User Manual

For

Vehicular Architectural Blended Modeling in EAST-ADL Supporting Software Product Lines

(Variability Resolution for Timing Constraints - Version 1)

1. Introduction

With the advent of modern vehicular technologies, the race for manufacturing new vehicle models with advanced functionalities has increased exponentially during the last few years. Each vehicle model is usually tailored into different sub-versions - referred as variants - based on different preferences like culture and cost. Customizing a vehicle model into different variants requires several variations in hardware and software components. This severely affects productivity and time-to-market objectives, as implementing each variant separately without promoting reusability is costly. To manage this, Product Line Architectures (PLAs) showed promising results in automotive systems.

EAST-ADL (Electronics Architecture and Software Technology – Architecture Description Language) was first introduced in the ITEA EAST-EEA project. It is a Domain Specific Modeling Language (DSML) that allows the development of a complete system model through four levels of abstraction i.e., Vehicle, Analysis, Design and Implementation level. Moreover, it offers other extensions like timing and variability to model timing constraints and variability configurations, respectively, at different levels of abstraction. Furthermore, it accommodates both software and hardware variabilities simultaneously which is a big plus. EAST-ADL offers advanced support for PLAs where variability can be defined between four levels of abstraction and also within each level of abstraction. At one end, the provision of immense variation points in EAST-ADL caters for the development of large and complex PLAs; on the other end, variability resolution becomes problematic as configuration decisions increase exponentially. This is especially true while dealing with timing constraints associated with variant-specific hardware and software components.

In BUMBLE and MoDeV projects, we are motivated to boost the Verification and Validation (V&V) activities of EAST-ADL PLAs. As a part of projects, we propose a timing-aware variability resolution approach in EAST-ADL architecture to generate valid product variants effectively. The method relies on existing EAST-ADL product line architecture for system modeling, where several variability points at different levels of abstraction are identified. A variability resolution algorithm is proposed where several configuration decisions starting from the topmost vehicle level down to the design level are incorporated for seamless variability resolution. Furthermore, timing decisions based on analysis and design prototypes are provided to generate variant-specific timing constraints. Consequently, the verification of product variants can be performed instantly in the presence of variant-specific timing constraints. The approach is validated via car wiper use case from our industrial partner - Volvo, where three product variants comprising a full system model with associated timing constraints are generated successfully.

The timing-aware variability resolution algorithm is implemented in JAVA that operates on EAXML format to offer broader tool-independent execution on EAST-ADL PLAs. The implementation architecture is shown in **Figure 1**.

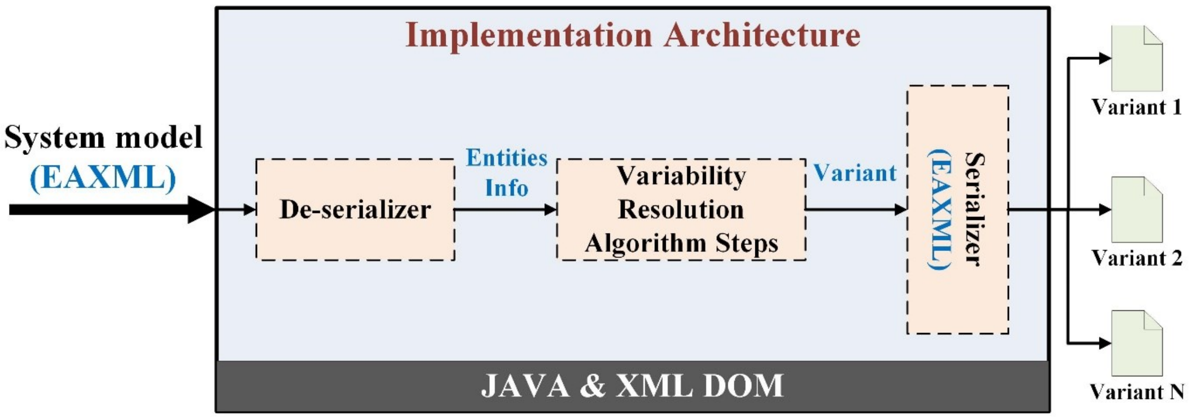


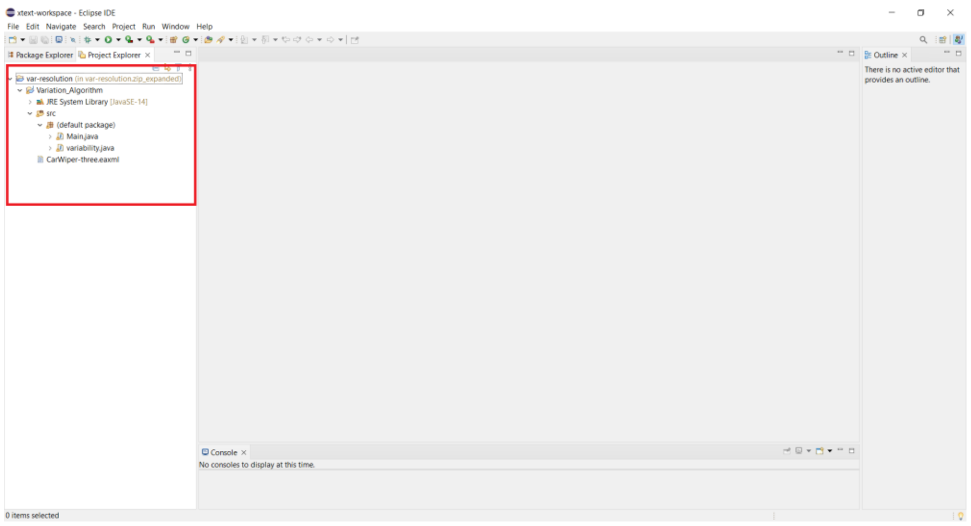
Figure 1: Implementation Architecture of Timing-aware Variability Resolution Algorithm

In this manual, we demonstrate the usage / working of timing-aware variability resolution algorithm via care wiper use case with three variants i.e. AutomaticWiperWithRainSensor, RearWiper and WithStalks. The implementations of variability resolution algorithm along with the detailed installation guide are publicly available at GitHub i.e. https://github.com/MDH-BUMBLE/EASTADL-SPL. We describe the essential steps to execute the timing-aware variability resolution algorithm in Section 2.

2. Timing-aware variability resolution algorithm in practice

Steps to execute variability resolution algorithm are as follows:

Step 1 – Import variability resolution project in Eclipse:- The detailed instructions about how to import variability resolution archive project in Eclipse are available in installation guide. Once a project is successfully imported into Eclipse, you can see two implementation Java files (i.e. Main and variability) as shown in **Figure 2**. All major steps of variability resolution algorithm are implemented in Main.java file while variability.java deals with the systematic execution of steps through function calls.

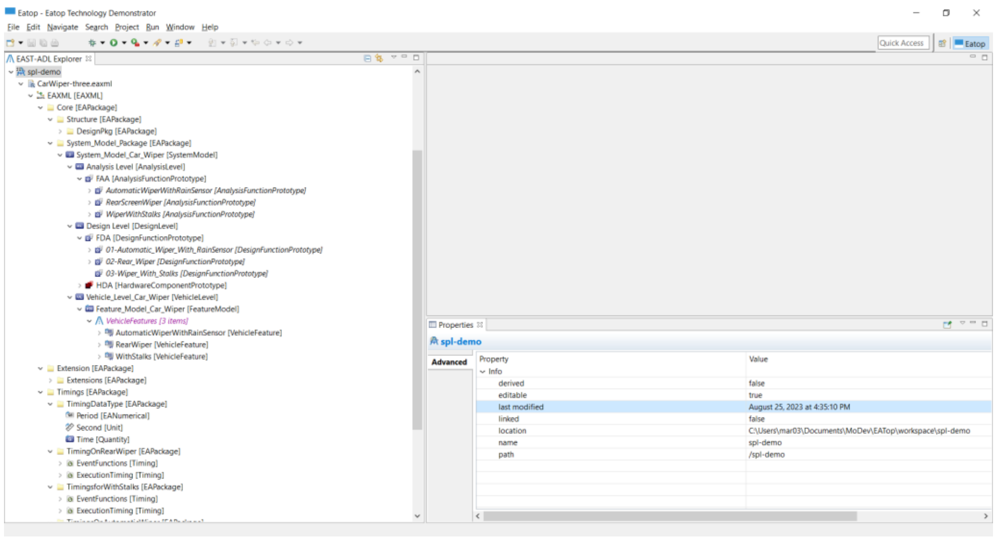

xtext-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Package Explorer
Project Explorer
Outline
There is no active editor that provides an outline.
Console
0 items selected

```
var-resolution (in var-resolution.zip_expanded)
  Variation_Algorithm
    JRE System Library [JavaSE-14]
    src
      (default package)
        Main.java
        variability.java
    CarWiper-three.eaxml
```

No consoles to display at this time.

Figure 2: Variability Resolution Project in Eclipse

Step 2 (Optional) – Visualization of wiper use case model in EATOP:- The wiper use case with three variants (i.e. AutomaticWiperWithRainSensor, RearWiper and WithStalks) is publicly available as **CarWiper-three.eaxml**. You can import / open wiper use case in EATOP for visualization as shown in **Figure 3**. You can also modify wiper use case model for experimentation purposes if needed. Please note step 2 is optional as you can directly execute the variability resolution algorithm on given use case (CarWiper-three.eaxml**)** if you don't want to modify it.

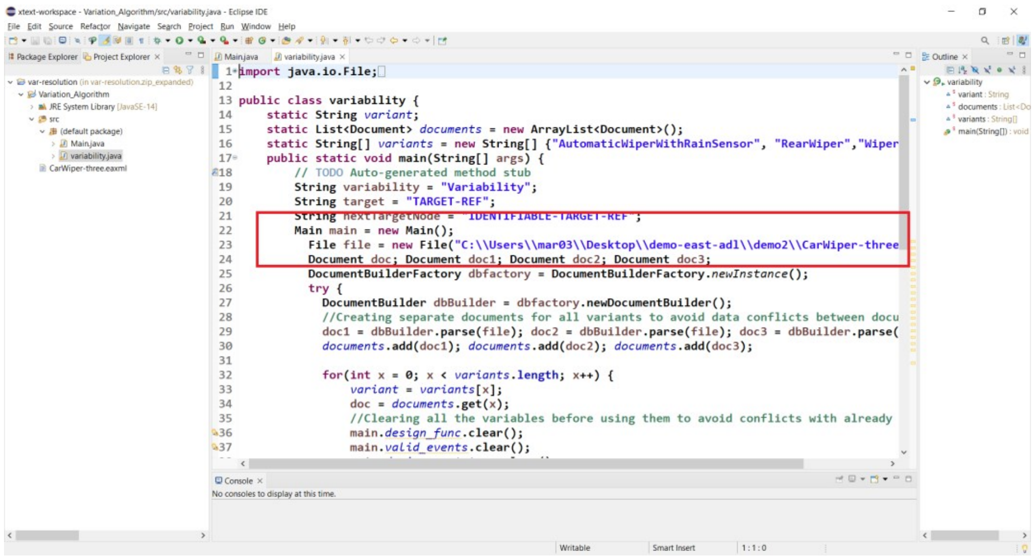

Eatop - Eatop Technology Demonstrator
File Edit Navigate Search Project Run Window Help
Quick Access
Eatop
EAST-ADL Explorer
Properties
spl-demo
Advanced

```
spl-demo
  CarWiper-three.eaxml
    EAXML [EAXML]
      Core [EAPackage]
        Structure [EAPackage]
          DesignPkg [EAPackage]
        System_Model_Package [EAPackage]
          System_Model_Car_Wiper [SystemModel]
            Analysis Level [AnalysisLevel]
              FAA [AnalysisFunctionPrototype]
                AutomaticWiperWithRainSensor [AnalysisFunctionPrototype]
                RearScreenWiper [AnalysisFunctionPrototype]
                WiperWithStalks [AnalysisFunctionPrototype]
            Design Level [DesignLevel]
              FDA [DesignFunctionPrototype]
                01-Automatic_Wiper_With_RainSensor [DesignFunctionPrototype]
                02-Rear_Wiper [DesignFunctionPrototype]
                03-Wiper_With_Stalks [DesignFunctionPrototype]
              HDA [HardwareComponentPrototype]
            Vehicle_Level_Car_Wiper [VehicleLevel]
              Feature_Model_Car_Wiper [FeatureModel]
                VehicleFeatures [3 items]
                  AutomaticWiperWithRainSensor [VehicleFeature]
                  RearWiper [VehicleFeature]
                  WithStalks [VehicleFeature]
      Extension [EAPackage]
        Extensions [EAPackage]
      Timings [EAPackage]
        TimingDataType [EAPackage]
          Period [EANumerical]
          Second [Unit]
          Time [Quantity]
        TimingOnRearWiper [EAPackage]
          EventFunctions [Timing]
          ExecutionTiming [Timing]
        TimingsforWithStalks [EAPackage]
          EventFunctions [Timing]
          ExecutionTiming [Timing]
```

Property	Value
Info	
derived	false
editable	true
last modified	August 25, 2023 at 4:35:10 PM
linked	false
location	C:\Users\mar03\Documents\MoDev\EATop\workspace\spl-demo
name	spl-demo
path	/spl-demo

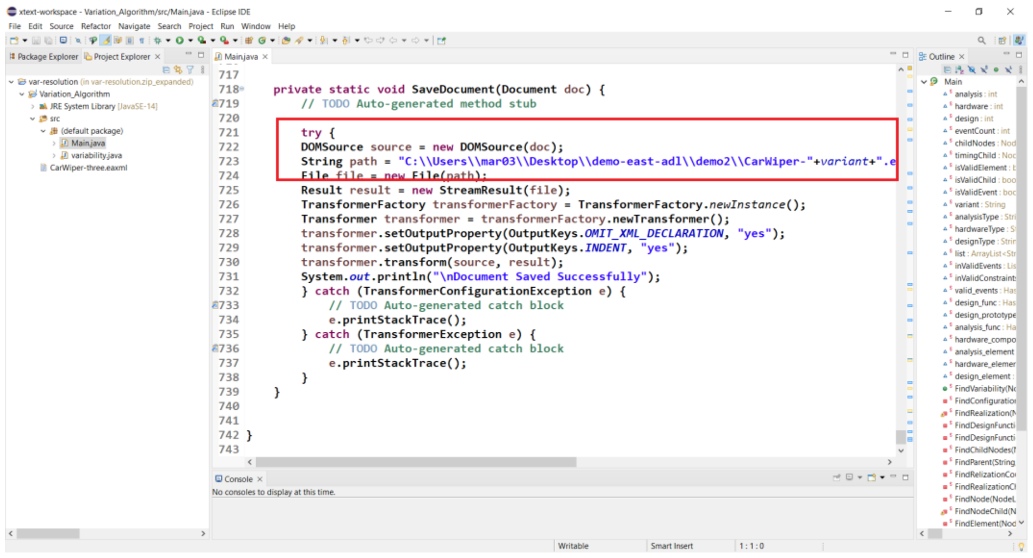
Figure 3: Wiper case study with three variants in EATOP

Step 3 – Update source and target models paths in project:- It is essential to update source and target models path in variability resolution project for seamless execution. Source model refers to the given wiper use case containing the information about three variants (CarWiper-three.eaxml). You should change the source model path in **variability.java at line 23 (see Figure 4)**, as per the configurations of your system. Target model path refers to the path of the folder where three variants need to be generated after the execution of variability resolution algorithm. You should change the target models path in **main.java at line 723**, as shown in **Figure 5**.



```
 1 import java.io.File;
12
13 public class variability {
14     static String variant;
15     static List<Document> documents = new ArrayList<Document>();
16     static String[] variants = new String[] {"AutomaticWiperWithRainSensor", "RearWiper","Wiper
17     public static void main(String[] args) {
18         // TODO Auto-generated method stub
19         String variability = "Variability";
20         String target = "TARGET-REF";
21         String nextTargetNode = "IDENTIFIABLE-TARGET-REF";
22         Main main = new Main();
23           File file = new File("C:\\Users\\mar03\\Desktop\\demo-east-adl\\demo2\\CarWiper-three
24           Document doc; Document doc1; Document doc2; Document doc3;
25           DocumentBuilderFactory dbfactory = DocumentBuilderFactory.newInstance();
26           try {
27             DocumentBuilder dbBuilder = dbfactory.newDocumentBuilder();
28             //Creating separate documents for all variants to avoid data conflicts between docu
29             doc1 = dbBuilder.parse(file); doc2 = dbBuilder.parse(file); doc3 = dbBuilder.parse(
30             documents.add(doc1); documents.add(doc2); documents.add(doc3);
31
32             for(int x = 0; x < variants.length; x++) {
33                 variant = variants[x];
34                 doc = documents.get(x);
35                 //Clearing all the variables before using them to avoid conflicts with already
36                 main.design_func.clear();
37                 main.valid_events.clear();
```

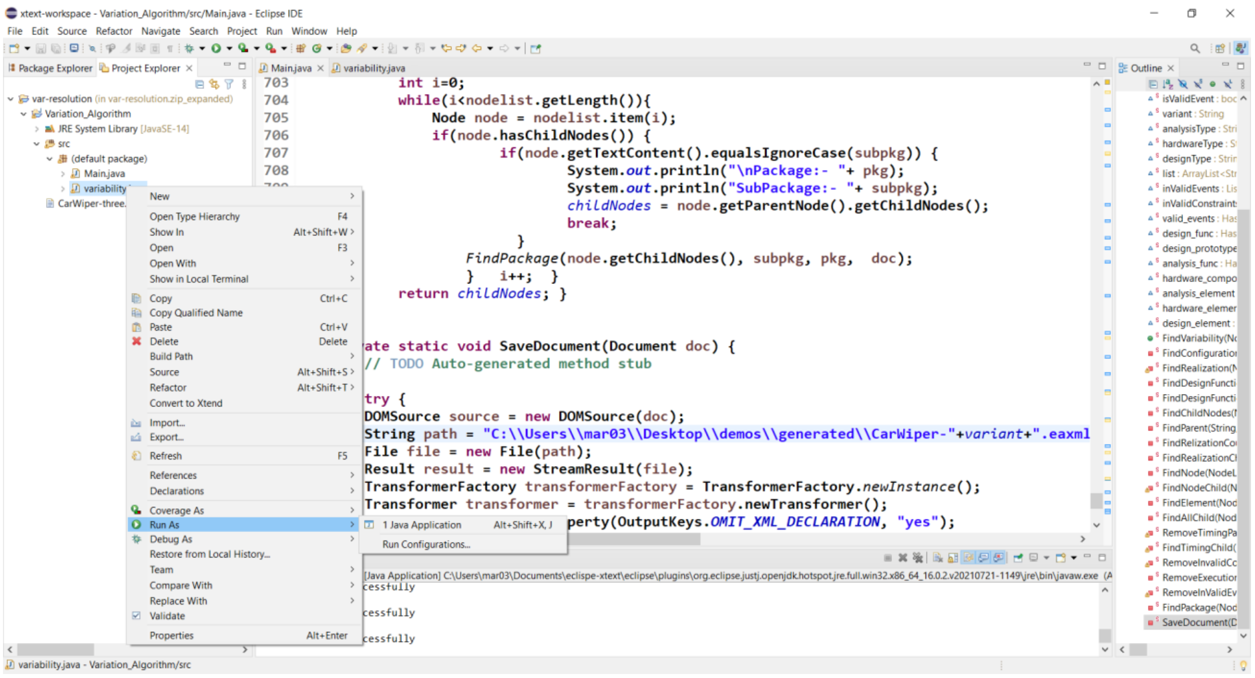
Figure 4: Update given wiper use case path in variability.java file



```
717
718     private static void SaveDocument(Document doc) {
719         // TODO Auto-generated method stub
720
721         try {
722         DOMSource source = new DOMSource(doc);
723         String path = "C:\\Users\\mar03\\Desktop\\demo-east-adl\\demo2\\CarWiper-"+variant+".e
724         File file = new File(path);
725         Result result = new StreamResult(file);
726         TransformerFactory transformerFactory = TransformerFactory.newInstance();
727         Transformer transformer = transformerFactory.newTransformer();
728         transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
729         transformer.setOutputProperty(OutputKeys.INDENT, "yes");
730         transformer.transform(source, result);
731         System.out.println("\nDocument Saved Successfully");
732         } catch (TransformerConfigurationException e) {
733             // TODO Auto-generated catch block
734             e.printStackTrace();
735         } catch (TransformerException e) {
736             // TODO Auto-generated catch block
737             e.printStackTrace();
738         }
739     }
740
741
742 }
743
```

Figure 5: Update folder path where product variants need to be generated

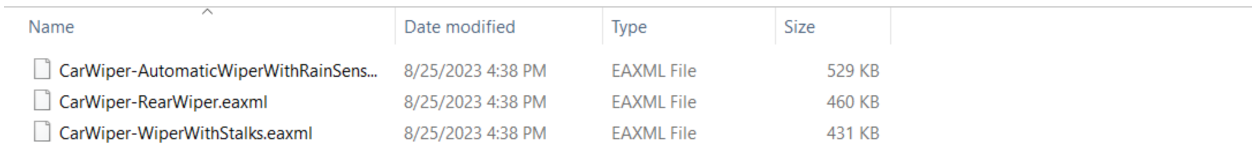
Step 4 – Execute variability resolution algorithm:- Once step 3 is successfully performed, variability resolution algorithm can be executed in Eclipse, as shown in **Figure 6**. Three product variants with associated software and hardware components along with timing constraints are successfully generated, as shown in **Figure 7**. The generated product variants can be directly loaded and visualized in EATOP, as shown in **Figure 8** for automatic wiper with rain sensor variant. It can be seen from **Figure 8** that all hardware and software components along with associated timing constraints are also available in generated automatic wiper variant.



```
        int i=0;
        while(i<nodelist.getLength()){
            Node node = nodelist.item(i);
            if(node.hasChildNodes()) {
                    if(node.getTextContent().equalsIgnoreCase(subpkg)) {
                            System.out.println("\nPackage:- "+ pkg);
                            System.out.println("SubPackage:- "+ subpkg);
                            childNodes = node.getParentNode().getChildNodes();
                            break;
                    }
                FindPackage(node.getChildNodes(), subpkg, pkg,  doc);
            }   i++;   }
        return childNodes; }

ate static void SaveDocument(Document doc) {
// TODO Auto-generated method stub

try {
DOMSource source = new DOMSource(doc);
String path = "C:\\Users\\mar03\\Desktop\\demos\\generated\\CarWiper-"+variant+".eaxml
File file = new File(path);
Result result = new StreamResult(file);
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
                                            perty(OutputKeys.OMIT_XML_DECLARATION, "yes");
```

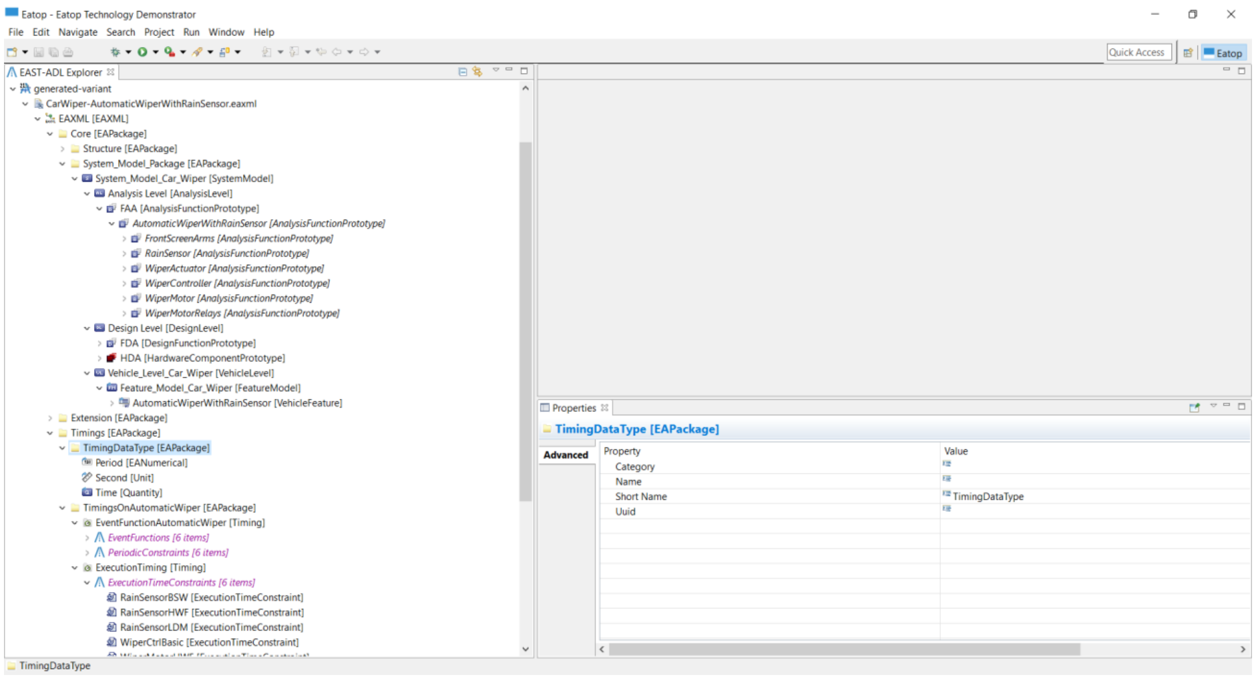
```
[Java Application] C:\Users\mar03\Documents\eclispe-xtext\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149\jre\bin\javaw.exe (A
cessfully

cessfully

cessfully
```

Figure 6: Executing Variability Resolution Algorithm in Eclipse



Name	Date modified	Type	Size
CarWiper-AutomaticWiperWithRainSens...	8/25/2023 4:38 PM	EAXML File	529 KB
CarWiper-RearWiper.eaxml	8/25/2023 4:38 PM	EAXML File	460 KB
CarWiper-WiperWithStalks.eaxml	8/25/2023 4:38 PM	EAXML File	431 KB

Figure 7: Generated Product Variants


Eatop - Eatop Technology Demonstrator
File Edit Navigate Search Project Run Window Help
Quick Access
Eatop
EAST-ADL Explorer
Properties
Advanced

```
generated-variant
  CarWiper-AutomaticWiperWithRainSensor.eaxml
    EAXML [EAXML]
      Core [EAPackage]
        Structure [EAPackage]
        System_Model_Package [EAPackage]
          System_Model_Car_Wiper [SystemModel]
            Analysis Level [AnalysisLevel]
              FAA [AnalysisFunctionPrototype]
                AutomaticWiperWithRainSensor [AnalysisFunctionPrototype]
                  FrontScreenArms [AnalysisFunctionPrototype]
                  RainSensor [AnalysisFunctionPrototype]
                  WiperActuator [AnalysisFunctionPrototype]
                  WiperController [AnalysisFunctionPrototype]
                  WiperMotor [AnalysisFunctionPrototype]
                  WiperMotorRelays [AnalysisFunctionPrototype]
            Design Level [DesignLevel]
              FDA [DesignFunctionPrototype]
              HDA [HardwareComponentPrototype]
            Vehicle_Level_Car_Wiper [VehicleLevel]
              Feature_Model_Car_Wiper [FeatureModel]
                AutomaticWiperWithRainSensor [VehicleFeature]
      Extension [EAPackage]
      Timings [EAPackage]
        TimingDataType [EAPackage]
          Period [EANumerical]
          Second [Unit]
          Time [Quantity]
        TimingsOnAutomaticWiper [EAPackage]
          EventFunctionAutomaticWiper [Timing]
            EventFunctions [6 items]
            PeriodicConstraints [6 items]
          ExecutionTiming [Timing]
            ExecutionTimeConstraints [6 items]
              RainSensorBSW [ExecutionTimeConstraint]
              RainSensorHWF [ExecutionTimeConstraint]
              RainSensorLDM [ExecutionTimeConstraint]
              WiperCtrlBasic [ExecutionTimeConstraint]
TimingDataType
```

TimingDataType [EAPackage]

Property	Value
Category	
Name	
Short Name	TimingDataType
Uuid	

Figure 8: Generated Product Variant (Automatic Wiper) in EATOP