Model-driven development of Flight Desk Displays

Cevahir TURGUT

Enver Veli ATABEK

Outline

- Introduction / Description of the Domain
- Domain Analysis/Domain Concepts
- Grammar
- Meta-modeling from Scratch
- Static Semantics
- Meta-modeling using Profiling at UML 2.*
- Model to Model Transformation: FDD to GMF
- Code Generation (FDD Model to C++/OpenGL)
- Discussion/Conclusion/Lessons Learned

Introduction

- What is Flight Deck Displays (FDD)?
  - Display systems used at glass cockpits
- Avionics Area
  - Safety critical software is needed and requested
  - Software products of FDD are also at high safey-critical level
- Why is this domain selected?
  - Professional Experience on Domain at Work
  - Suitable for MDSD Approach

Introduction

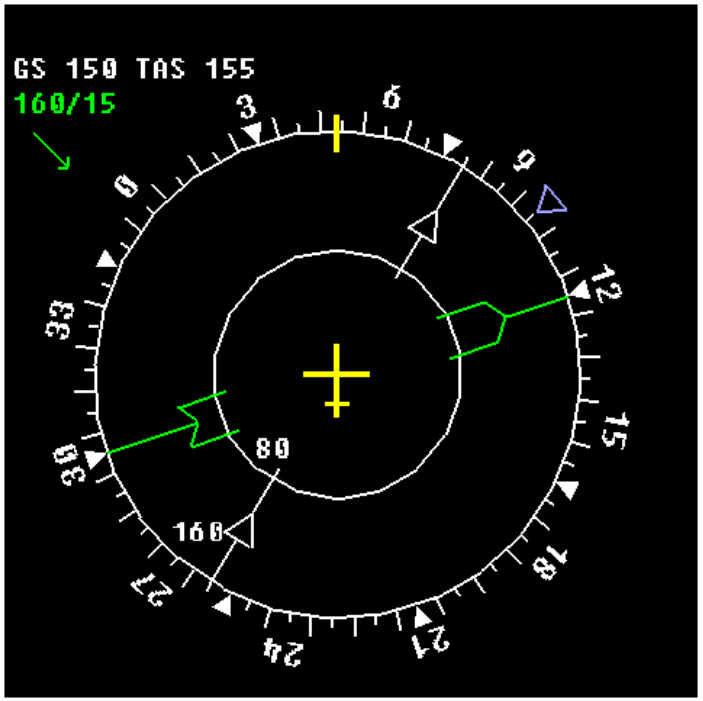
- Aim of FDD Modeling
  - To increase productivity and produce easily reusable SW
    - Visual software development using visual model elements of domain
    - Code Generation from models
  - To produce SW with reduced certification costs
    - Code Generation conforms to standards (i.e. Khronos OpenGL ES-SC 1.0
  - Reduce maintenance costs

Description of the Domain

- What is glass cockpit? [3]

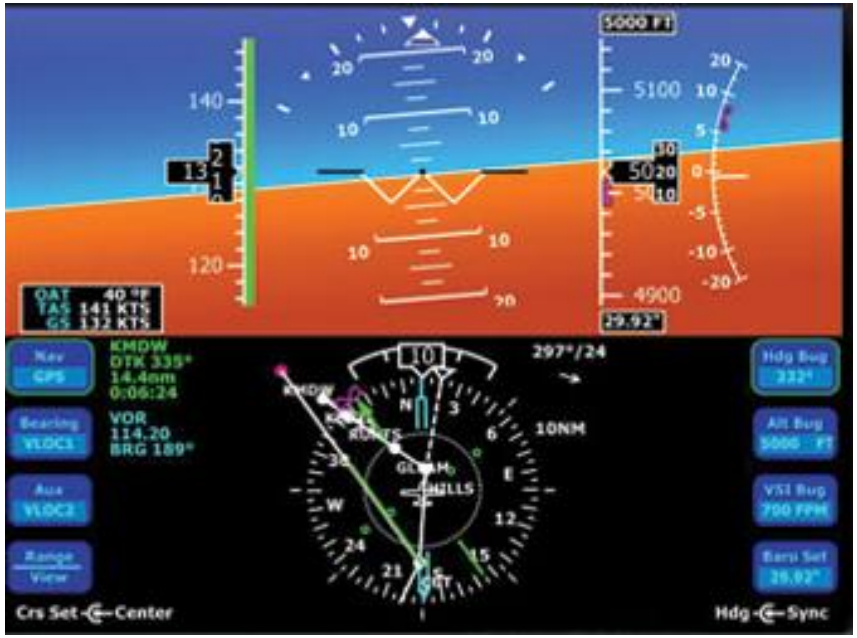
Description of the Domain

- Some glass cockpit screen shots



Description of the Domain

- Some glass cockpit screen shots



Description of the Domain

- FDD are for aircrafts (i.e. helicopters, airplanes)
  - Interactions are done via glass cockpit systems
- What is Glass Cockpit?
  - Interaction are done via electronic display systems instead of old manual switches and indicators
- What is NOT Flight Deck Display?
  - Graphical User Interface
  - Just graphics
- Today's new aircrafts are equipped with glass cockpit systems

Domain Analysis/Domain Concepts

- Personal Professional Experiences
  - Software Development/Verification Activities at Avionics Domain more than two Avionics Projects
- FAA Guidelines
  - FAA: Federal Aviation Administration [1]
  - Mission of FAA: To provide the safest, most efficient aerospace system in the world
  - Determines Regulations & Policies for Avionics
    - Advisory Circulars (ACs), FAA Regulations, Handbooks & Manuals
- Some aircraft documentations
  - i.e. DO178B: Software Considerations in Airborne Systems and Equipment Certification [2]

Domain Concepts

- **Display**: The main scene. A display contains symbologies.
- **Symbology**: A place holder that groups the components.
- **Text**: Texts. Usually used to display information, warnings, messages and errors. There are three kinds of texts; Warning, Normal, Error.
- **Label**: Label is a definitive component for another component. Labels are separated into two: TextLabel and IconLabel.
- **TextLabel**: It is kind of a text however it color is static and defined for another component.
- **IconLabel**: IconLabel has an image for it is component.
- **Symbol**: It is a kind of visual component. Symbols are separated into two: TerrainSymbol and AircraftSymbol.
- **AircraftSymbol**: This component is the aircraft symbol. A consistent aircraft symbol is used for an FDD.
- **TerrainSymbol**: Terrain symbols are used to show geographical elements and buildings such as mountains, tall buildings, airports etc.
- **Indicator**: Indicators are used to show some information, e.g. speed, fuel, temperature. There are two kinds of indicators Gauge and Bar.
- **Gauge**: Gauge indicators are like a speed indicator in a car.
- **Bar**: Bar indicators shows the information with a bar.

Domain Analysis/Domain Concepts Example

DSL Grammar

- EBNF Notation is used

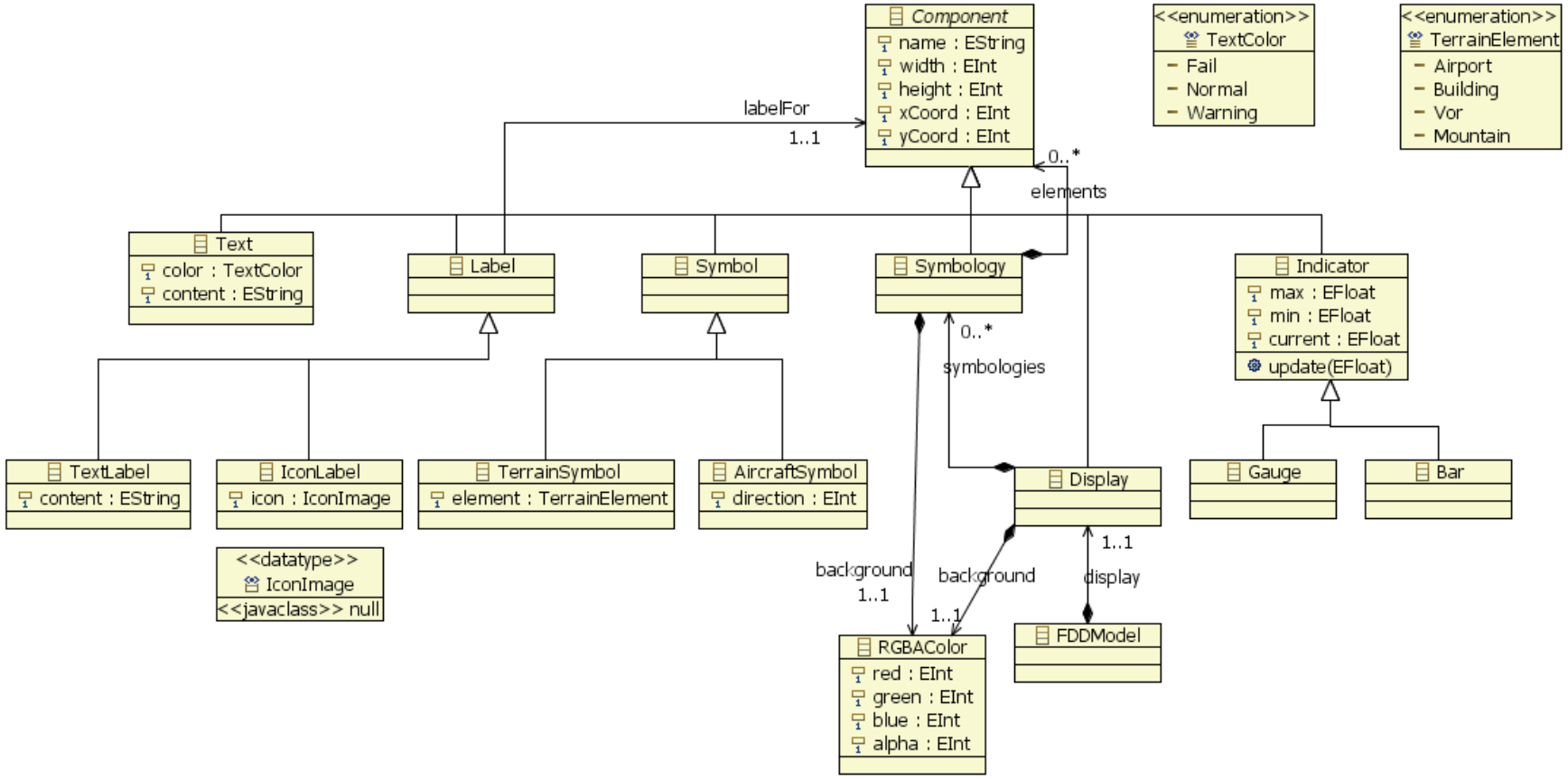
  FDDModel = Display;
  Display = {Symbology};
  Symbology = {Component};
  Component = Text | Label | Symbol | Symbology | Indicator;
  Label = TextLabel | IconLabel;
  Symbol = TerrainSymbol | AircraftSymbol;
  Indicator = Gauge | Bar;

  Terminals are: Gauge, Bar, Text, TextLabel, IconLabel, TerrainSymbol, AircraftSymbol

  Non-terminals are: FDDModel, Display, Symbology, Component, Label, Symbol, Indicator

Abstract Syntax of FDD

- Meta-model of FDD

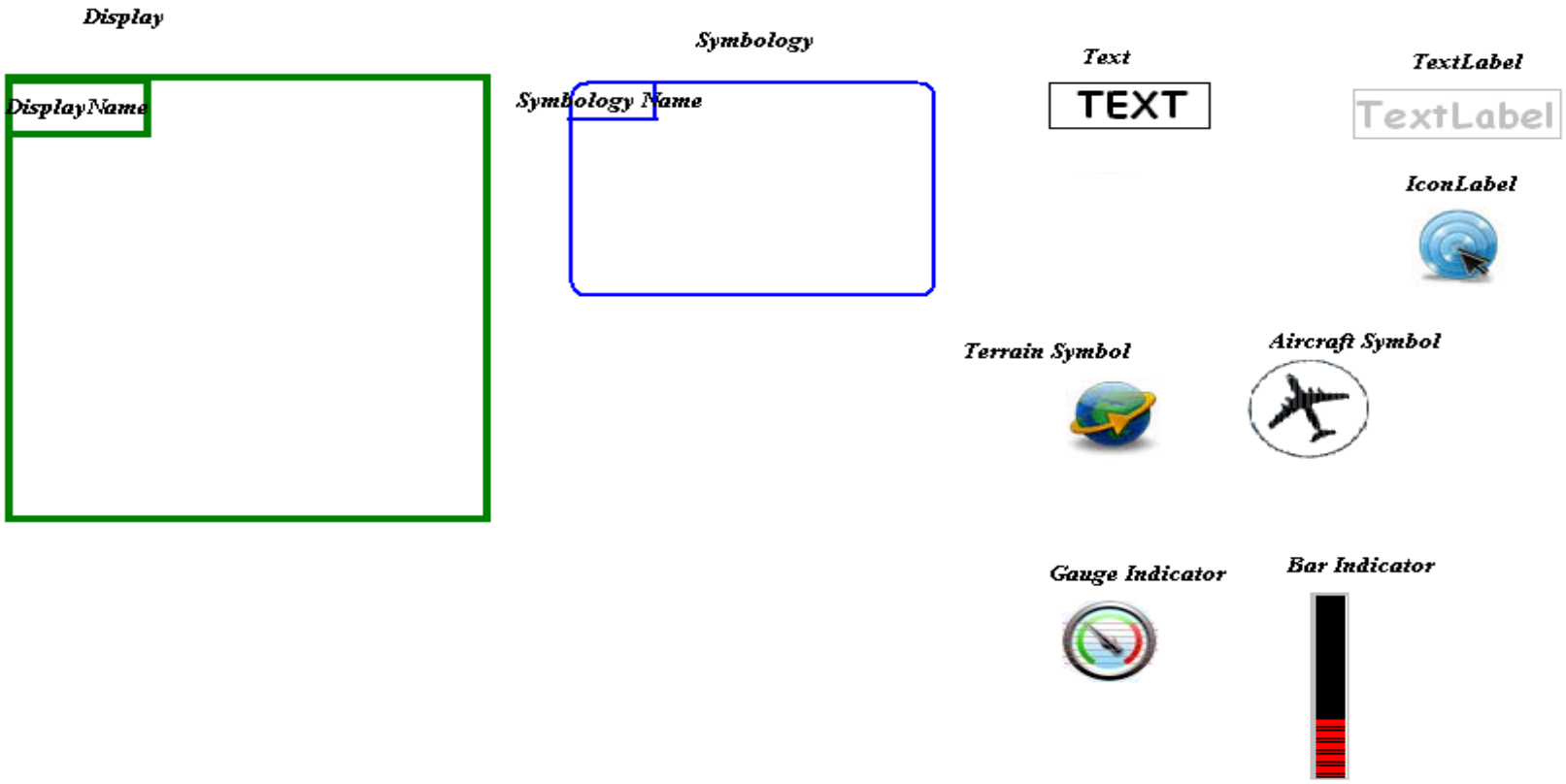


Meta-modeling from Scratch

- Used Tools
  - Eclipse IDE
  - oAW (openArchitectureWare)
    - ECore (for metamodeling)
    - Check Language (for static semantics)
- Ecore is simplified version of MOF
- More expressive than grammar

Concrete Syntax

- Example Concrete Syntax of meta-model



Static Semantics

- Used Notation: oAW Check Language, 15 rules
  Rules are used at code generation
  - **context** FDDModel **ERROR** "No Display Defined" :
    display != **null**;
  - **context** Symbology **ERROR** "All symbologies of Display have to be unique" :
    ((Display)**this**.eContainer).symbologies.select(e|e.name == **this**.name) == 1;
  - **context** Component **ERROR** "All elements of Symbology have to be unique" :
    ((Symbology)**this**.eContainer).elements.select(e|e.name == **this**.name) == 1;
  - **context** Display **ERROR** "Out of Width" :
    this.symbologies.exists(e|e.width<=**this**.width);
  - **context** Display **ERROR** "Out of Height" :
    this.symbologies.exists(e|e.height<=**this**.height);
  - **context** Indicator **ERROR** "Current Value is Out of Range" :
    this.current >= **this**.min && **this**.current <= **this**.max;
  - **context** Component **ERROR** "Invalid X-Y Coordinate" :
    this.xCoord >= 0 && **this**.yCoord >= 0;

Static Semantics

- **context** Symbology **ERROR** "Invalid X-Y Coordinate" :
  this.elements.exists(e|e.xCoord<=**this**.width) && **this**.elements.exists(e|e.yCoord<=**this**.height);

- **context** AircraftSymbol **ERROR** "Invalid Direction" :
  this.direction <= 360 && **this**.direction >= 0;

- **context** Label **ERROR** "Label has to be referenced to a Component" :
  this.labelFor != **null**;

- **context** Display **WARNING** "Background color of Display has to be more gray" :
  this.background.red <= 235 && **this**.background.green <= 235 && **this**.background.blue <= 235;

- **context** Symbology **ERROR** "Sybology cannot have element at Display or Symbology type" :
  this.elements.typeSelect(Display) == **false** && **this**.elements.typeSelect(Symbology) == **false**;

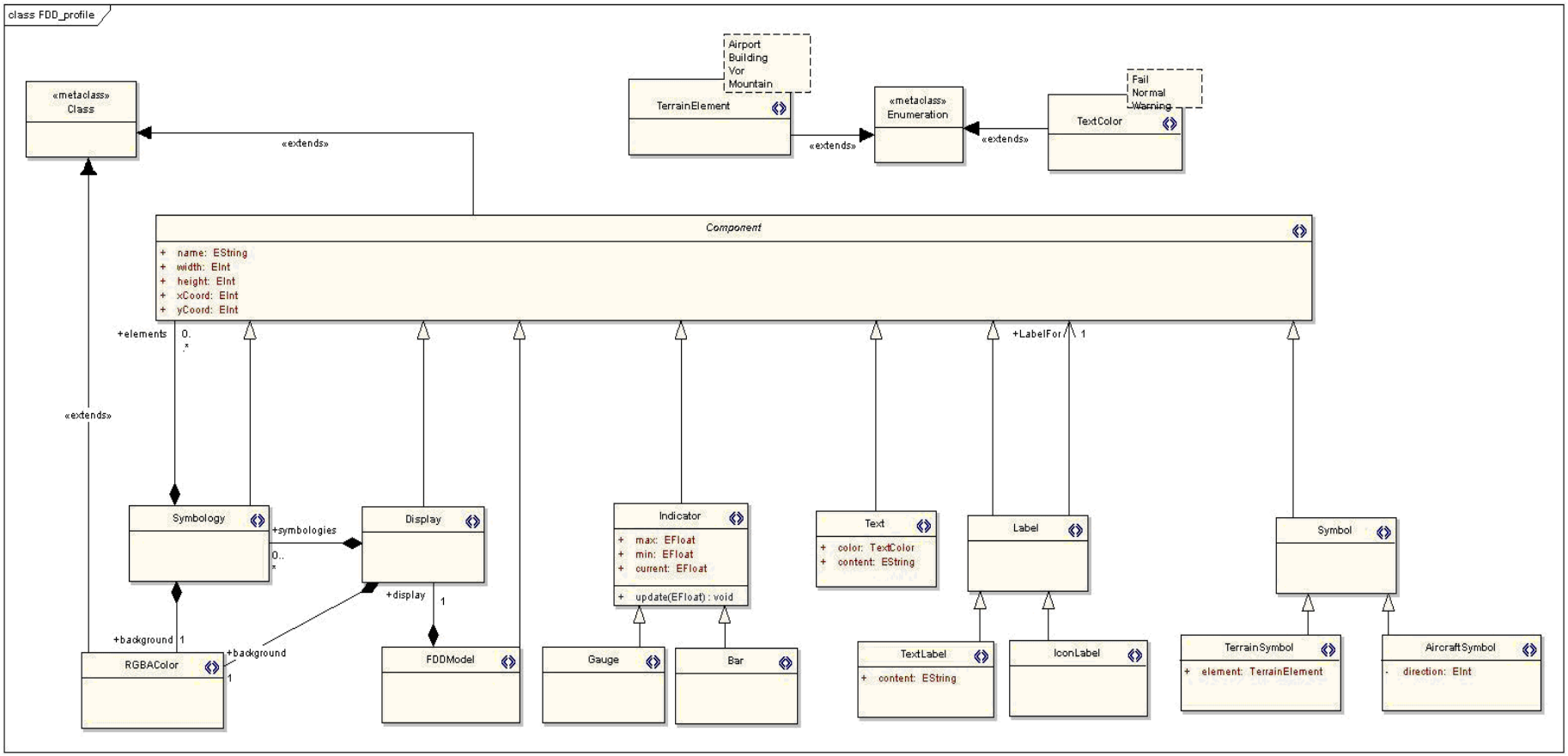
- **context** TextLabel **ERROR** "Text has to be defined for a TextLabel" :
  this.content != **null**;

- **context** IconLabel **ERROR** "Icon image has to be defined for a IconLabel" :
  this.icon != (IconImage)(**null**);

- **context** Component **ERROR** "Name has to defined" :
  this.name != **null**;

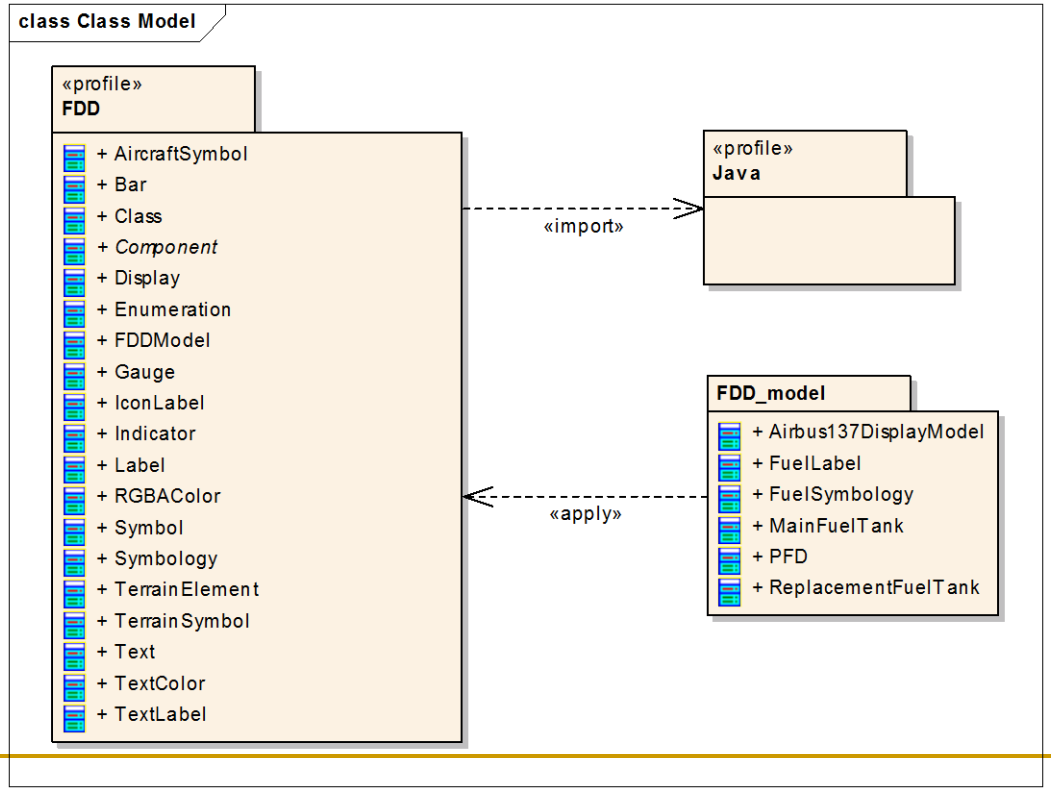
Metamodeling using UML 2.* Profiling

- Used Tool: Enterprise Architect



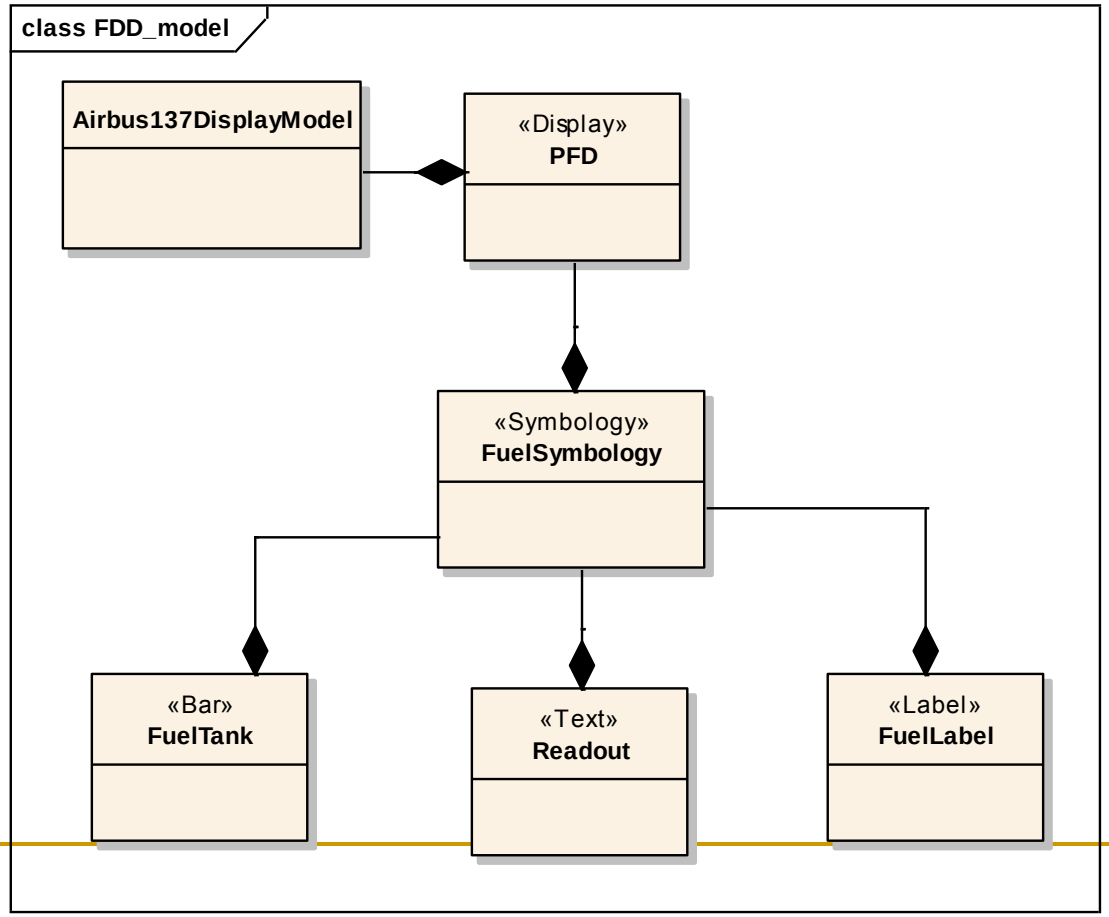
Metamodeling using UML 2.* Profiling

- Applying Developed UML Profile for FDD to Model



Metamodeling using UML 2.* Profiling

- Example Model from Developed Profile for FDD

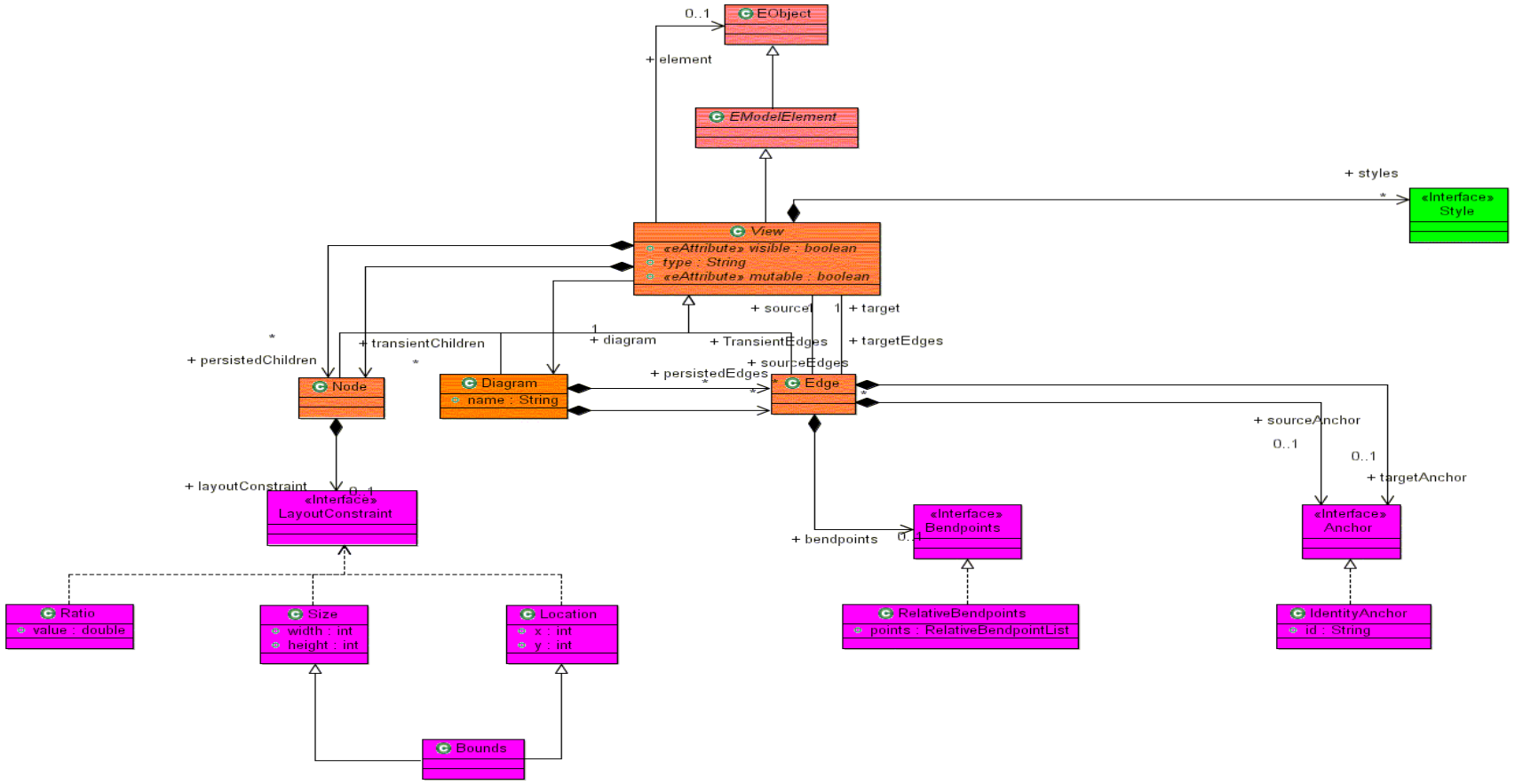


Model to Model Transformation: FDD to GMF

- ATL (Atlas Transformation Language) is used for model to model transformation
- Target model is chosen as GMF (Graphical Modeling Framework)
- Why GMF?
  - It is aimed to develop software through visual model with FDD Modeling
  - Generated tool for FDD Modeling will not be commonly used
  - GMF is framework for visualizing the models
  - GMF is commonly used

Model to Model Transformation: FDD to GMF

- GMF Core Notation Metamodel



Model to Model Transformation: FDD to GMF

- FDD is mapped to GMF as follows:
  - Display to Diagram
  - Component to Node
  - Also; Text, Labels, Indicators, Symbology, Symbol to Node
  - Size and Styles of FDD components are mapped to LayoutConstraint of GMF

Model to Model Transformation: FDD to GMF

- 5 Helper Functions & 6 Rules with 1 Abstract

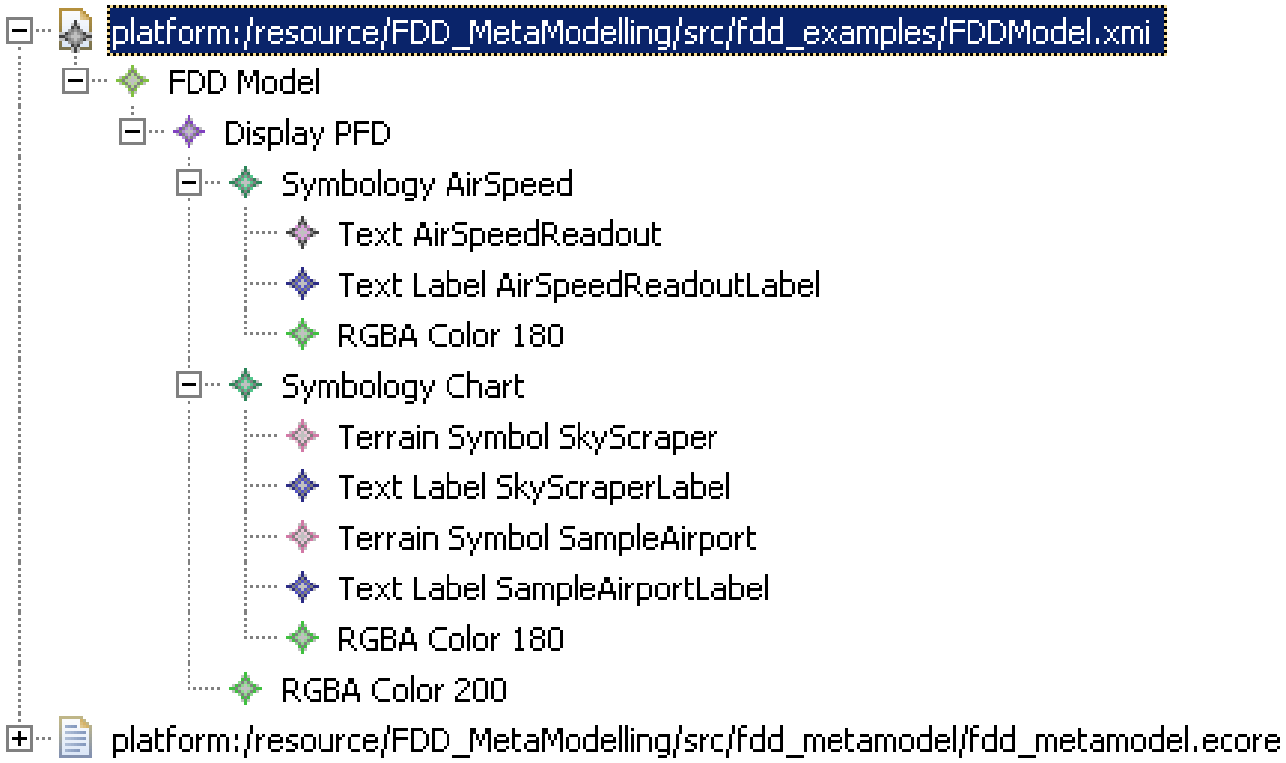
```
abstract rule Component2Node
{
   from
      component : FDD_metamodel!Component
   to
      node : GMF!Node
      (
         type <- component.name,
         layoutConstraint <- bound,
         visible <- true,
         mutable <- true
      ),
      bound : GMF!Bounds
      (
         x <- component.xCoord,
         y <- component.yCoord,
         width <- component.width,
         height <- component.height
      )
}
```

Model to Model Transformation: FDD to GMF

```
rule Display2Diagram
{
   from
      display : FDD_metamodel!Display
   to
      diagram : GMF!Diagram
      (
         name <- display.name,
         type <- 'FDD_Display',
         measurementUnit<-#Pixel,
         children <- display.symbologies,
         visible <- true,
         mutable <- false,
         styles <- style,
         layoutConstraint <- size
      ),
      style : GMF!ShapeStyle
      (
         fillcolor <- display.background
      ),
      size : GMF!Size
      (
         width <- display.width,
         height <- display.height
      )
}
```

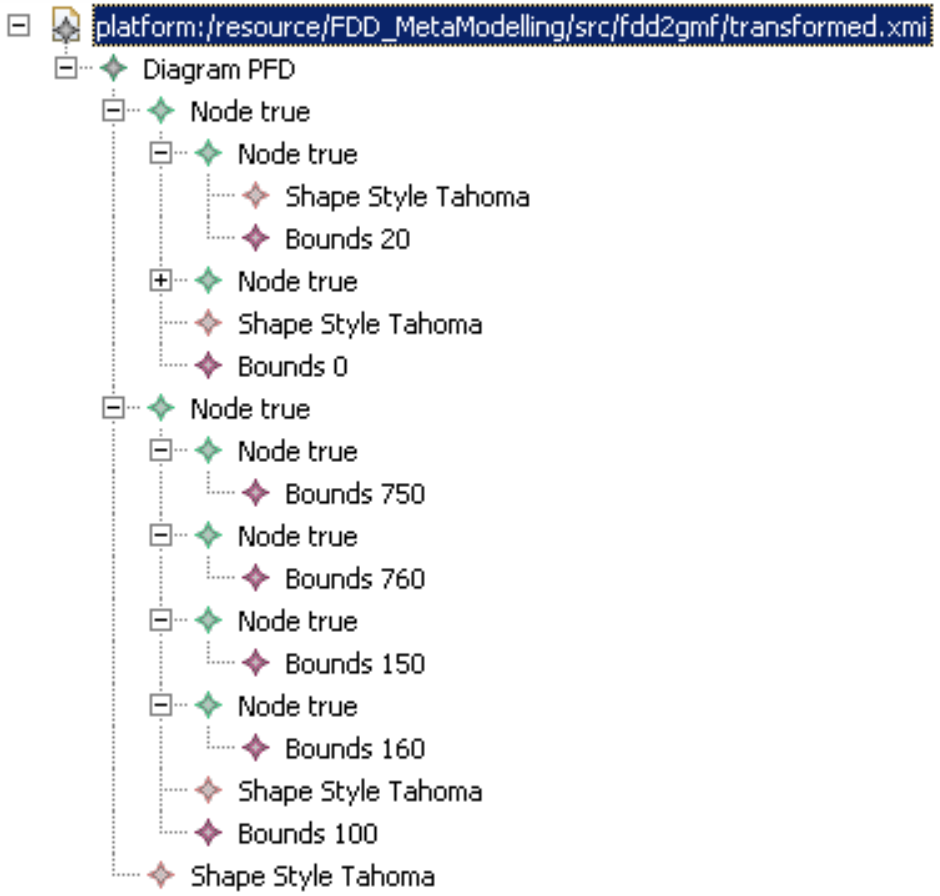
Model to Model Transformation: FDD to GMF

- Example model transformation: FDD Model



Model to Model Transformation: FDD to GMF

- Example model transformation: GMF Model



Code Generation (FDD Model to C++/OpenGL)

- Motivations of Code Generation
  - Reusable,
  - Certifiable,
  - High Quality code with FDD Modeling.
- Working Product after Design Phase
  - Cost effective: Design phase of software product has to be performed for avionics software products according DO178B standard

Code Generation (FDD Model to C++/OpenGL)

- Platform Specific vs Platform Independent Transformation
  - Platform specific text transformation technique since Khronos ES – SC OpenGL is widely used
  - C/C++ mostly used in embedded and real time systems
  - If OpenGL is replaced by another technology; the only thing to do is to develop platform specific rules for new technology

Code Generation (FDD Model to C++/OpenGL)

- Generated code segments call non-generated code contained in libraries
  - OpenGL APIs are called by generated code via library
- Xpand is used for model to text transformer
  - One of the most capable m2t language
  - Template based and easy to use

Code Generation (FDD Model to C++/OpenGL)

```
«IMPORT fdd_metamodel»
«EXTENSION fdd_template_m2t::GeneratorExtensions»
«DEFINE main FOR fdd_metamodel::FDDModel»
«FILE "FDDModel.cpp"»
int main() {
bool retVal = true;
Display «display.name» = new Display("«display.name»", «display.width», «display.height», (new RGBAColor(«display.background.red», «display.background.green», «display.background.blue», «display.background.alpha»)));
«FOREACH display.symbologies AS s»
        //Create «s.name» symbology
        Symbology «s.name» = new Symbology("«s.name»", «s.width», «s.height», «s.xCoord», «s.yCoord», (new RGBAColor(«s.background.red», «s.background.green», «s.background.blue», «s.background.alpha»)));
        «FOREACH display.symbologies.elements AS e»
  «REM»Create source file of used elements. Too long, not given«ENDREM»
        «ENDFOREACH»
        //Add «s.name» symbology to «display.name»
        «display.name».addSymbology(«s.name»);
«ENDFOREACH»
while (retVal == true) {
        retVal = «display.name».myCode(); }
return 0;
}
«ENDFILE»
«EXPAND display_cpp FOR display»
«EXPAND fddModel2code_classes::fdd_common»
«EXPAND fddModel2code_classes::fdd_symbology»
«ENDDEFINE»
```

Discussion: Used Tools

- oAW (openArchitectureWare)
  - Lots of bugs
    - Change at one view does not effect at other view of same model/component
  - Ecore and Check Language
    - Neither OCL nor MOF are fully supported
    - ECore and Check Language are supported
    - ECore and Check Language are easy to use
  - Not fully documented
    - No fully descriptive tutorials
    - No commonly used help file
  - Successful code template based generation
    - Xpand and Xtext

Discussion: Used Tools

- Enterprise Architect (EA)
  - Constraints at EA not compatible with other tools
  - It can be described via UML notes at diagram
- Vectorial Graphic Tools
  - It is not easy to use GMF
  - Easier and more practical way of defining concrete syntax
  - Need to produce a compatible tool in order to use generated concrete syntax while modeling
- ATL
  - More mature compared to other languages/tools
  - Complex to make an executable transformer

Conclusion/Lessons Learned

- Grammar is not good way to DSL
  - Constraint cannot be defined preciously
  - Visualization is not possible (especially for relations)
- Tools for MDSD are not interoperable
  - Constraints import/export between oAW and EA
- Differentiating M1 & M2 is difficult and critical issue
  - Classifying domain concepts
- Deciding on concrete syntax
  - No common and widely used symbols/notations

Conclusion/Lessons Learned

- Model to Model Transformation
  - Enables Interoperability
  - Effective way to use created domain specific models to commonly used tools
- Model to Text Transformation
  - It is aimed to generate reusable, certifiable, high quality code with FDD Modeling in this project
  - Nearly 100% percentage code generation with libraries
  - Productive way of developing code
  - Reusable: Platform specific parts can easily updated
  - Reduces certifications effort and costs at safety critical projects by using certifiable libraries, standards at code generator templates

Conclusion/Lessons Learned

- MDSD new approach but very promising
- MDSD is very suitable for FDD systems
  - Also suitable for embedded real time software at application level
  - Only one product will be updated with changes
  - Design will be alive during product life cycle
  - Visualization reduces complexity and decreases maintenance efforts (Easy to understand code)
  - Some constraints are made at metamodeling level
    - Reduces software defects

References

- **[1]** U.S. Department of Transportation Federal Aviation Administration, Advisory Circular 25-11A – Electronic Flight Deck Displays, USA, 6/21/07
- **[2]** RTCA, Inc., Do178B Software Considerations in Airborne Systems and Equipment Certification, USA
- **[3]** L-3 Avionics Systems, SmartDeck Brochure, Grand Rapids, MI USA, 2008
- **[4]** http://www.openarchitectureware.org, openArchitectureWare User Guide, Version 4.3.1, 17/04/2009
- **[5]** http://www.eclipse.org/modeling/gmf/, Eclipse Graphical Modeling Framework, Accessed at 17/04/2009
- **[6]** http://en.wikipedia.org/wiki/DO178B, DO178B, Accessed at 17/04/2009
- **[7]** http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.gmf.doc/prog-guide/runtime/Developer%20Guide%20to%20Diagram%20Runtime.html, Accessed at 16/05/2009
- **[8]** http://www.eclipse.org/m2m/atl/, Accessed at 16/05/2009
- **[9]** Architecture Board ORMSC, Model Driven Architecture (MDA), Document number ormsc/2001-07-01, 9/7/ 2001
- **[10]** http://www.omg.org/technology/documents/modeling_spec_catalog.htm, Accessed at 15/04/2009

THANK YOU

Questions?