Outline

• MDA/MDD challenges
• The Integration Challenge
  – Rich Services Framework
  – Logical and Technical Blueprint
  – Development Process
• Example: Failure Management
  – Model System with Failures
  – Verify the Model
  – Leverage the RichService Process
  – RIS – Managed Service
• Case Study: CLS
  – Requirements
  – Logical vs. Deployment Models
  – Failure Hypothesis
  – Model Verification
• Conclusions
MDA/MDD Challenges

• Requirements for avionics and automotives are highly heterogeneous
  – Need for modeling techniques that result in a consistent, integrated requirements specification

• Modeling plays an important role in both requirements engineering and architecture specification, serving as a common interface between them
  – Need for traceability between architecture specifications at various levels of abstraction and the requirements that are implemented at these levels

• Modeling is a key engineering activity, but state of the art focuses on forward engineering
  – Need to support a full round-trip engineering
  – Models should be reusable not only across development activities for the same product, but also across the development of different products
MDA/MDD Challenges

• Models need to be expressive so as to support both generic and domain-specific aspects of the system
  – Need to integrate different models and ensure consistency between them

• Modeling reveals key insights about the specifics of the application domain
  – requires expertise both in software engineering and in the domain the model represents
  – Need to foster cross-disciplinary programs in engineering education

• In the long run, MDA/MDD will be the norm rather than the exception - the systems we are developing are too complex, societally important and critical not to be done right!
The Integration Challenge

- Main issue for complex distributed systems is integration

- Yesterday Prof. Salzwedel presented a telling figure:

  In the integration phase the aerospace industry discovers that they have to change 70% of the system

- To address this problem we need models:
  - From the requirement phase
  - Supporting requirement validation (help finding contradictions and incompleteness)
  - That help tracking requirements to implementation (even with code generation)
  - That support decomposition of systems (so that different suppliers can work on different parts)
  - Formally capture knowledge from domain experts
Our Approach

• To address this problem we need models:
  – From the requirement phase
  – Supporting requirement validation (help finding contradictions and incompleteness)
  – That help tracking requirements to implementation (even with code generation)
  – That support decomposition of systems (so that different suppliers can work on different parts)
  – Formally capture knowledge from domain experts

• Service-Oriented
  – Uses models of interaction patterns
  – Natural to specify requirements of distributed systems in this way
  – Foster separation of concerns and loose coupling between services

• Iterative Development Process
  – Supports requirements tracking through the development
  – Support collaboration with domain experts to create and validate models

• Support of Formal Verification
  – Of both logical and deployment artifacts
  – Of some functional and non functional properties
Rich Services

• An extension of the service notion, based on an architectural pattern
  – Supports both “horizontal” and “vertical” service integration

• Manage the complexity of a system-of-systems
  – Decomposing into primary and crosscutting concerns
  – Providing flexible encapsulation for these concerns
  – Generating a model that can easily be leveraged into a deployment

• Service Composition
  – Service orchestration at the infrastructure or application level
Rich Services Architectural Blueprint

<<Rich Service>> S

Crosscutting Infrastructure Services

Policy
Encryption
Logging
Failure Manager
...

Rich Application Services

S.1
S.2
...

Integration of Application Services

Support for Decomposition

<<Infrastructure>> Services

Route/Interceptor

Decoupling via Messenger

Dual Decoupling

<<Rich Service>> S.n

Policy
Encryption
Logging
Failure Manager
...

Rich Application Services

S.n.1
S.n.2
S.n.m

Support for Decomposition
Rich Services Development Process
Example: Failure Management

Use cases/ Scenarios & Failure Hypothesis

SADL

Verification

Code generation & Deployment
Use cases and Failure Hypothesis

- **Domain Model**
- **System Services**
- **Failure Hypothesis:**
  - Which components/domain elements can fail
  - How many failures are allowed concurrently
Is the proposed architecture safe?

- **Specification & Failure Hypothesis (SADL)**
- **S2Promela tool**
- **SPIN Model Checker**

Verification

Use cases/Scenarios & Failure Hypothesis

Code generation & Deployment
Process Implementation

Use cases/ Scenarios & Failure Hypothesis

SADL

Logical Layer:
Roles, Services, Messages,…

Mapping

Physical Layer:
Sensors, Actuators, Busses, ECUs,…

Verification

Code generation & Deployment
Managed Services

- Increase dependability of SOA by extending the service model to a Managed Service
- A managed service adds two elements to a service
- Fault Detector identifies erroneous interactions due to a failure
- Mitigator defines a recovery strategy for the failure
- We have encoded these SOA extensions in a Service ADL
Case Study: Central Locking System (CLS)

• Example from the automotive domain
  – CLSs are very complex and require the integration and cooperation of up to 20 electronic control units (ECUs)
  – Complex set of comfort functions are connected to the locking and unlocking of the vehicle
    – Examples:
      – movement of the driver’s seat
      – setting of radio presets

• Functional Requirements
  – Unlock the door when the key fob unlock button is pressed.
  – Lock the door when the key fob lock button is pressed.
  – Unlock all doors when an impact is detected.

• Quality Requirements
  – The time between an impact and the door unlock must be less than 0.1 seconds.
  – The emergency door unlock must be guaranteed even in presence of ECUs failures.

• Various types of requirements
CLS Services

Diagram showing the relationships and states of various components involved in a control system, including interactions like 'unlock', 'unlock_ok', and 'door_unld_sig'.
• **Requirement**
  - The time between an impact and the door unlock must be less than 0.1 seconds.

• **Define a deadline in the model**
Deployment Model

- We use a deployment model that defines the physical layout of the system.
- Each role defined in the sequence diagrams of the rich services is associated to some component.
- Failure information is connected to components.
- Logical and the deployment models present RS elements.
- Their configurations can differ.
Failure Hypothesis

- We defined a failure model to describe different failures
- We use this model to express a failure hypotheses
- A failure hypothesis relates failure effects to their causes
- It enables the specification of Fault Detectors and Mitigators
Managed Service

• How do we deal with possible failures?
  – Model the behavior with deadlines
  – If deadlines violated in the running system then a mitigation service is executed
Model Transformation: from MSCs to State Machines
- The failure hypothesis influence the generation of components’ state machines
- Example: Introduction of sink state to represent fail stop
Safety Verification

• We have developed a tool (M2Code) that generates state machines from the interaction specification
• State Machines are generated from Sequence Diagrams and Failure Hypotheses
• M2Code can generate Promela models that can be verified with the SPIN model checker
Conclusions

• MDA/MDD is at our doorsteps
• How can we leverage the potentials of MDA/MDD:
  – domain-relevant modeling
  – model integration and transformation
  – modeling at all scales
  – model exploitation at development and runtime
  – models as assets
  – understand modeling workflows, data and product models
  – build strong industry/university partnerships
Conclusions

- **Service-Oriented Failure Management**
  - Focus on interactions as unit of modularity
  - Improved syntax and semantics for interaction specifications
  - Explicit, scalable models for failures, unmanaged and managed services
  - Novel Architecture Definition Language (ADL) integrating services, logical and deployment architectures with failure models
  - Transformation from ADL to model checker for automated fail-safety checking
Thank you!