse Tools (EMF, GEF, GMF, etc.) can be good starting point for the implementation of DSLs

- GMF still in heavy development, major changes to be expected until version 1.0
- Advanced modeling support (multi-user, rights management, change management, versioning, etc.) has to be provided by other tools or to be self-implemented
- For complete modeling solution for PAI Operational Model, some major effort has to be applied, but generative approach makes solution very flexible and changes can be applied easily

DaimlerChrysler

Agenda

- Goals
- Model-Driven Software Development
- Pro-active Infrastructure (PAI)
- Operational Aspects
- PAI Operational Model
- DSL for the Operational Model
- Model Transformations
- Implementation with Eclipse Tools
- Demo
- **Conclusion**
- References

References

- „Modellierung operational Aspekte von Systemarchitekturen“ - Mirko Bleyh
  http://www.mirkobleyh.de/diplom/Diplomarbeit.pdf
- GMF Tutorial Part 1 + 2
  http://wiki.eclipse.org/index.php/Graphical_Modeling_Framework
- IBM Model Transformation Framework
  http://www.alphaworks.ibm.com/tech/mtf
- Kent OCL Library
  http://www.cs.kent.ac.uk/projects/ocl/
- „Modellgetriebene Softwareentwicklung“ - M. Völter, T. Stahl - dpunkt.verlag
  www.voelter.de
- „Moderne Softwarearchitektur“ - J. Siedersleben - dpunkt.verlag

The End

Thank you!

Contact: mirko.bleyh@gmx.de