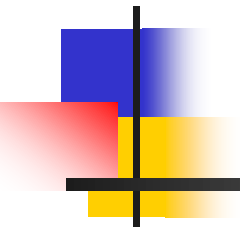
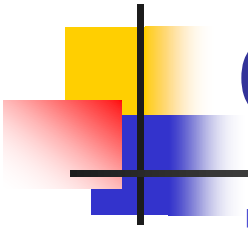


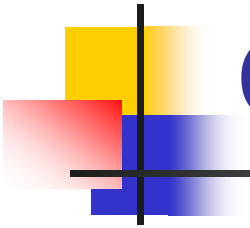
Developing JAUS Compliant Communication Infrastructures for C2 of Unmanned Systems through MDSD

İskender YAKIN
Faruk BELET
Ferhat KUTLU
yakin,belet,fkutlu@cs.bilkent.edu.tr



Outline

- Introduction
  - Context
  - Problem Statement
- Domain Analysis
  - Domain Description
- BNF Grammar
- Meta-model
  - Abstract Syntax
  - Concrete Syntax
  - Static Semantics
  - Example Model
- Profiling
- Model to Model Transformation (M2M)
- Model to Text Transformation (M2T)
- Conclusions



Context

- Unmanned systems
  - Reduce exposure of personnel to harmful environments
  - Perform tasks not possible for humans
  - Provide cost effective solutions to repetitive tasks.
- A large number of unmanned system products are being introduced to the market.
- Many of these systems are characterized as task dependent and non-interoperable.
- Interoperability?
  - Heterogeneous Systems
- A joint architecture
  - JAUS (Joint Architecture for Unmanned Systems)

Unmanned Systems

Common Operator Control Unit (OCU)

Rugged PDA

Heterogeneous Unmanned Platforms

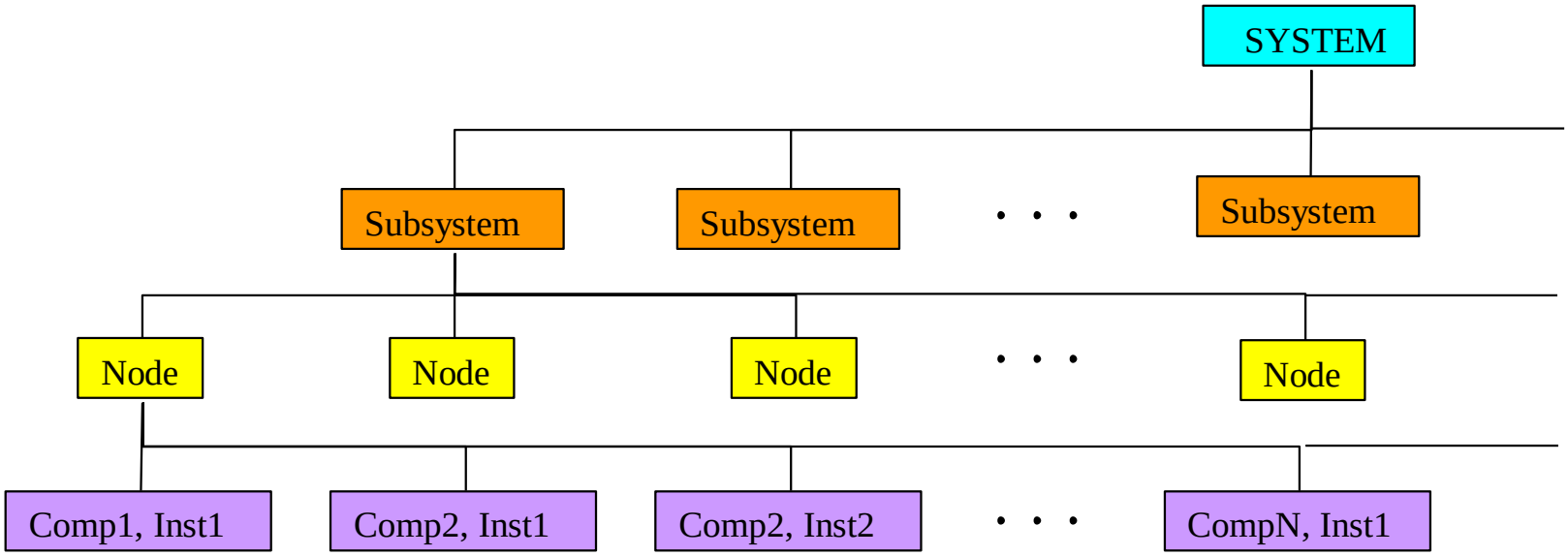
JAUS

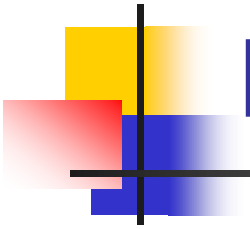
- JAUS provides the necessary definitions to develop architecture to support the following objectives,
  - Support all classes of unmanned systems
  - Rapid technology insertion
  - Interoperable operator control unit
  - Interchangeable/interoperable payloads
  - Interoperable unmanned systems

Joint Architecture for Unmanned Systems (JAUS)

- The Joint Architecture for Unmanned Systems (JAUS);
  - is the architecture for development and acquisition of Unmanned Systems,
  - provides a description of the structure of JAUS based systems,
  - describes a component based system structure.

JAUS System Topology





Problem

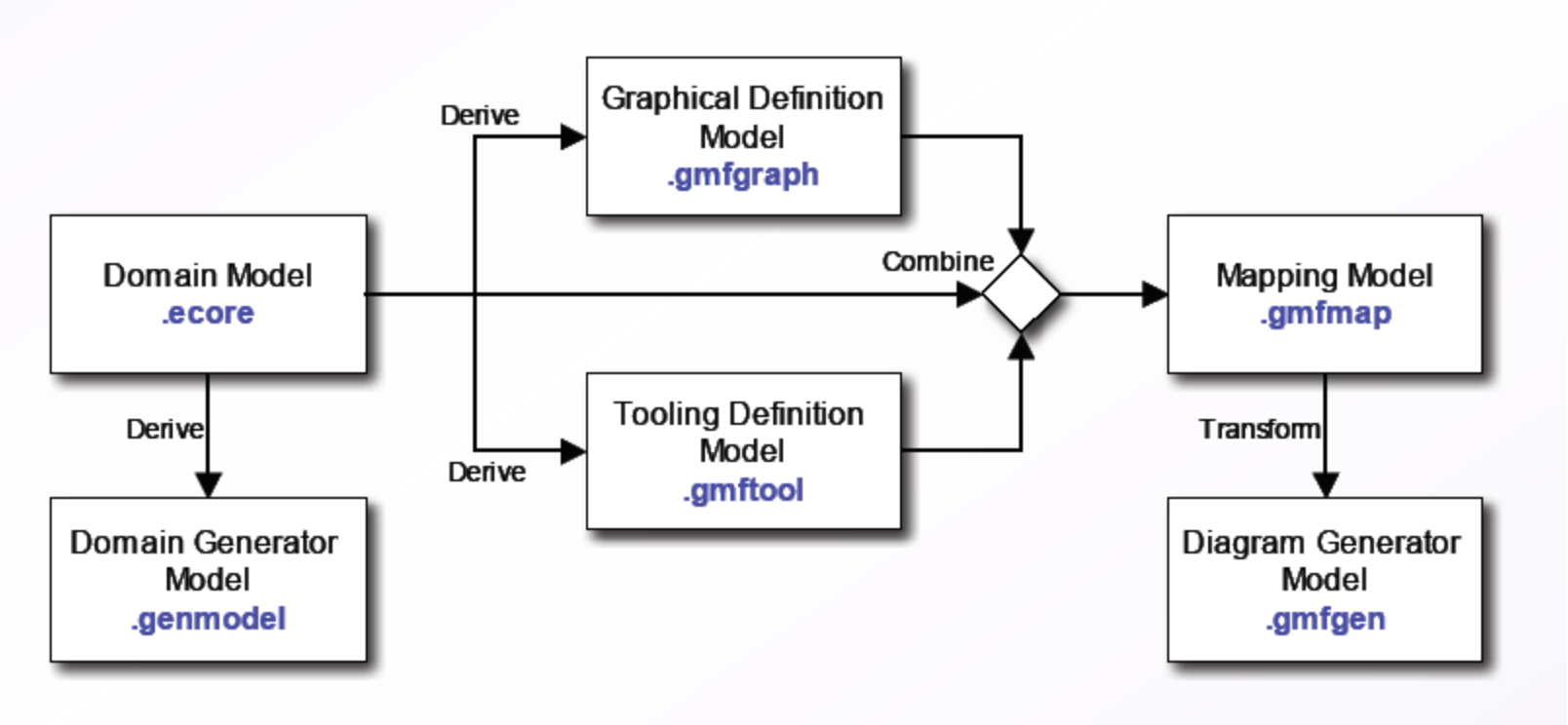
- Even if JAUS provides a flexible and extendable infrastructure for developing unmanned systems, as the number of components increases in systems, it becomes difficult to define the communication parameters among components without any design aid.

- We aim to develop a graphical design tool for JAUS compliant systems through MDSD with the use of EMF (Eclipse Modeling Framework) and GMF (Graphical Modeling Framework) frameworks in Eclipse Ganymede 3.4.

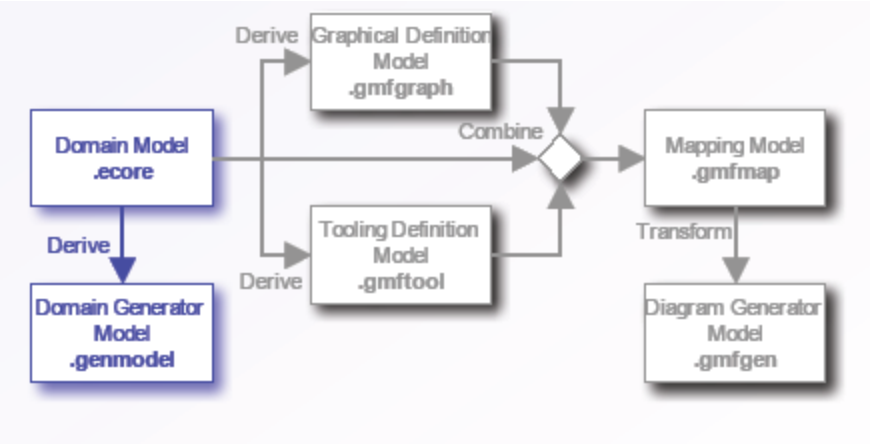
DSL Grammar

- <System> ::= <SystemEntity> <SubsystemList>
- <SubsystemList > ::= <Subsystem> | <Subsystem> <SubsystemList>
- <Subsystem> ::= <SystemEntity> <NodeList>
- <NodeList > ::= <Node > | <Node> <NodeList>
- <Node> ::= <SystemEntity> <IPAdsress> <ComponentList>
- <IPAddress> ::= <Number> . <Number> . <Number> . <Number>
- <ComponentList> ::= <Component> | <Component> <ComponentList>
- <Component> ::= <SystemEntity> <InstanceList>
- <InstanceList> ::= <Instance> | <Instance> <InstanceList>
- <Instance> ::= <SystemEntity> <JAUSAddress> <ConnectionList> <ConnectionList>
- <JAUSAddress> ::= <Number> . <Number> . <Number> . <Number>
- <ConnectionList> ::= <CommLink> |<CommLink> <ConnectionList>
- <CommLink> ::= <Instance> <Instance> <Port>
- <SystemEntity> ::= <name> <ID>

GMF Model Chain

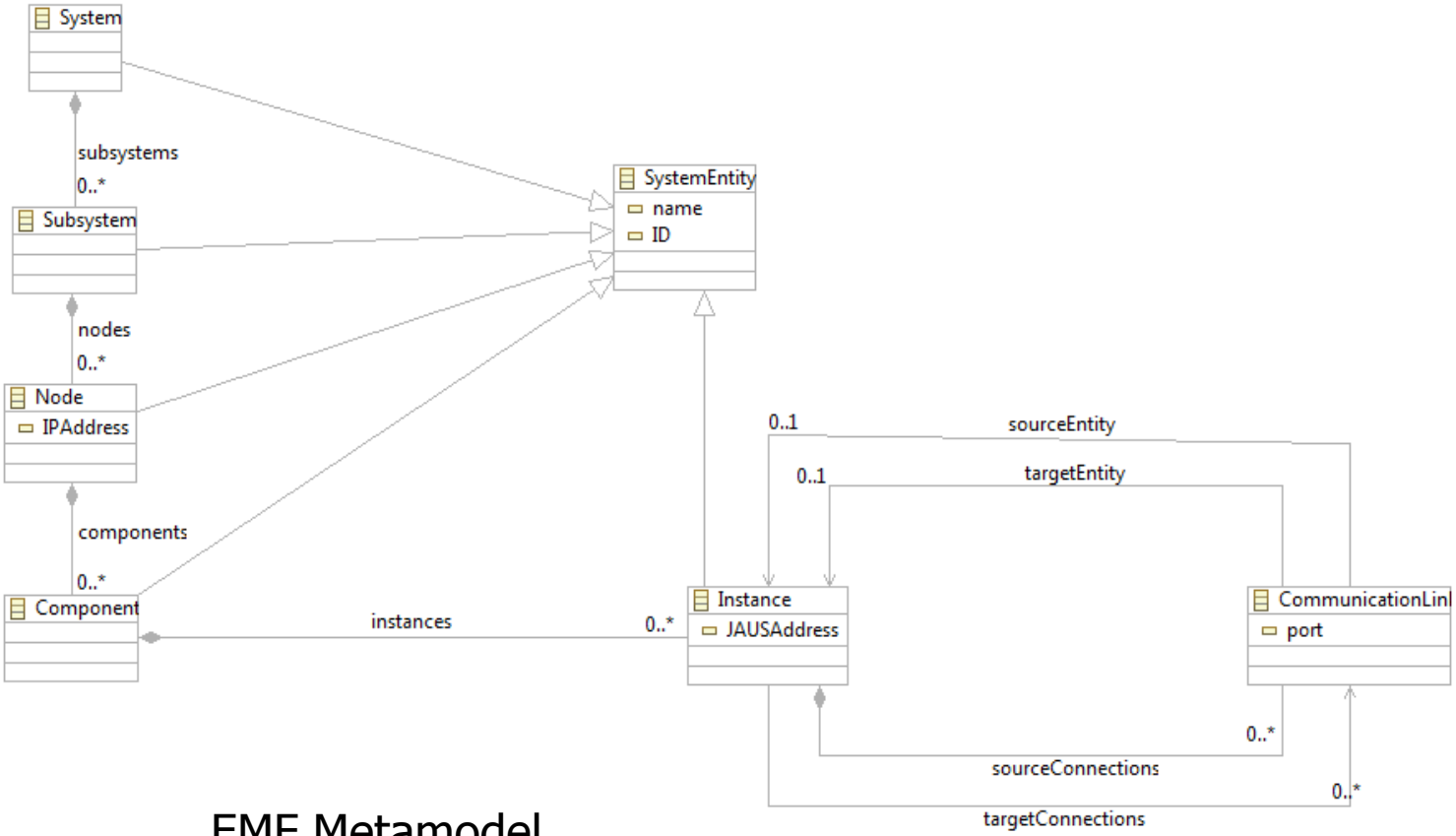


GMF Model Chain



- Domain Model (EMF based)
  - Metamodel in MDSD
  - Defines abstract syntax
  - Structural constraints used to model static semantics
- Domain Generator Model
  - Decorator model around the metamodel
  - Holds information about e.g. package base names, templates to use, ...

Abstract Syntax



EMF Metamodel

Static Semantics

Control of the IP addresses in Nodes to sustain each Node has a unique IP Address.

context *Subsystem* ***inv****:*

*Node.allInstances()->**for All**(n1|Node.allInstances()*

*->**forAll**(n2|n1.IPAddress<>n2.IPAddress))*

Control of the JAUS addresses in Instances to sustain each Instance has a unique JAUS Address.

context *Component* ***inv****:*

*Instance.allInstances()->**forAll**(j1|Instance.allInstances()*

*->**forAll**(j2|j1.JAUSAddress<>j2.JAUSAddress))*

Control of the ports in Communication Links to sustain each Communication Link has a unique port.

context *Instance* ***inv****:*

*CommunicationLink.allInstances() ->**forAll**(p1|CommunicationLink.allInstances()*

*->**forAll**(p2|p1.port<>p2.port))*

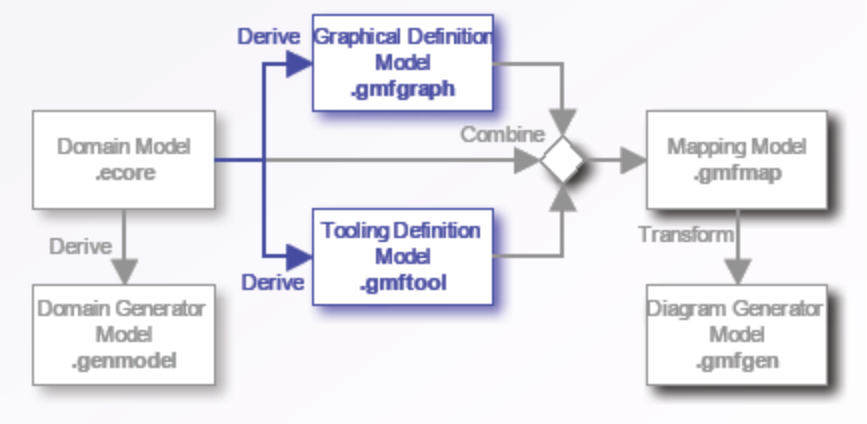
Control of the connections in an Instance to sustain sources and targets to be different Instances.

context *Instance* ***inv****:*

CommunicationLink.allInstances()

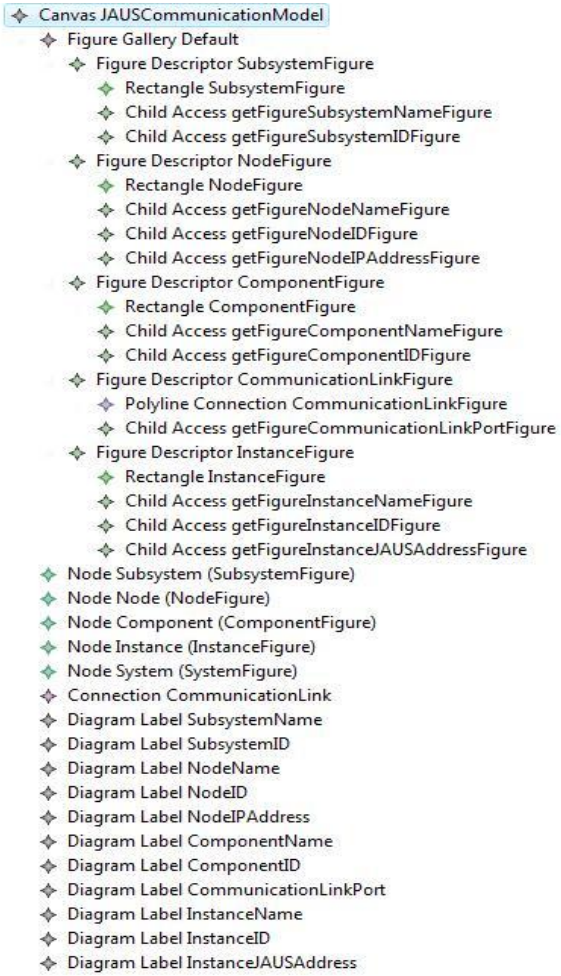
*->**forAll**(c|c.sourceEntity<>c.targetEntity)*

GMF Model Chain

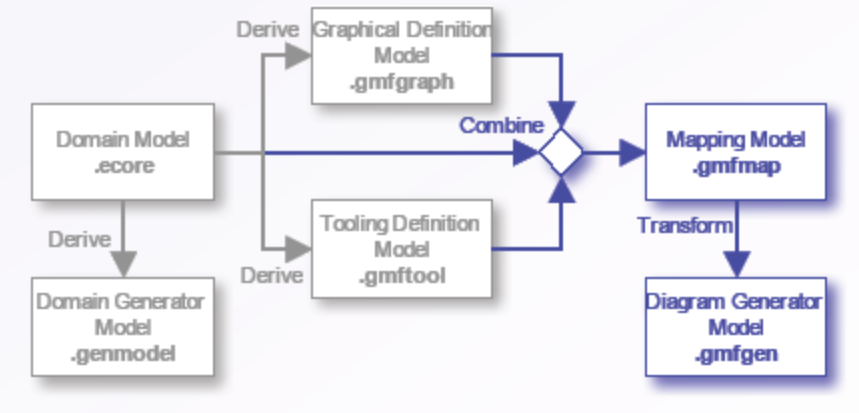


- Graphical Definition Model
  - Model defining graphical notation: shapes, decorations, labels and connections
  - Defines concrete syntax
- Tooling Definition Model
  - Model for the editor's palette

Concrete Syntax

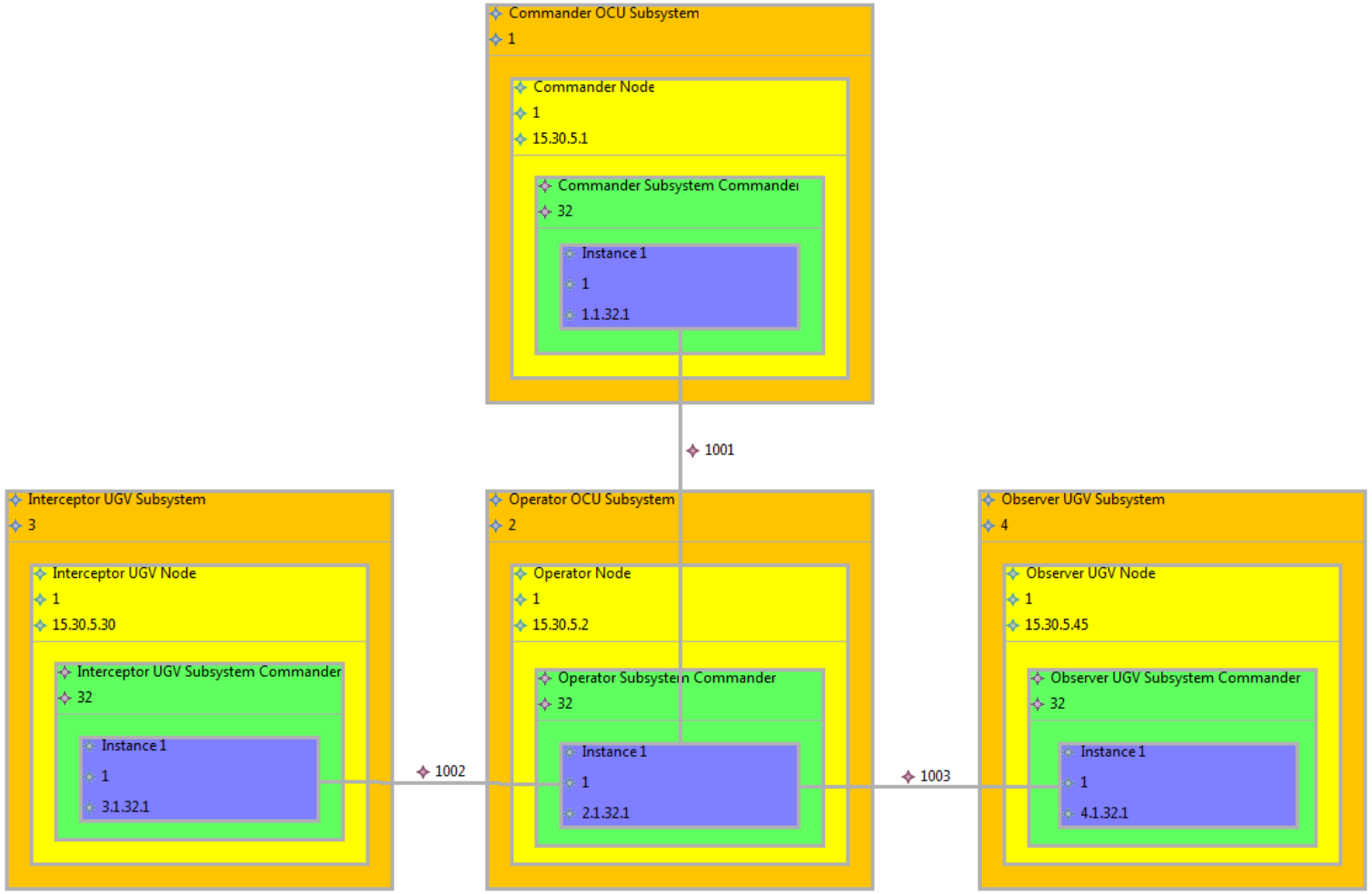


GMF Model Chain



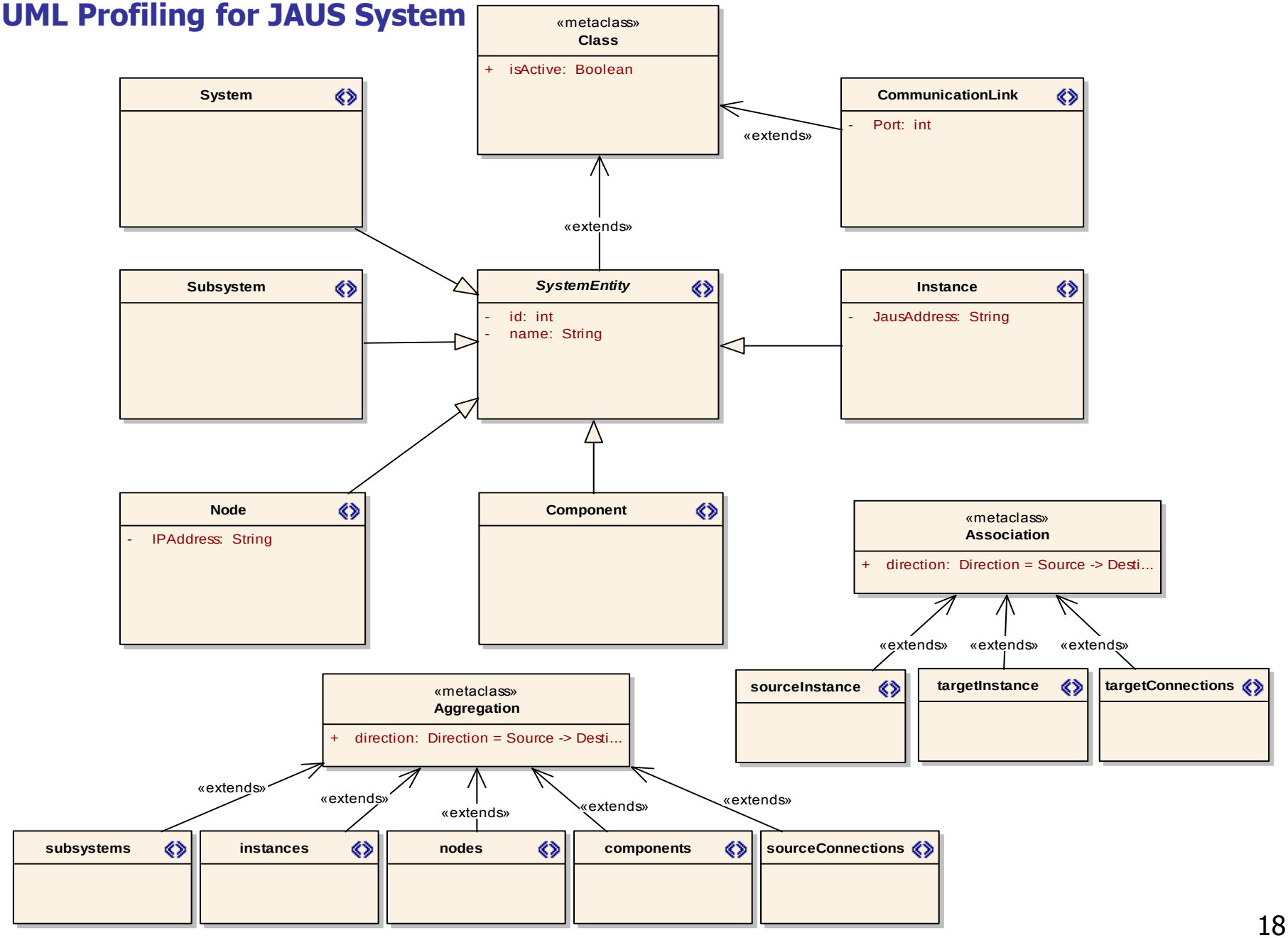
- Mapping Model
  - Binds graphical definition and tooling with the meta model
  - Defines how graphical elements are mapped to a model instance with respect to the meta model
- Diagram Generator Model
  - Decorator model around the mapping model
    - Holds information such as package base names, enabling of validation, -decorators, ...

JAUS Communication Model

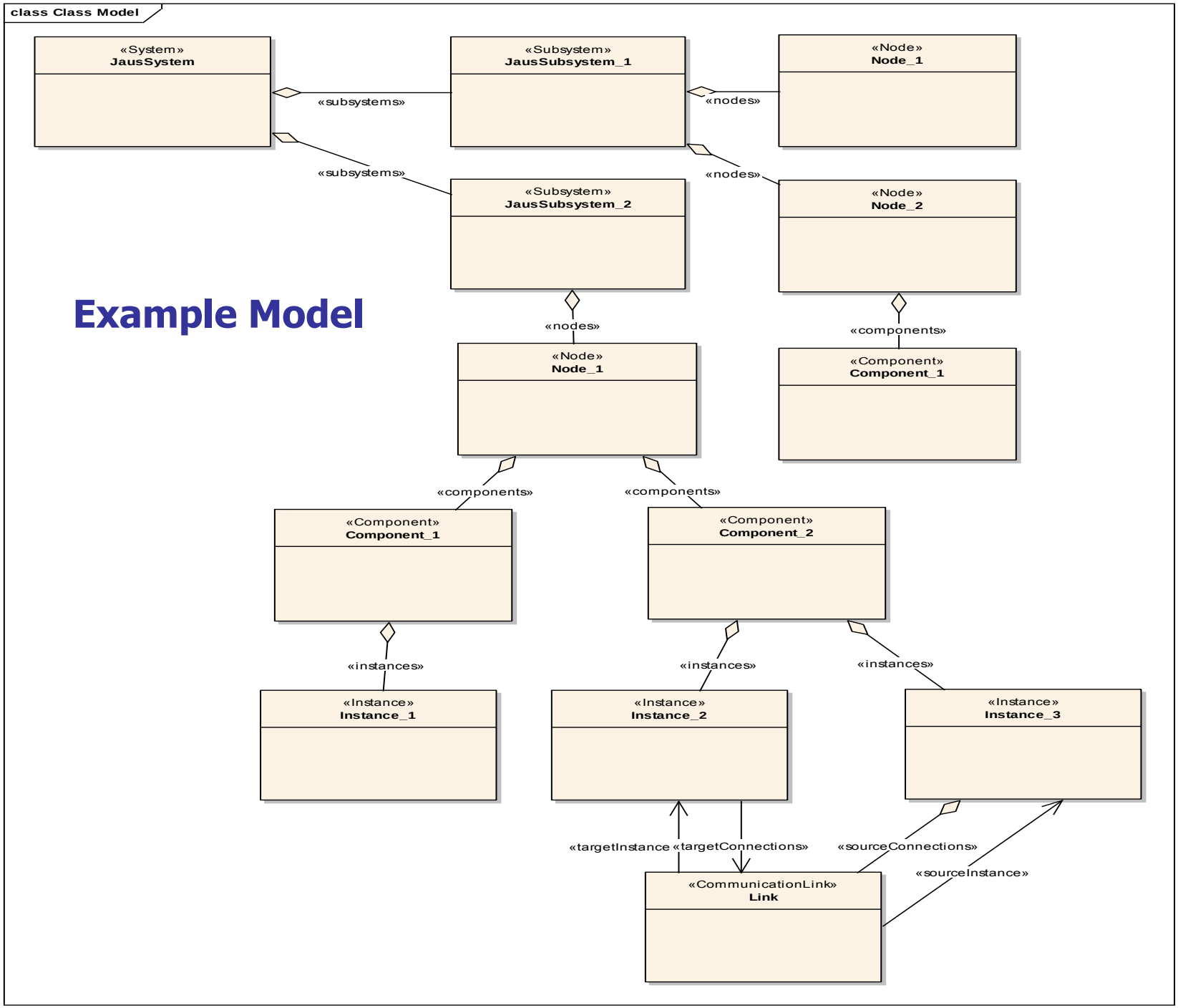


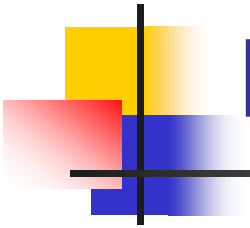
class JAUSProfile

UML Profiling for JAUS System



Example Model

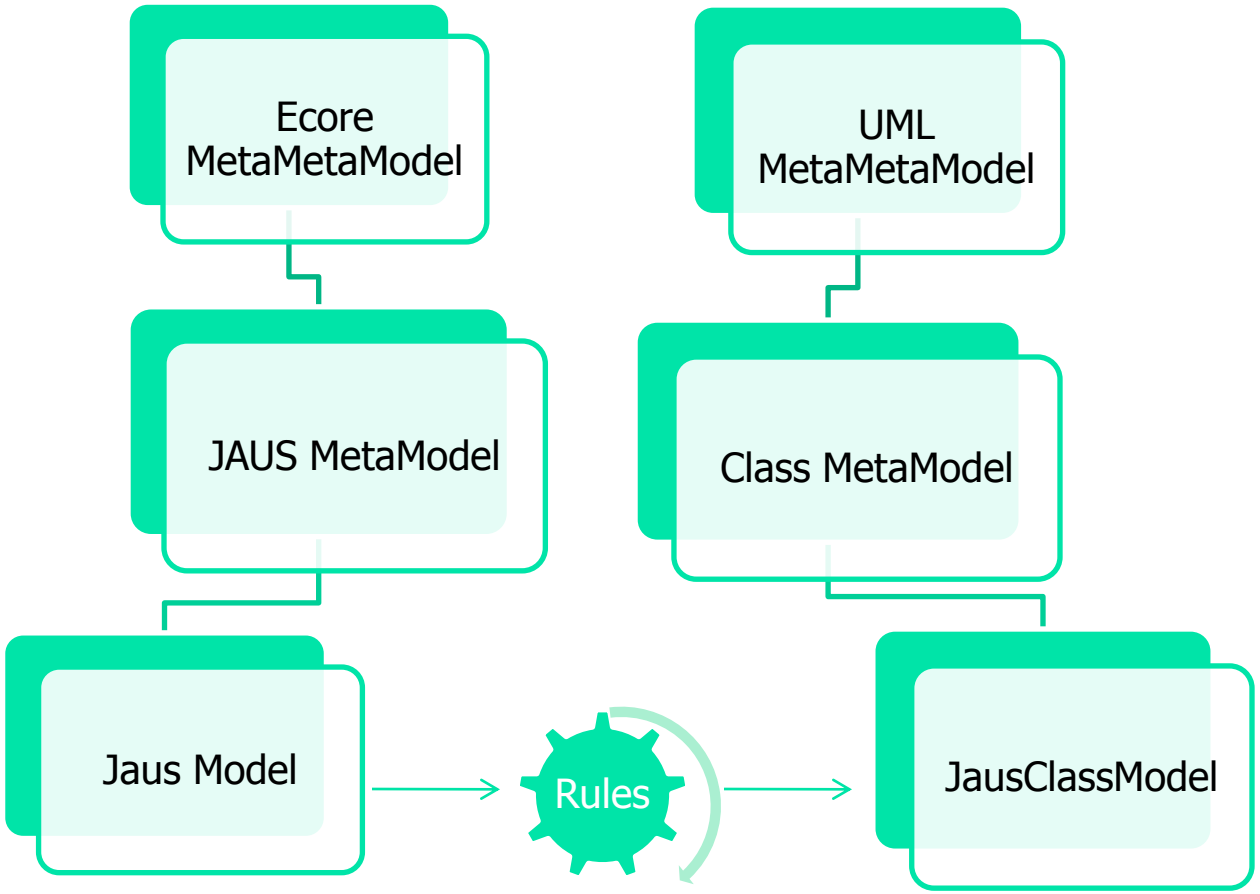




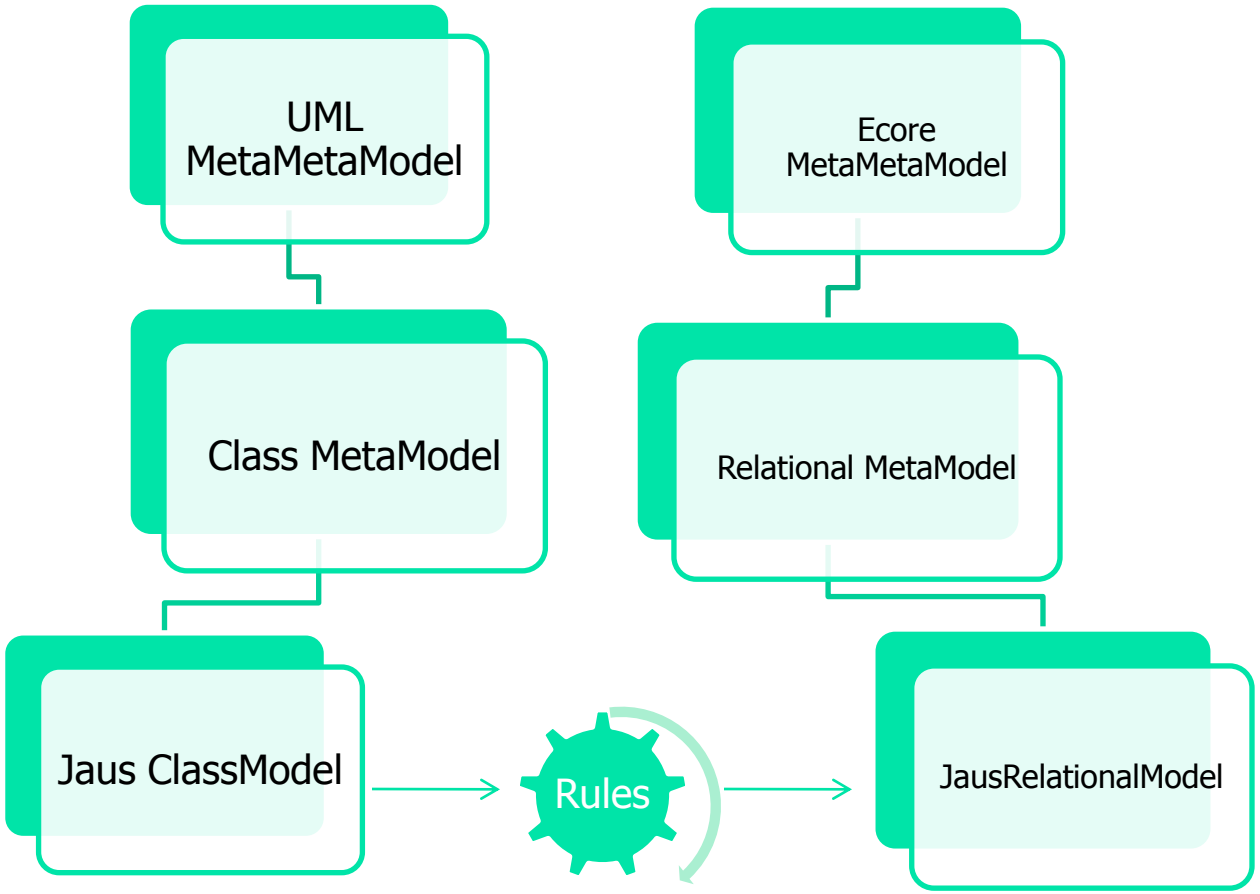
Model to Model Transformation

- ATL (ATLAS Transformation Language) is used for M2M transformation
  - ATL provides ways to produce a set of target models from a set of source models
  - Ecore Model to Class Model
  - Class Model to Relational Model
- ATL Transformation Process
  - Define Target & Source Metamodels
  - Given a source model conforming to the source metamodel create the target model conforming to the target metamodel.

Model to Model Transformation (Step1)

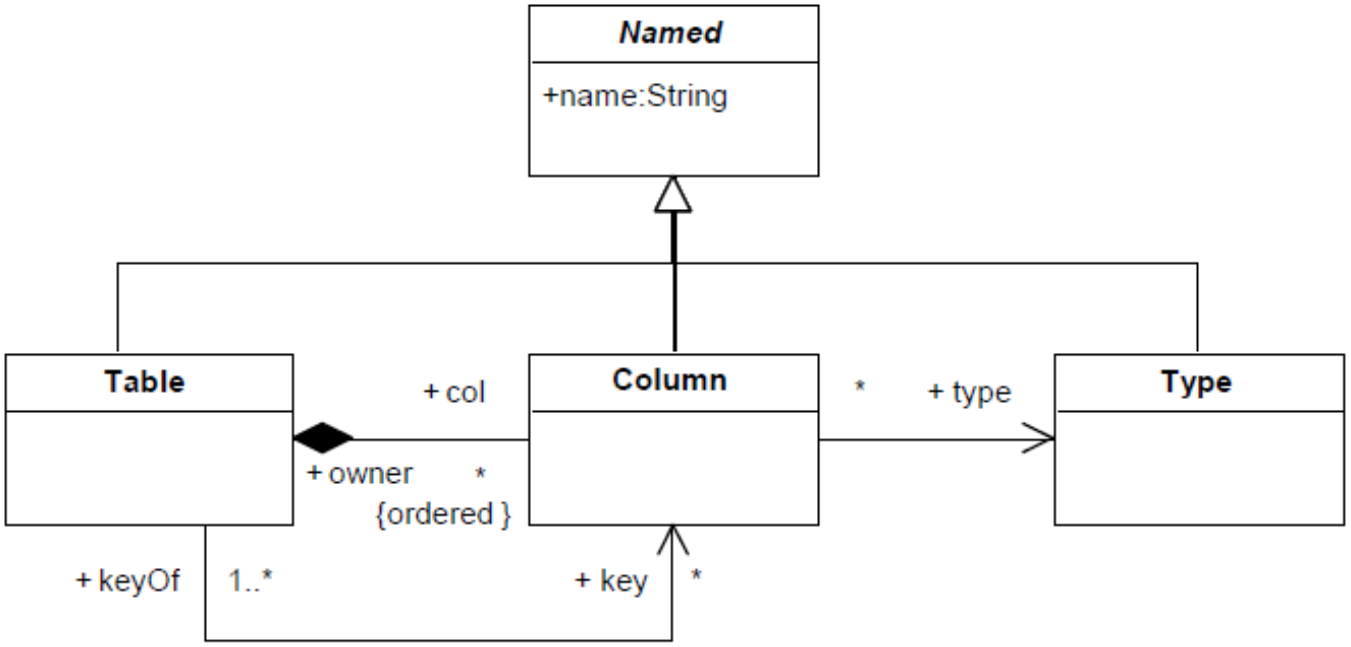


Model to Model Transformation (Step 2)



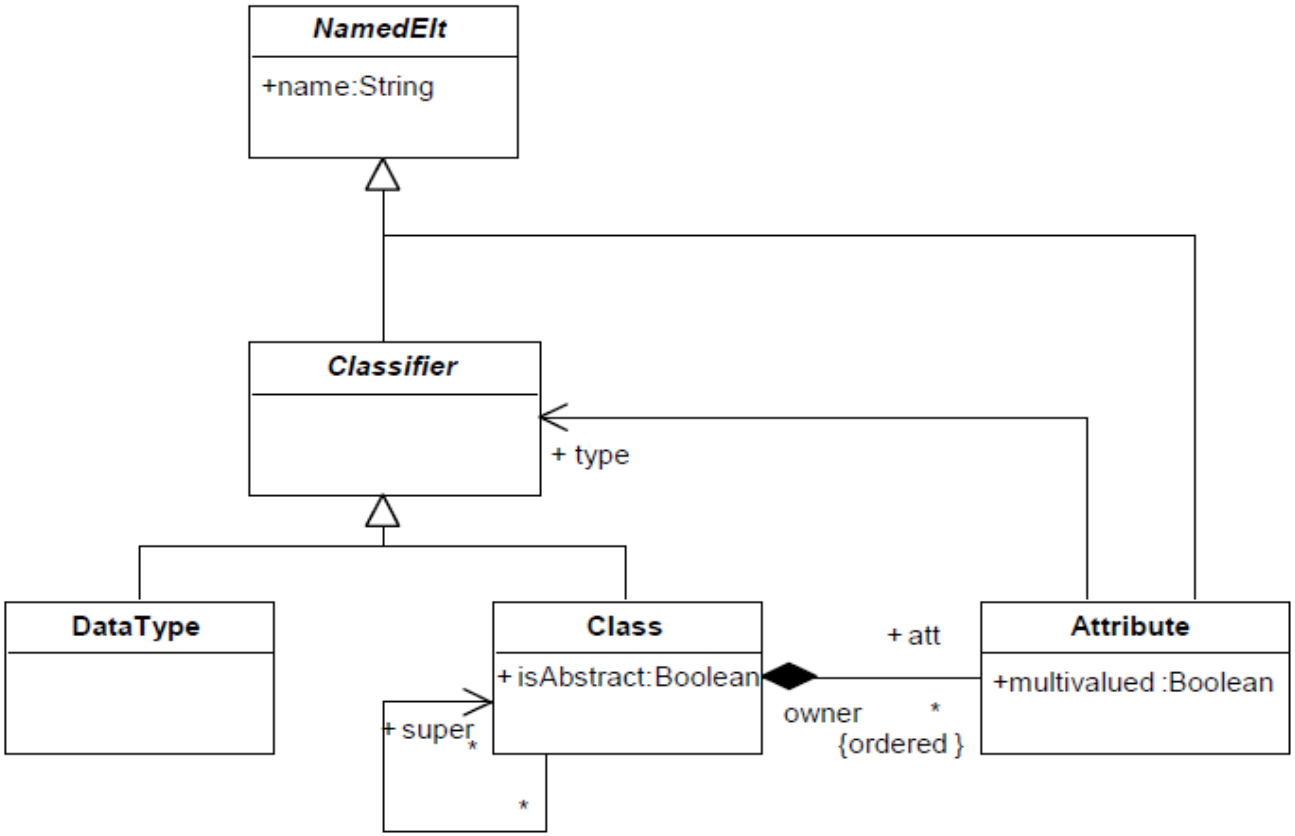
Model to Model Transformation

- Relational metamodel



Model to Model Transformation

- Class metamodel



Model to Model Transformation

Sample ATL Transformation Rule

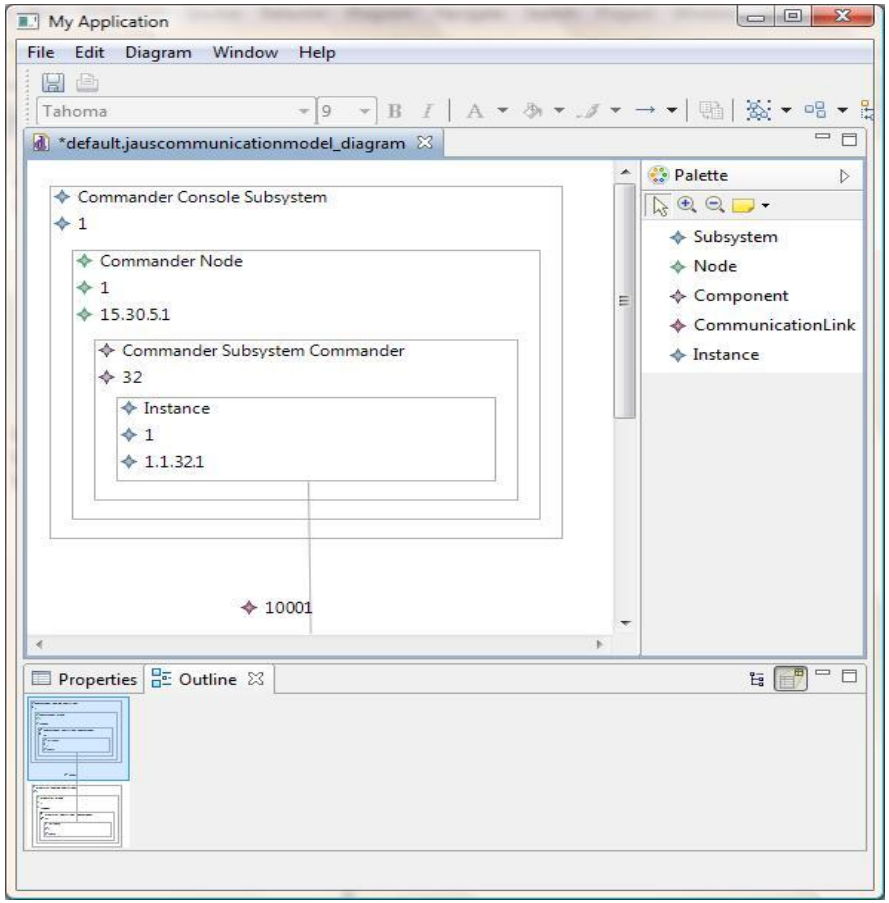
```
rule DataType2Type {
        from
                dt : Class!DataType
        to
                out : Relational!Type (
                        name <- dt.name
                )

rule ClassAttribute2Column {
      from
            a : Class!Attribute (
                  a.type.oclIsKindOf(Class!Class) and not a.multiValued
            )
      to
            out : Relational!Column (
                  name <- a.name + 'Id',
                  type <- thisModule.objectIdType
            )
}
```

Model to Model Transformation

- Example relational database schema.

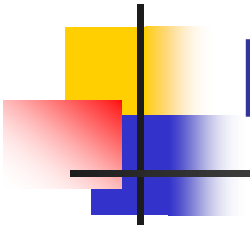
SubsystemID	NodeID	ComponentID	InstanceID	IP
1	1	32	1	15.30.5.1
2	1	32	1	15.30.5.2
3	1	32	1	15.30.5.30
4	1	32	1	15.30.5.45

Model to Text Transformation



- JAUS Communication Model Editor (a model to text transformation example)
- Generated from GMF Model Chain.
  - 35.000 lines of code

Model to Text Transformation



- For model to text (M2T) transformation we use Eclipse JET (Java Emitter Templates) plug-in.

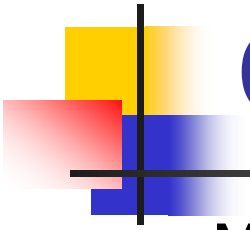
- A JET template includes both the static part of the code such as,

  ```
  // Commander
  new UDPInterface(inetAddress, parameter(port), false, true)
  ```

  and dynamic parameters that should be put into the code.

- For this example the code segment 'parameter' is the data that we are looking for in our model. When this parameter is passed from EMF .ecore to the template we get the following auto generated code line,

  new UDPInterface(inetAddress, **1001**, false, true).

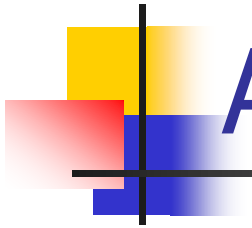


Conclusions

- Meta-model definition takes a considerable amount of time.
- Implementing a given architecture decreases the design time but requires the architecture to be understood in detail.
- 100% auto generated code with Eclipse EMF/GMF.
  - However, it takes a great amount of time to define the models and mapping among these models.
- GMF has some problems with creating the Mapping Model.
- Use set/create transactions in EMF.
  - Eliminates concurrent update exceptions.

Conclusions

- Changing requirements are easily reflected to the models.
- BNF Grammar helps to understand the system and the structure of system entities.
- Relational Metamodel for storage case of our system is seen as a good practice.
  - Persistent data storage.
  - Running SQL queries.
  - Using dynamic views provided by relational databases
  - Storing data from several communication models
- By using model-to-text transformations code including the communication parameters of the system can be auto generated.
  - This reduces time spent for maintenance and also reduces the time for testing the modified system.
  - Since we have the communication parameters in our model, changing the parameters at the model and generating the code dependent on these parameters is possible through model to text (java code) transformation.



Acknowledgement

Dr. Bedir Tekinerdoğan

References

- Hui Huang, Autonomy Levels for Unmanned Systems (ALFUS) Framework, JAUS WG Meeting August 26, 2004. Pittsburgh, PA.
- C4ISR Handbook for Integrated Planning (CHIP), DoD Integrated C4I Architectures Division, April 1998
- C4ISR Interoperability Working Group, Department of Defense. Levels of Information Systems Interoperability (LISI). Washington, D.C., 1998
- NATO C3 Technical Architecture Volume 2 Architectural Descriptions and Models Version 7.0, 15 December 2005
- JAUS, Domain Model, Volume 1. Version 3.2., 10 March 2005.
- JAUS, Reference Architecture Specification, Volume 2, Part 1, Architecture Framework.

Questions & Answers