Developing MDSD Model for Flight Deck Displays

Enver Veli ATABEK[1], Cevahir TURGUT[2]
[1,2]*Middle East Technical University, METU*
[1,2]*Ankara, TURKEY*
[1]*20393177@mail.baskent.edu.tr,* [2]*cevahir.turgut@gmail.com*

Abstract

Principal infrastructure of model driven software development (MDSD) for flight deck displays is implemented by creating metamodels using MOF-from scratch and profiling at UML 2 with the static semantics of domain which is flight deck display system. Grammar for flight deck display system is also written. This paper is written to explain implementation details and to give comparisons of used method. Also difficulties which are experimented at development processes are given.

Keywords

MDSD, metamodel, metamodeling, MOF, UML, profiling, grammar, EBNF, Flight Deck Display, modeling, DSL, static semantics [10]

1. Introduction

Today's new aircrafts are equipped with glass cockpit systems. Glass cockpit term means the interaction are done via a display system instead of old manual switches and indicators. Flight Deck Displays (FDDs) which are the display systems for aircrafts (i.e. helicopters, airplanes etc) are the main display for the pilot. Interactions are done via glass cockpit systems. Pilots interfere with the aircraft with the control mechanisms on the glass cockpit systems.

A model-driven software approach is developed for Flight Deck Displays in this project; since it is suitable to visualize components of FDDs using diagrams, it is a cost effective way to use code generator for FDDs software.

Components of FDD are visual components; so software of FDDs can be performed easily. Developing software for FDD systems using visual components can increase productivity, reduce development and maintenance costs.

Code generators are cost effective because life-span of aviation products is very long. As an example, there are cargo airplanes which are produced before 40 years and it is aimed to use these airplanes for more than few decades. Used technologies change with time. Before few decades ADA was popular but now C++ are popular; it is not clear that after few decades, which programming language will be popular. After few decades, it can be very easy to adapt new technology by changing code generator template.

Another issue of FDD Modeling in our project is that defect at the FDD system can cause dead; this means that FDDs are safety critical systems. A critical issue for the aircraft systems is that ever piece of code must be certifiable. Certifications are done by different authorities. For example; civil aviation authority of the USA is the Federal Aviation Administration (FAA), Civil Aviation Safety Authority (CASA) is the Australian federal agency, and European Aviation Safety Agency (EASA) is an agency of the European Union (EU). International coordination of these authorities is done by International Civil Aviation Organization (ICAO). These authorities – FAA and EASA are responsible for the certifications of civil aviation and they determine regulations & policies for avionics. Standards for avionics software are strict. For the display system, a certifiable library is preferable by FAA. Khronos ES – SC 1.0 is a certifiable OpenGL subset and most of the FDDs are developed with that library. It is aimed to generate code from developed FDD model which conforms to Khronos ES – SC 1.0 with our project. It is also aimed to generate a compatible code with DO178B (Software Considerations in Airborne Systems and Equipment Certification) which is a guidance for software development published by RTCA, Incorporated. The standard was developed by RTCA and EUROCAE. The FAA accepts use of DO-178B as a means of certifying software in avionics [1, 2, 6].

The paper is organized as follows: Modeling of FDD (domain analysis of FDD, defining domain concepts, etc.) is given in Section 2. Section 3 follows with mapping of domain concepts to grammar. Section 4 illustrates definition of FDD metamodel based on MOF-from scratch and Section 5 explains static semantics of our FDD metamodel. An alternative metamodel using UML Profiling is given at Section 6. Section 7 gives model to model transformation which is applied from FDD model to GMF model and associated rules. Section 8 follows with model to text transformation. Finally, Section 9 discusses development process of FDD Modeling and Section 10 concludes by giving important points of FDD Modeling.

2. Flight Deck Display Modeling

FDD systems are suitable for model-driven software development because models are very expressive while developing display systems. Especially, visual modeling facilitates development activities by demonstrating system at the start of development process. For example, textual requirements and design are less expressive than visual ones. Also it is easy to go though model to code.

Based on the authors' profession experiences and knowledge on avionics domain, some guidelines, some

aircraft documents are reviewed. In addition to company restricted documents at the first domain selection process, FAA documents are reviewed. As a result of review process, 25-11A (ELECTRONIC FLIGHT DECK DISPLAYS) document is selected as a base document for our domain analysis process. This document provides guidance for the design, installation, integration, and approval of electronic flight deck displays, components, and systems installed in transport category airplane [3].

There are many components of FDDs. However, it has been chosen a subset for the domain in scope of this project. Chosen components are the main components of flight deck display systems. Also selection is done in order to not restrict developers while performing development process based on our meta-model. Resulted domain model include instances of the following components: "display, symbology, text, label, symbol, indicator". Glossary of resulted domain is given at TABLE I.

TABLE I. DOMAIN CONCEPTS OF FLIGHT DECK DISPLAY MODELING

Terms	Description
Display	The main scene. A display contains symbologies.
Symbology	A place holder that groups the components.
Text	Texts. Usually used to display warnings, messages and errors. There are three kinds of texts; Warnings, Normal, Errors.
Label	Label is a definitive component for another component. Labels are seperated 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 seperated 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 buildinds 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.

3. DSL Grammar

EBNF notation is used to express resulted domain while mapping domain concepts to grammar. FDDModel can be Display as seen from grammar of FDD. Display includes multiple symbologies and symbology includes multiple components. Component can be Text, Label, Symbol, Symbology or Indicator since it is like a base class of these elements. In the same manner; Label can be TextLabel or Icon, Symbol can be TerrainSymbol or IconLabel, Indicator can be Gauge or Bar.

3.1. EBNF Notation of DSL of FDD

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

4. Definition of metamodel based on MOF-from scratch

4.1. Developed Metamdeol Based From Scratch

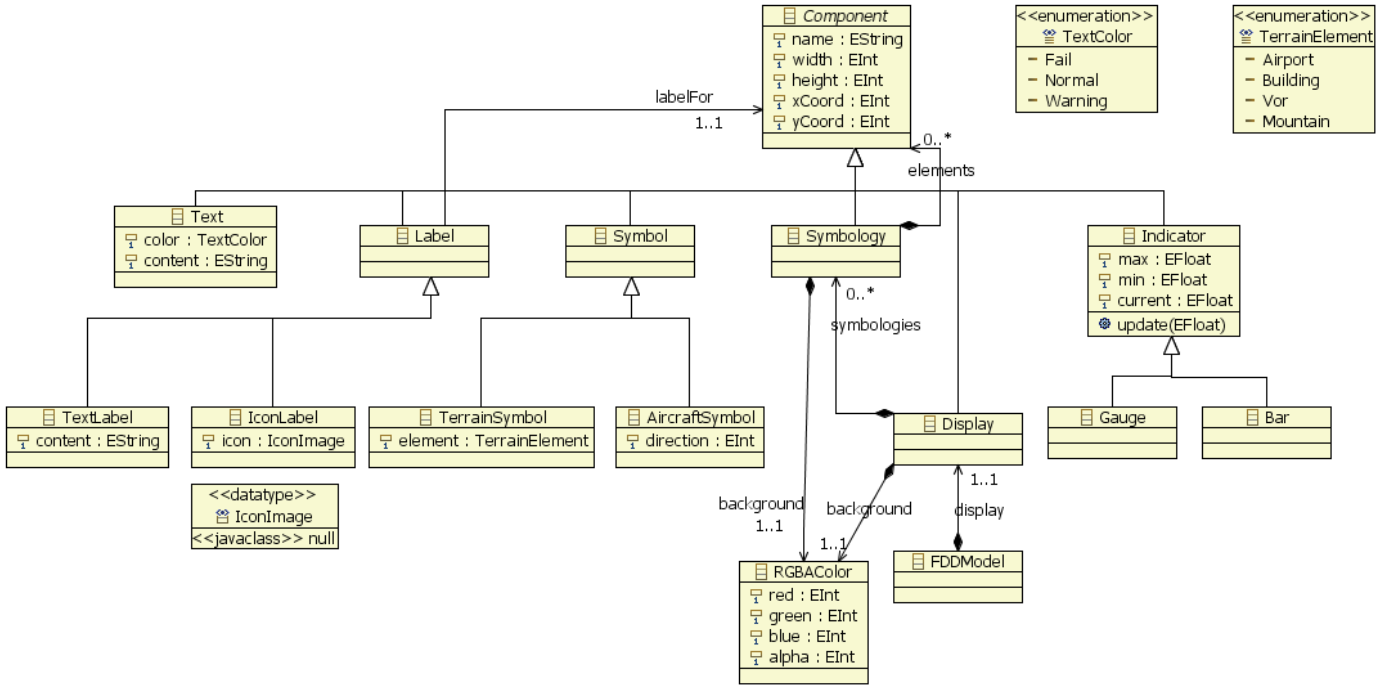

Figure 1. Metamodel Using ECore Meta-metamodel

The meta-model from scratch is illustrated in Figure 1. Component item is an abstraction for our domain concepts. Common attributes like name of component, width, height, X-axis and Y-axis positions are inherited from Component item. Text item has color and content attributes. Type of color attribute of Text item is TextColor and it can express Fail, Normal or Warning conditions. Text item is also an abstraction, in the M1 level there won't be an instance of Text. There are two special types of Label item. One of them is TextLabel which has content as an attribute and other is IconLabel which has an icon at IconImage type as an attribute. Both kinds of Label items must be associated with a component item. Since Label item is a description for an object. There is a Symbol item to represent general graphics and there are two special types for this item. TerrainSymbol is the first special type of Symbol which has element at TerrainElement type as an attribute. TerranElements can be airport, building, vor and mountain in general. AircraftSymbol is another special type of Symbol which has direction as an attribute to express direction of aircraft. In the same manner with Text item Label and Symbol items are abstractions. Symbology and Display items have background attribute to store background color of these items. Type of background attributes is RGBAColor which has red, green, blue and alpha components. Indicator item has max, min and current attributes to represent maximum, minimum and current values of the indicated object, respectively. Also Indicator item has an operation named update to update current value of indicated object. Gauge and Bar items are the special types of Indicator item. Indicator item is an abstraction and won't be instantiated in M1 level. Symbologies consist of elements which are at type Component but symbologies cannot include Display and Symbology items as an element. This is expressed at the static semantics of our mete-model. In the same manner, Display item consists of symbologies and it cannot include Display item as an included symbologies. Our FDDModel is formed with a Display.

Eclipse IDE and oAW (openArchitectureWare) framework are used as the development environment. ECore is used to construct our meta-model. ECore is simplified version of MOF.

4.2. Example Concrete Syntax for Metamodel based on MOF from Scratch

An example concrete syntax notation which is given at Figure 2. is created using a Vectorial graphics tool. There can be lots of concrete syntax for developed meta-model for FDD since one can define any of the components given in metamodel with any visual element. In example concrete syntax, Aircraft is defined with airplane image, but it can be defined with any other visual element like “clouds”. Given example is just one of the possible concrete syntaxes.

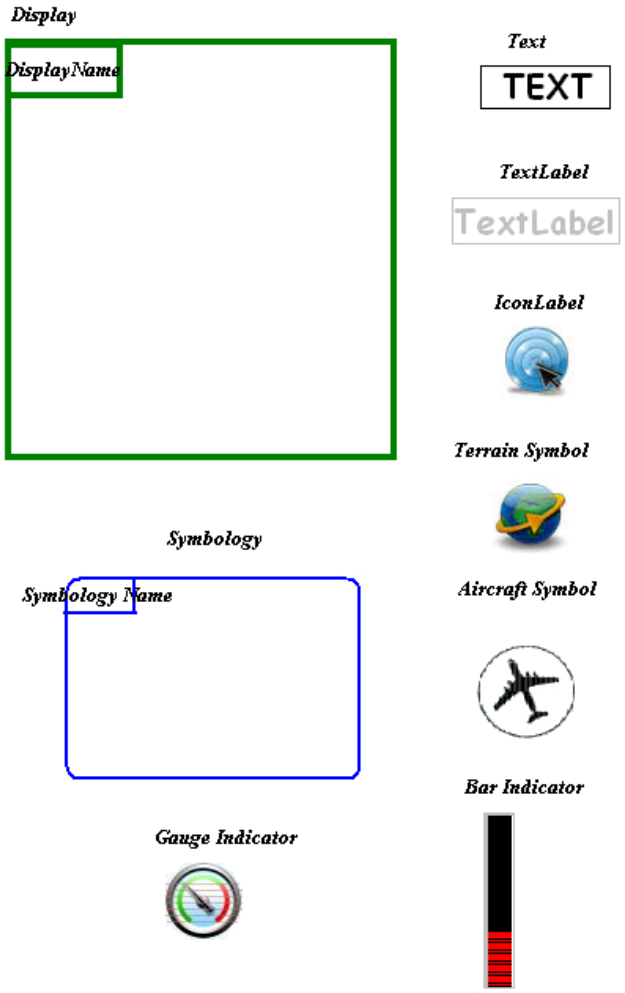

Figure 2. Example Concrete Syntax for Metamodel from MOF

5. Static Semantics

15 (Fifteen) constraints are defined for resulted meta-model. They are specified using Check Language which is provided by oAW (openArchitectureWare) tool. Check Language is a syntactical mixture of Java and OCL (Object Constraint Language) [4]. Statics semantics of our meta-model is given at TABLE II. These constraints are very useful while checking validity of developed model based on meta-model and also they will be used while code generation.

FDDModel has to have a valid Display according to first rule. Second and third rules requires that all symbologies of Display and all elements of Symbology have to be unique; respectively. None of the symbology has width which is greater than display width according to fourth rules. Fifth rule is same as fourth but it is for height constraints. Also none of the component can have negative x-axis or y-axis position according to seventh rule.

TABLE II. STATIC SEMANTICS OF METAMODEL

```
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;

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;
```

6. Alternative metamodel using UML profiling

UML Profiling provides a generic extension mechanism for customizing UML models for particular domains and platforms. In our project, FDD profile is created using UML 2.* Profiling according to our FDD meta-model.

While creating FDD profile, Component and RGBAColor items are extended from class type of metaclass of UML meta-model. Component is generalization of Symbology, Display, FDDModel, Indicator and its specializations, Text, Label and its specializations, Symbol and its specializations. TerrainElement and TextColor items are extended from enumeration type of metaclass of UML meta-model. Resulted UML 2.* profile for our meta-class is given at Figure 5. Also generated XMI code for FDD profile is interoperable with other modeling tools.

Example usage of developed UML profile is shown at Figure 3. and Figure 4. . In Figure 3. , it is shown that FDD Profile imports Java Profile and FDD Profile is applied to FDD_model which is an example model of FDD Profile. FDD_model is an example for PFD (primary flight display) display for sample aircraft. Fuel symbology of PFD which has bar type indicator for FuelTank, readouts and TextLabel type label is shown in Figure 4.

EA (Enterprise Architect) has been used while creating UML profile for FDD system. EA uses stereotype “extends” for stereotype symbol. UML concrete syntax is used for this meta-model.

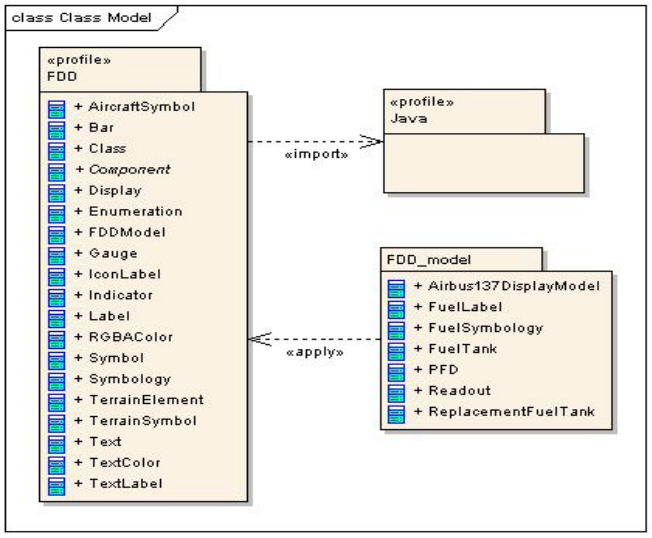

Figure 3. Applying Developed UML Profile for FDD to Model

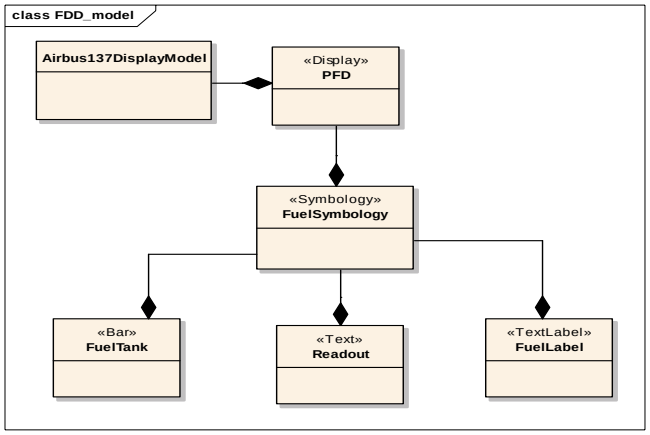

Figure 4. Example Model from Developed Profile for FDD

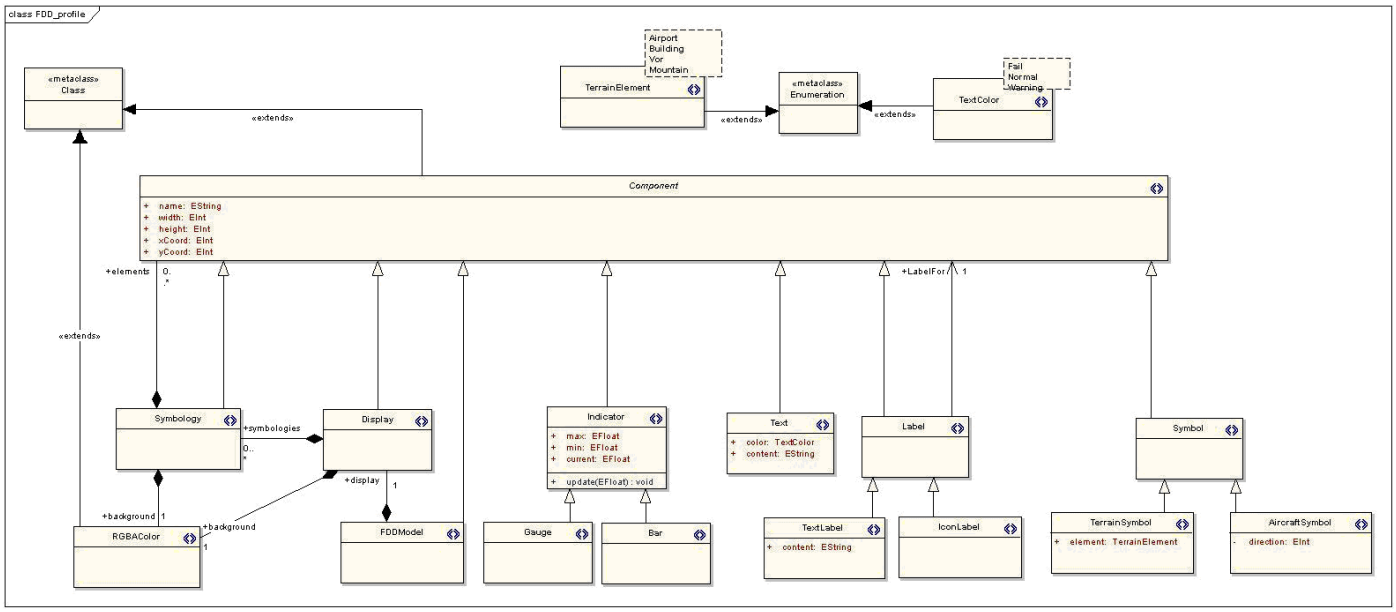

Figure 5. UML Progile for FDD

7. Model to Model Transformation: FDD to GMF

In this project, it is aimed that FDD Modeling is based on visualization of models. Although metamodel is constructed and example concrete syntax is selected, tool for generating models using our metamodel and concrete syntax is not developed due to time constraints. We are assuming that we have developed such a tool for FDD Modeling for this section. In this case, another problem - interoperability problem arises. Generated models with our tool cannot be used at any other tools (Also note that, any UML 2.* compatible tool can be used if modeling is done using FDD Profile). For example, one can request to open model developed using our tool at The Eclipse Graphical Modeling Framework (GMF). So model to model transformation is need to use generated model at other modeling tools.

GMF is commonly used framework for visualizing the models. Custom graphical editors based on metamodels defined via EMF (Eclipse Modeling Framework) can be achieved using GMF. Ecore is provided by EMF for defining metamodels using base components of MOF. FDD metamodel is also defined using Ecore. Also in our assumption, developed tools provide visualizing for FDD metamodel using pre-defined concrete syntax same as GMF. So GMF is very suitable example framework for requesting to use developed models using our tool. Model to model transformation from FDD to GMF is defined in the scope of this project using Atlas Transformation Language (ATL). ATL provides to define transformation rules to transform model which conforms source metamodel to another model which conforms to target metamodel. We have defined FDD metamodel, generated example model and GMF metamodel is already defined and being used commonly. FDD components are mapped to core hierarchy of the GMF notation meta-model [7] components using defined ATL rules [8]. Using defined rules, example model is transformed to GMF model. Format of our example model and transformed model are in XMI format.

ATL is used as a model to model transformation language since standard development tools like syntax highlighting, ATL compiler, debugger, etc. are provided by The ATL Integrated Environment [8]. Also necessary documentation of ATL is available. Header and helpers of our ATL file has been given in TABLE III. “FDD_metamodel” is the name of FDD metamodel and “aFDD” is the name of example FDD model which is given in Figure 6. ATL Helpers are method like structures of ATL. They are defined for commonly used operations like color conversions. Main transformation rules are also given in TABLE IV. When ATL transformation is applied to example model using FDD metamodel and GMF metamodel, model which is given in Figure 7. is generated. FDD is mapped to GMF as follows: “Display to Diagram; Component to Node”. Since Text, Labels, Indicators, Symbology, Symbol are specialization of Component; these elements are also mapped to Node. Component to Node rule is defined abstract in order to achieve mentioned transformation. For example Symbology rule extends from Component2Node rule and additional transformation constraints are defined in Symbology2Node rule. For example background color of symbology is transformed to style of Node element using “colorvalue” helper. Connections have not been defined at GMF side since there are no visual elements developed.

TABLE III. ATL TRANSFORMATION HELPERS FOR FDD TO GMF

```
module FDD2GMF; -- Module Template
create aGMF : GMF from aFDD :
FDD_metamodel;

helper context
FDD_metamodel!RGBAColor def :
colorvalue : Integer =
      (self.red*255 + self.green*255
+ self.blue*255 + self.alpha*255);

helper context
FDD_metamodel!TextColor def :
normalColor : FDD_metamodel!RGBAColor
=
      Sequence{128, 128, 128, 128};

helper context
FDD_metamodel!TextColor def :
failColor : FDD_metamodel!RGBAColor =
      Sequence{255, 0, 0, 0};

helper context
FDD_metamodel!TextColor def :
warningColor :
FDD_metamodel!RGBAColor =
      Sequence{255, 255, 0, 0};

helper context
FDD_metamodel!TextColor def:
getRGBAColorFromTextColor() :
FDD_metamodel!RGBAColor =
      if self.color =
FDD_metamodel!TextColor.Normal
      then
         self.normalColor
      else
            if self.color =
FDD_metamodel!TextColor.Fail
            then
               self.failColor
            else
               self.warningColor
            endif
    endif;
```

TABLE IV. ATL TRANSFORMATION RULES FOR 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 )
}

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 )
}

rule Symbology2Node extends
Component2Node
{
from
   symbology :
FDD_metamodel!Symbology
to
   node : GMF!Node (
     styles <- style,
     children <- symbology.elements
),
  style : GMF!ShapeStyle (
     fillcolor <-
symbology.background.colorvalue )
}
```

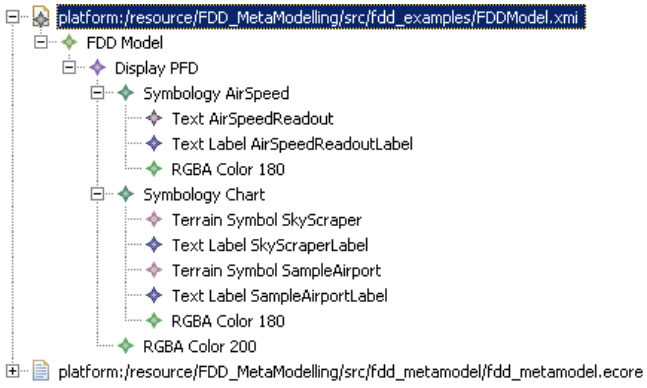

Figure 6. Example FDD Model

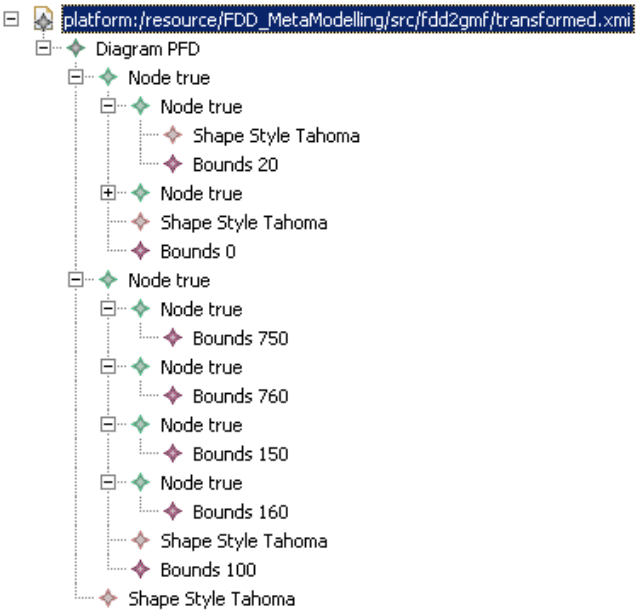

Figure 7. Generated GMF Model

8. Model to Text Transformatıon

It is aimed to generate reusable, certifiable, high quality code with FDD Modeling in this project. Certification costs of avionics software products are very high since every statement and every condition of code that will be execute on aircraft has to be verified (tested, analyzed). This procedure has to be performed for any modifications on code. Generating code from model is very productive and cost effective way for two reasons. First, once transformation template is written, high percentage of code can be generated automatically from developed models. Design phase of software product has to be performed for avionics software products according DO178B standard. Mostly, designs of software are done using UML or UML like tools. Once code generator can be certified cost of software development will be reduced and code phase will be passed very fast since very high percentage of code can be generated after design phase using transformation tool. Second, maintenance costs

will be reduced by reducing software life cycle phases. This is especially valid at high mature companies since they obey robust and expensive processes. By modifying model, code changes are automatically done.

There are two code generation techniques. Platform Specific and Platform Independent transformations can be performed using model to text transformation techniques. We have selected platform specific text transformation technique since Khronos ES – SC OpenGL is widely used graphic library at flight deck systems as many embedded safety-critical systems. Also after a few decades, OpenGL can be replaced by another technology; the only thing to do is to develop platform specific rules for new technology. This seems extra cost but since there is only one widely used platform, platform specific transformation is seemed effective engineering practice. If there were two widely used platforms, then platform independent transformation definition would be required; but it would also required that there are two platform specific transformation definitions for independent platform definition.

Xpand is one of the most capable models to text transformation language. We have used Xpand for platform specific (C++, Khronos ES – SC OpenGL) model to text transformation. Our Xpand template definition for main application of flight deck display model is given in TABLE V. This definition generated application code of input model.

There is “Component” in FDD metamodel as a base element for other elements except data elements like “RGBAColor”. In our code generation, common data types are defined like type definitions, RGBAColor class definition, Component class definition to generate their codes. Classes which are inherited from Component are generated whenever they are needed according to input model. This is first Display source file is created and it is instantiated in main method according model; then if display has symbologies, Symbology class source file is created. Then Symbology is instantiated and added to symbologies of display object. According to FDD metamodel, Symbology has elements which are inherited from Component. Type of Symbology element is checked and its source file is created then created class is instantiated and added to elements of Symbology object according to the input. This process is performed for all of elements of Symbologies.

There is an abstract “myCode” method at Component class. The derived classes implements “myCode” method by performing its related responsible jobs like necessary OpenGL API calls. If a class has sub items; for example Display class has symbologies as sub items, after performing its job, then it calls “myCode” of its sub items. So that execution of system is performed by periodically calling “myCode” method of display object, then it calls its sub items’ “myCode” in a nested manner.

Finally, generated code segments call non-generated code contained in libraries like OpenGL library in developed model to code transformation.

9. Discussion

During the project, lots of problem occurred but also MDSD importance is also experienced. The benefits of MDSD are examined. They are given at below according to subtopics.

9.1. Tools

During the project, it was experienced that some obstacles were aroused from used tools. Especially oAW has lots of bug and it decreases productivity. Model – graphical model compatibility may be lost during changes. These immature tools force you to solve some problems and waste some of your time. Also, our major concern is that oAW does not support standards. It means that nor ECore neither Check Language is a standard. We expected to use MOF and OCL, however oAW framework does not support. Another major issue is that these tools are not well documented. But visual interface to develop ECore is very effective after learning tool.

For profiling we used Enterprise Architect, easy to use but it has problems with import and export. For example, EA is not interoperability with other tools like oAW while exporting constraints to oAW. But EA generates XMI output very easily and it can be easily used at other tools.

ATL is flexible language for defining transformation rules. But there are some bugs while working at ATL Eclipse environment. For example; we had problems while compiling ATL file. ATL files are compiles automatically when ATL file is saved. But when we faced this problem, we had to construct ATL project again in order to compile ATL files. oAW Xpand is also very flexible language to perform model to text transformation.

9.2. Grammar

Grammar usage is not an efficient and suitable for defining a Domain Specific Language. EBNF notation is more suitable for solution (i.e. a programming language) domain; it is not for problem domain (i.e. for model domain). Constraints and relations cannot be expressed clearly. Also it is open to ambiguity. A more expressive meta-syntax can be constructed for MDSD approach instead of EBNF.

9.3. Meta-modeling from Scratch

Meta-modeling from scratch is easier than defining the grammar. In fact, in our project we firstly create our meta-model and then construct grammar. Meta-modeling is much more expressive. Internal types and abstractions are used. Constraints and relationships are clearly defined. Also meta-modeling process provides you to distinct M1 and M2 items. In our opinion meta-modeling is the best way for MDSD approach.

TABLE V. MAIN CODE GENERATION XPAND DEFINITION

```
«IMPORT fdd_metamodel»
«EXTENSION fdd_template_m2t::GeneratorExtensions»

«DEFINE main FOR fdd_metamodel::FDDModel»
«FILE "FDDModel.cpp"»
#include"Display.h"
#include <iostream>
using namespace std;

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»
		//Create «e.name» element
		Component «e.name» = new
«e.metaType.toString().subString(15,e.metaType.toString().length)»("«e.name»",
«e.width», «e.height», «e.xCoord», «e.yCoord»);
		//Add «e.name» symbology to «s.name»
		«s.name».addElement(«e.name»);

		«IF e.metaType.toString().subString
(15,e.metaType.toString().length).matches("Indicator")»
			«EXPAND fddModel2code_classes::Indicator»
		«ENDIF»
		«IF e.metaType.toString().subString
(15,e.metaType.toString().length).matches("Gauge")»
			«EXPAND fddModel2code_classes::Gauge»
			«EXPAND fddModel2code_classes::Indicator»
		«ENDIF»
  «REM»Comment: Other elements can be transformed with similar rules«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»
```

9.4. Developing Concrete Syntax for Meta-model from Scratch

A generative component and runtime infrastructure for developing graphical editors based on EMF (Eclipse Modeling Framework) and GEF (Graphical Editing Framework) are provided by The Eclipse Graphical Modeling Framework (GMF) [5]. We realized that developing a graphical editing surface for a particular domain (in this project our domain, FDD systems) by GMF (Graphical Modeling Framework) wastes too much time. Then we have created example concrete syntax for components of FDD domain using graphics program.

9.5. UML 2.* Profiling

UML 2.* Profiling is the most effective and productive way to define meta-model with its concrete syntax if it is not requested to create new concrete syntax which is more expressive.

9.6. MDSD

MDSD is an effective approach and increases productivity. It focuses on problem domain and produces a solution for the specific domain. Once a domain specific meta-model is constructed than model for the problem domain easily be generated. As a main point of MDSD, model is executable so that productivity is increased. Also, MDSD increases re-usability. Since re-usable items are defined with use of meta-modeling.

9.7. Model to Model Generation

Model to model transformation is necessary when metamodel from scratch is used as metamodeling technique. Since, developed models will not be interoperable with commonly used tools like UML, GMF tools.

9.8. Model to Text Transformation - Code Generation

Code generation phase is like composition of design and coding phases of standard software engineering. Xpand is very effective template based text generator. Various generation rules can be defined like string operations on file, variable names based on model elements. Also checks can be made in Xpand and it is possible to decide which files can be generated according to input model

10. Conclusion and Future Work

In our case, we aim to construct a tool that generates executable from our model that conforms to create meta-model. In the avionics, as stated before, a software part must be certifiable. However, if you certify the tool that generates the code you do not need to certify your code or certification costs of developed software reduces. That means lower cost.

We constructed a platform specific code generation template and we have successes nearly 100% percentage of code generation from our models which conforms to our FDD metamodel. We have used libraries like OpenGL as a manual code. But in this case, when it is requested to use another platform like The Microsoft DirectX® instead of OpenGL for graphics library, platform specific parts of code generation templates have to be updates. There is a trade of between using platform specific and platform independent text transformation.

A tool for generating models using our metamodel with any concrete syntax can be developed. This can be achieved by developing a plug-in for Eclipse or writing application software for this purpose. Model creation, model to model transformations and model to text transformation can be done using mentioned tool. Using this tool, abstraction level will be increased and complexity problem of software domain can be decreased. Even there is no such a tool, models can be developed using Ecore Model Editor. Sample model is developed using mentioned editor and it is seen that model are generated very effective. So, after generating model, code generation is just a few seconds using developed model to text transformation definitions. Usage of these techniques can be used at avionics which is live domain since lifetimes of aircrafts are very long.

Generated code can be optimized in order to achieve quality and safety constraints since developed code generation rules are written to illustrate how effectively and how flexible that code can be generated. Also for the generated code segments, a text generator template for automatic test cases can be developed.

Finally, it is seen that software for flight deck display systems are suitable domain for model-driven software development in our practice of FDD Modeling. Model-driven software development approach can be seen as investment since at the first steps (domain analysis, metamodeling) of this approach has costs. But costs of software life cycle processes after design phase are reduced significantly after constructing modeling environment of selected domain. MDSD approach for flight deck displays or any other selected safety-critical systems can be more productive and cost effective if MDSD is applied as a whole in effective way like also developing automatic test case generator.

11. Acknowledgments

We want to thank Asst. Prof. Bedir Tekinerdogan for his precious advices and contributions during this study.

12. References

[1] RTCA, Inc., Do178B Software Considerations in Airborne Systems and Equipment Certification, USA

[2] U.S. Department of Transportation Federal Aviation Administration, Advisory Circular 20- 115B – RTCA, Inc., Document RTCA/DO-I 78B, USA, l/ll/93

[3] U.S. Department of Transportation Federal Aviation Administration, Advisory Circular 25-11A – Electronic Flight Deck Displays, USA, 6/21/07

[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