

ARText.

Driving developments with Xtext.

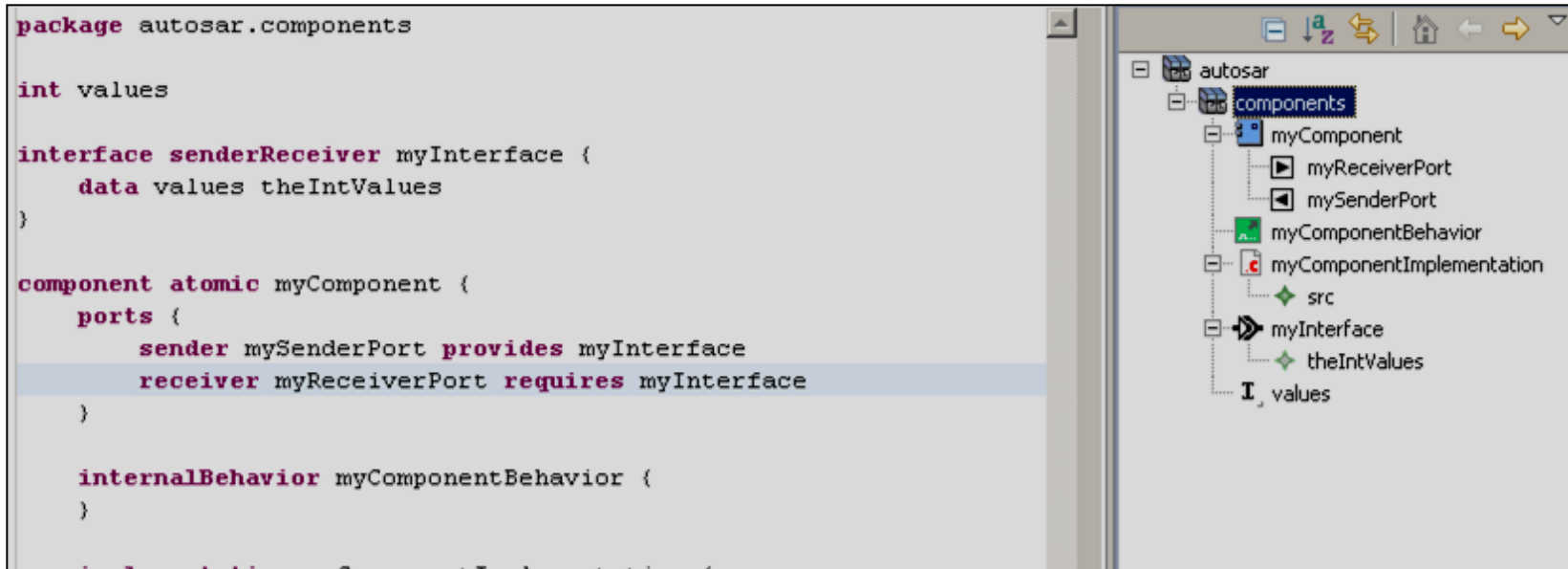
```
package autosar.components

int values

interface senderReceiver myInterface {
    data values theIntValues
}

component atomic myComponent {
    ports {
        sender mySenderPort provides myInterface
        receiver myReceiverPort requires myInterface
    }

    internalBehavior myComponentBehavior {
    }
}
```

The image shows a screenshot of an IDE with two panes. The left pane displays Xtext code for an AUTOSAR component. The code defines a package 'autosar.components', an integer 'values', an interface 'senderReceiver myInterface' with a data attribute 'theIntValues', and an atomic component 'myComponent'. The component has two ports: 'mySenderPort' which provides 'myInterface', and 'myReceiverPort' which requires 'myInterface'. It also has an internal behavior 'myComponentBehavior'. The right pane shows a project tree for 'autosar' with a sub-package 'components'. The tree contains 'myComponent' (with sub-elements 'myReceiverPort' and 'mySenderPort'), 'myComponentBehavior', 'myComponentImplementation' (with a source folder 'src'), 'myInterface' (with a source folder 'theIntValues'), and a variable 'values'.

Sebastian.Benz@bmw-carit.de

Dana.Wong@bmw-carit.de

BMW Car IT



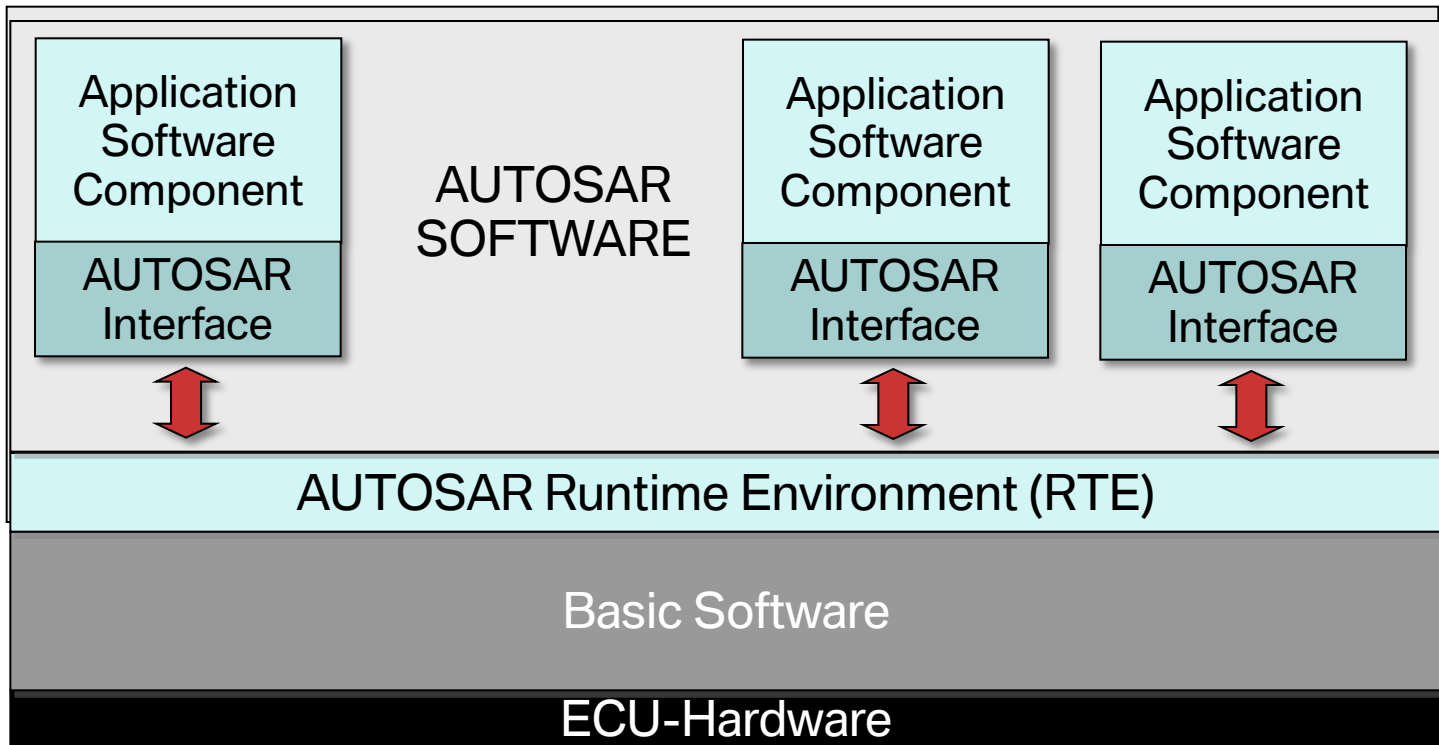
AUTOSAR.

Automotive Open Systems Architecture.



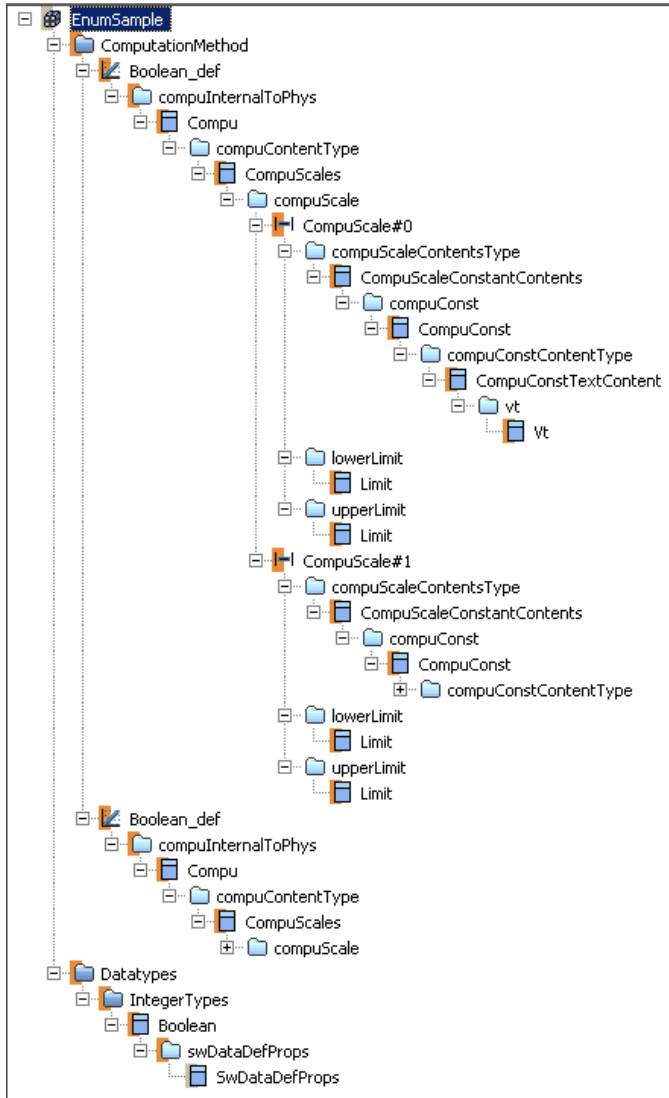
AUTOSAR.

Automotive Open Systems Architecture.



AUTOSAR.

Modeling AUTOSAR Systems.



```
package EnumSample {
    enum Boolean ( TRUE, FALSE)
}
```

Goal.

A Textual Language for AUTOSAR.

The image displays two windows from an IDE. The left window, titled 'components.artext', shows the following ARText code:

```
package AUTOSAR.components

interface senderReceiver mySRInterface {
    data SInt16 intArg
    data Boolean boolArg
}

component atomic myComponent {

    ports {
        sender sPort provides mySRInterface
        receiver rPort requires mySRInterface
    }

}

composition System {
    prototype myComponent comp1
    prototype myComponent comp2

    connect comp1.sPort to comp2.rPort
    connect comp2.sPort to comp1.rPort
}
```

The right window, titled 'AUTOSAR Browser', shows a hierarchical tree view of the model. The root is 'AUTOSAR', which contains 'DataTypes', 'Interfaces', 'SwBaseTypes', and 'components'. Under 'components', there is a 'System' component containing 'comp1' and 'comp2'. Below 'System' are two connection nodes: 'myComponent_sPort_to_myComponen' and 'myComponent_sPort_to_myComponen'. Below these is the 'myComponent' component, which has 'rPort' and 'sPort' ports. At the bottom is the 'mySRInterface' component, which has 'boolArg' and 'intArg' data types.

AUTOSAR Textual Language.

There is an Eclipse Project for that...

Xtext

AUTOSAR Textual Language. Challenges.

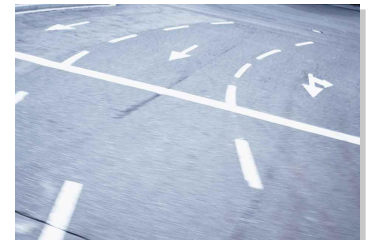
1. Performance



2. Workspace integration



3. Metamodel support



Xtext. Performance.

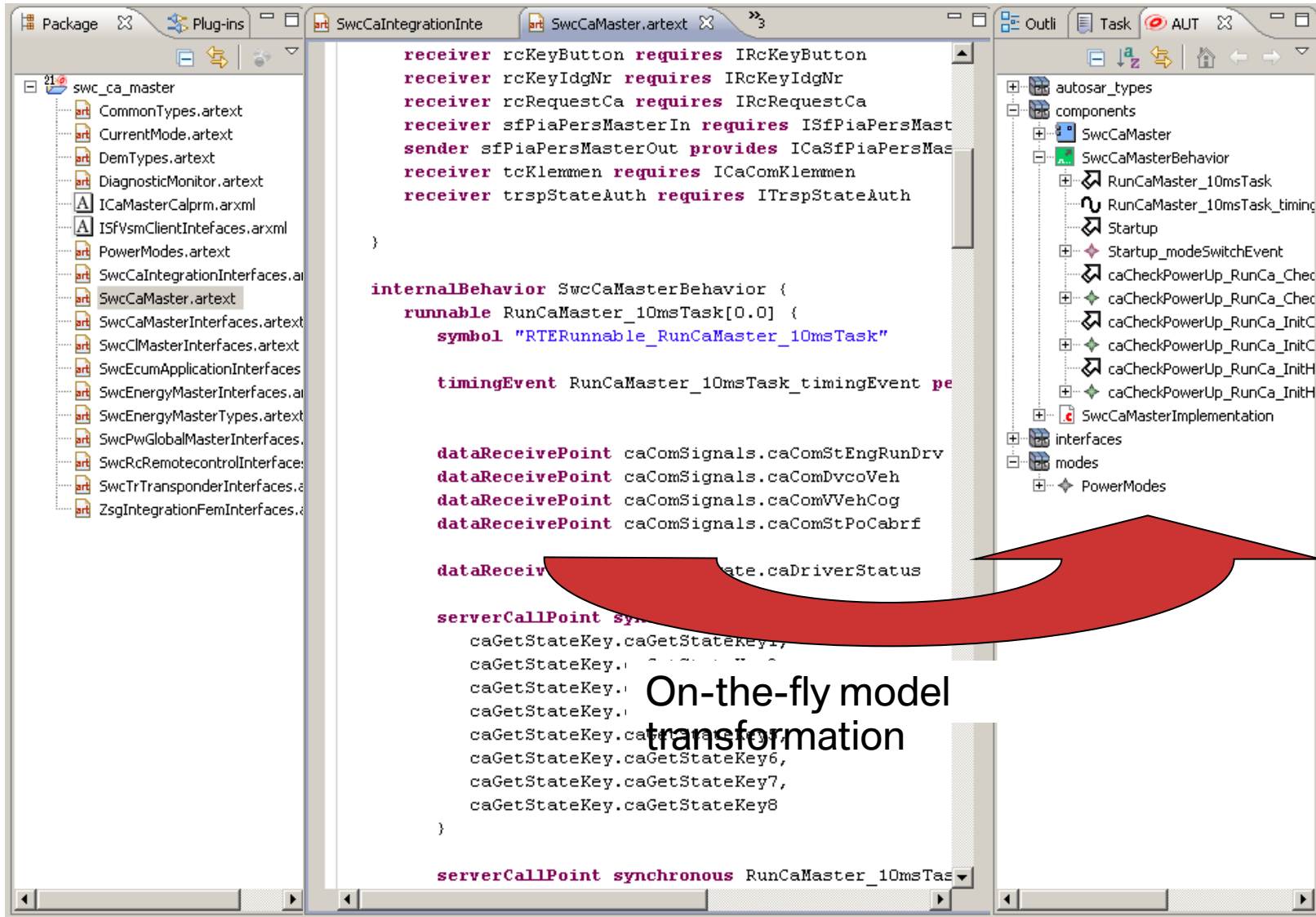
Time for parsing a series model (1500 files):



AUTOSAR Tool Platform. Integration into Artop.



Artop. Workspace Integration.



The screenshot displays the Artop IDE workspace with three main panels:

- Left Panel (Package Explorer):** Shows a tree view of the project structure under 'swc_ca_master', including files like 'CommonTypes.artext', 'CurrentMode.artext', 'DemTypes.artext', 'DiagnosticMonitor.artext', 'ICaMasterCalprm.arxml', 'ISFvsmClientInterfaces.arxml', 'PowerModes.artext', 'SwcCaIntegrationInterfaces.arxml', 'SwcCaMaster.artext', 'SwcCaMasterInterfaces.artext', 'SwcCIMasterInterfaces.artext', 'SwcEcumApplicationInterfaces.arxml', 'SwcEnergyMasterInterfaces.arxml', 'SwcEnergyMasterTypes.artext', 'SwcPwGlobalMasterInterfaces.arxml', 'SwcRcRemotecontrolInterfaces.arxml', 'SwcTrTransponderInterfaces.arxml', and 'ZsgIntegrationFemInterfaces.arxml'.
- Center Panel (Code Editor):** Displays the source code for 'SwcCaMasterBehavior'. The code includes several receiver and sender methods, an internal behavior block with a runnable task and a timing event, and server call points for state keys. A red arrow points from the 'RunCaMaster_10msTask' symbol in the code to the 'RunCaMaster_10msTask' component in the model tree.
- Right Panel (Model Explorer):** Shows a hierarchical model tree with components like 'SwcCaMasterBehavior', 'RunCaMaster_10msTask', 'Startup', and 'PowerModes'.

Code Snippet:

```
receiver rcKeyButton requires IRcKeyButton
receiver rcKeyIdgNr requires IRcKeyIdgNr
receiver rcRequestCa requires IRcRequestCa
receiver sfPiaPersMasterIn requires ISfPiaPersMasterIn
sender sfPiaPersMasterOut provides ICaSfPiaPersMasterOut
receiver tcKlemmen requires ICaComKlemmen
receiver trspStateAuth requires ITrspStateAuth

}

internalBehavior SwcCaMasterBehavior {
  runnable RunCaMaster_10msTask[0.0] {
    symbol "RTERunnable_RunCaMaster_10msTask"

    timingEvent RunCaMaster_10msTask_timingEvent pe

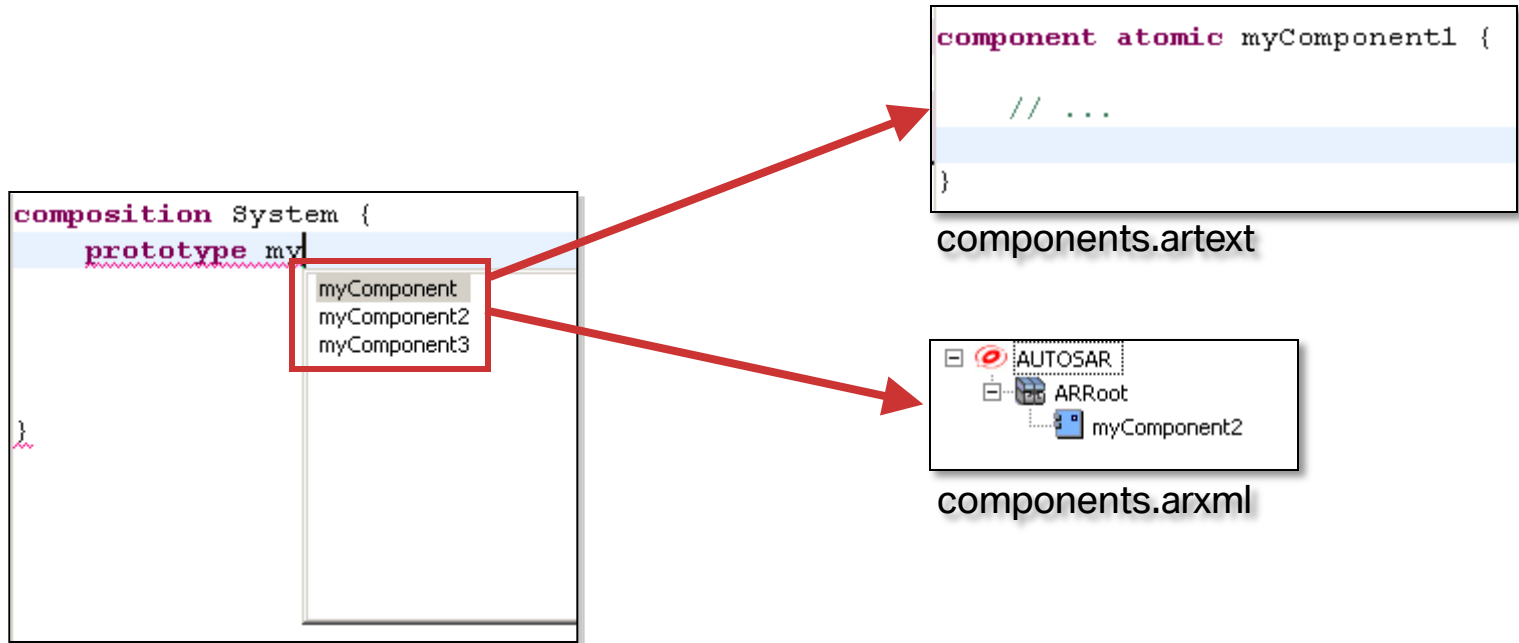
    dataReceivePoint caComSignals.caComStEngRunDrv
    dataReceivePoint caComSignals.caComDvcoVeh
    dataReceivePoint caComSignals.caComVvehCog
    dataReceivePoint caComSignals.caComStPoCabrf

    dataReceivePoint caComSignals.caComStPoCabrf

    serverCallPoint synchronous RunCaMaster_10msTask {
      caGetStateKey.caGetStateKey1,
      caGetStateKey.caGetStateKey2,
      caGetStateKey.caGetStateKey3,
      caGetStateKey.caGetStateKey4,
      caGetStateKey.caGetStateKey5,
      caGetStateKey.caGetStateKey6,
      caGetStateKey.caGetStateKey7,
      caGetStateKey.caGetStateKey8
    }
  }
}
```

On-the-fly model
transformation

Artop. Workspace Integration.



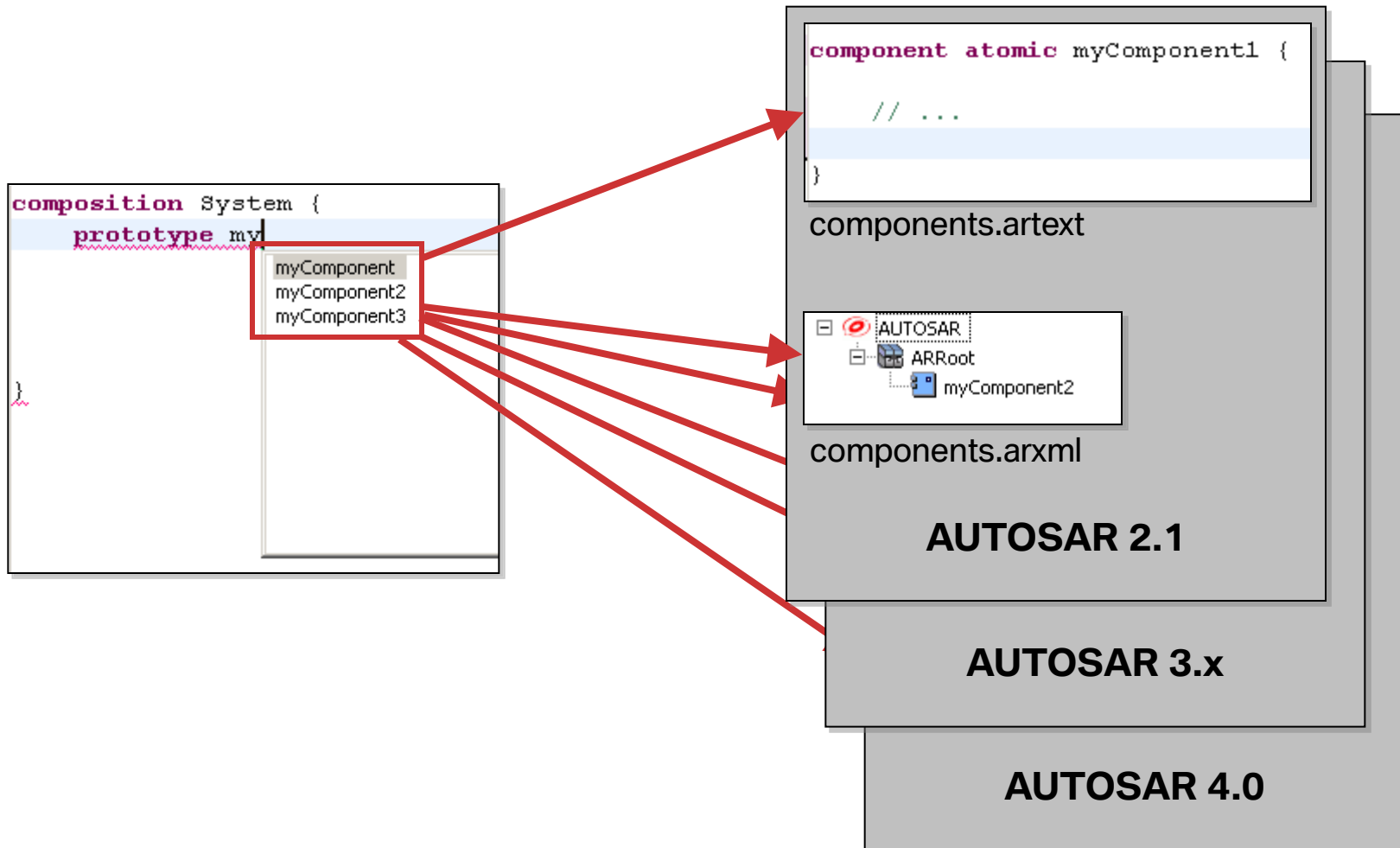
The AUTOSAR Dilemma.

Handling different AUTOSAR Releases.



The AUTOSAR Dilemma.

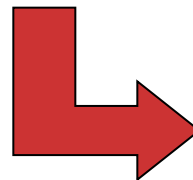
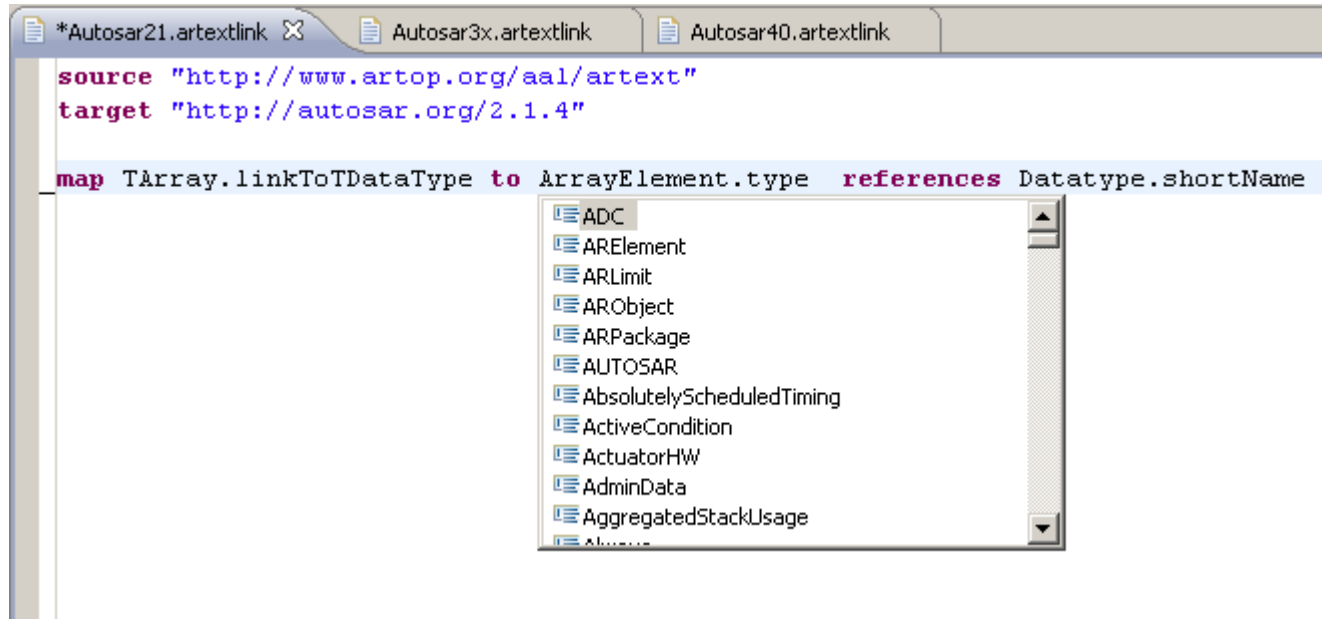
Supporting different AUTOSAR Releases.



Supporting different AUTOSAR Releases. The Linking DSL.

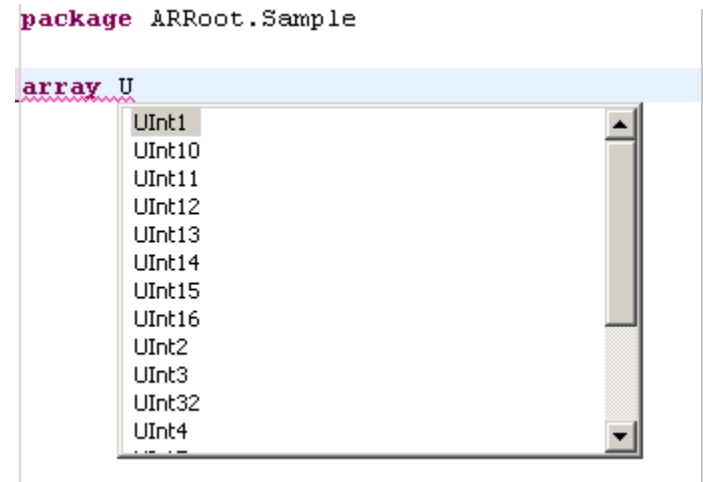
```
*Autosar21.artextlink  Autosar3x.artextlink  Autosar40.artextlink
source "http://www.artop.org/aal/artext"
target  "http://autosar.org/2.1.4"

map TArray.linkToDataType to ArrayElement.type references Datatype.shortName
```

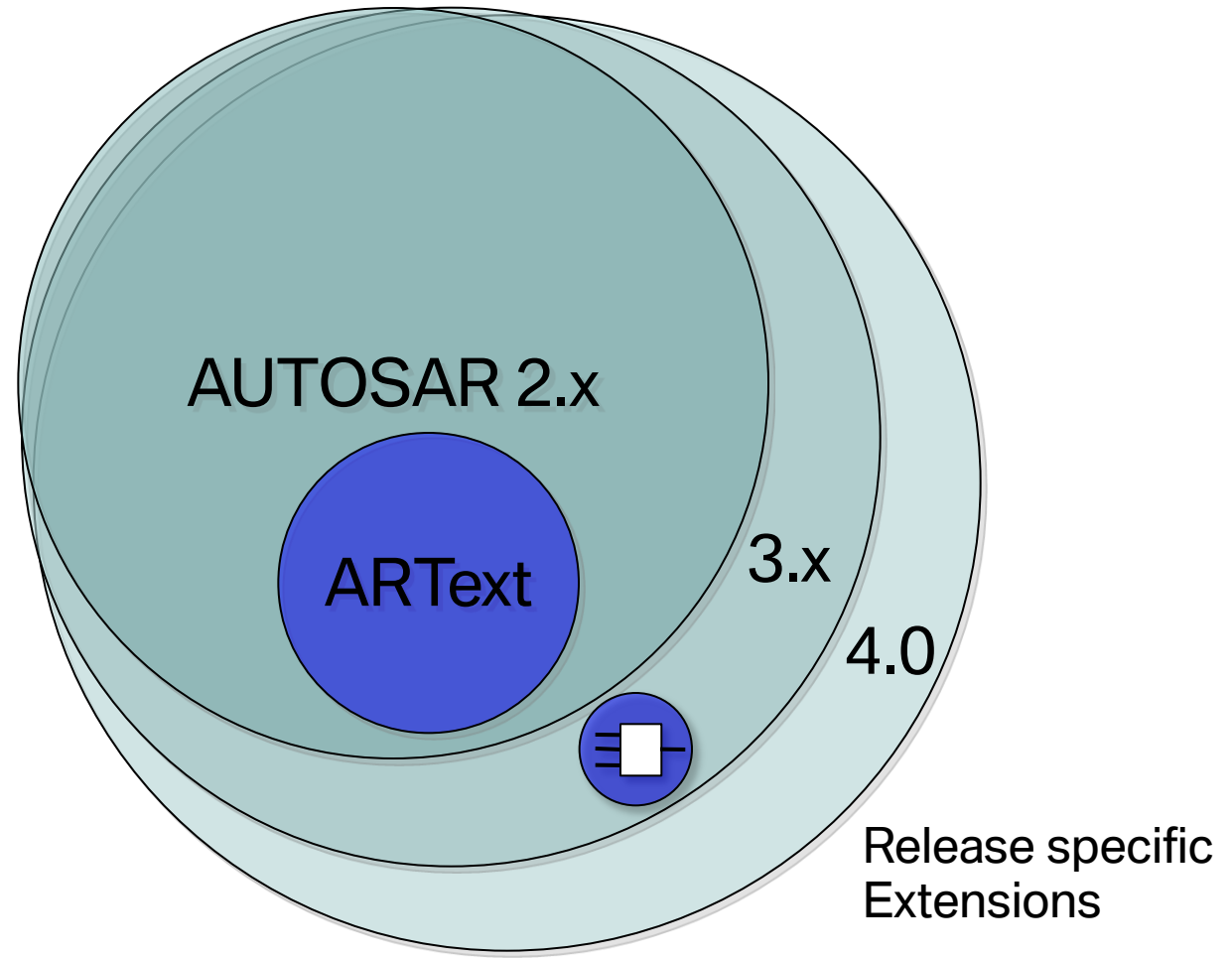


```
package ARRoot.Sample

array U
  UInt1
  UInt10
  UInt11
  UInt12
  UInt13
  UInt14
  UInt15
  UInt16
  UInt2
  UInt3
  UInt32
  UInt4
```



The AUTOSAR Dilemma. Metamodel Differences.



Supporting different AUTOSAR Releases. Language Extensions.

TCompositionType:

```
'composition' name=ID '{'  
    (...|  
    extensionPoints += TExtensionPoint)*  
'}'
```

;

TExtensionPoint:

```
keyword=ID (args+=TArg (',' args+=TArg)* )?;
```

TArg:

```
ListLiteral | BoolLiteral | StringLiteral | ...;
```

Supporting different AUTOSAR Releases. Language Extensions.

The screenshot shows the Eclipse IDE's 'Extensions' dialog for the plugin `org.artop.aal.artext.autosar21`. The dialog is titled 'Extensions' and contains two main panels: 'All Extensions' and 'Extension Element Details'.

All Extensions: This panel allows defining extensions for the plug-in. It includes a search filter 'type filter text' and a tree view of extensions. The tree shows the following structure:

- `org.artop.aal.artext.artextExtension` (Add...)
- `org.artop.aal.artext.artextautosar21.CheckSumExtension` (Remove)
- `org.artop.aal.autosar21 (autosarVersion)` (Remove)

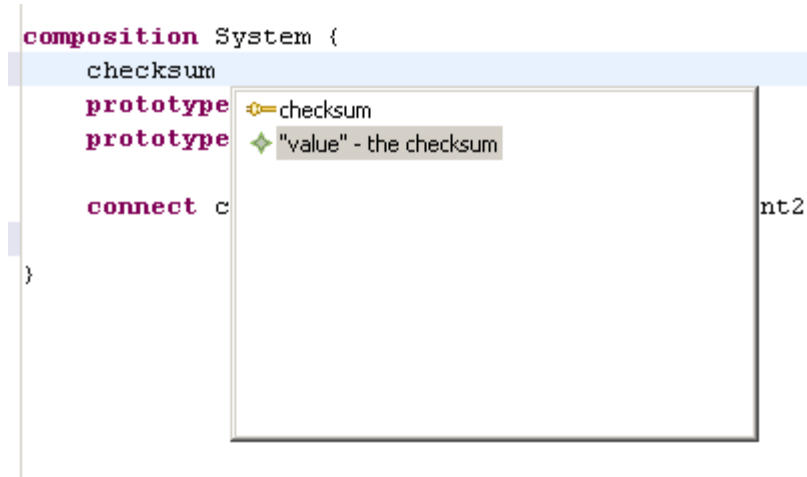
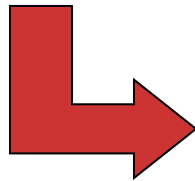
Buttons for 'Up' and 'Down' are also present. A scroll bar is visible at the bottom of the tree view.

Extension Element Details: This panel is used to set the properties of the selected extension element. It includes a 'class*' field with the value `org.artop.aal.artext.artextautosar21.CheckSumExtension` and a 'Browse...' button. The text above the field states: 'Set the properties of "artextExtension". Required fields are denoted by "*".'

The bottom of the dialog features a breadcrumb trail: Overview | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MF | plugin.xml | build.properties.

Supporting different AUTOSAR Releases. Language Extensions.

```
@ArtextExtension(  
    artextElement = TCompositionType.class,  
    keyword = "checksum",  
    params = { "the checksum" }  
)  
public void execute(CompositionType container, String checksum){  
    i.setChecksum(theChecksum);  
}
```



Results.

Modeling Efficiency.

Mouse Clicks:



Key presses:



Modeling time (min):



Summary.

Using Xtext in practice.

Scalable: handles large projects.



Extendable: thanks to Guice
non-trivial customizations are possible.



Efficient: short development time.



Thank you for your attention.

Questions?

The image shows two windows from the ARText development environment. The left window, titled 'components.artext' and 'interfaces.artext', displays the following code:

```
package AUTOSAR.components

interface senderReceiver mySRInterface {
    data SInt16 intArg
    data Boolean boolArg
}

component atomic myComponent {

    ports {
        sender sPort provides mySRInterface
        receiver rPort requires mySRInterface
    }
}

composition System {
    prototype myComponent comp1
    prototype myComponent comp2

    connect comp1.sPort to comp2.rPort
    connect comp2.sPort to comp1.rPort
}
```

The right window, titled 'AUTOSAR Browser', shows a hierarchical tree view of the model. The root is 'AUTOSAR', which contains 'DataTypes', 'Interfaces', 'SwBaseTypes', and 'components'. Under 'components', there is a 'System' component containing 'comp1' and 'comp2'. Below 'System' are two connection nodes: 'myComponent_sPort_to_myComponen' and 'myComponent_sPort_to_myComponen'. Under 'myComponent' are 'rPort' and 'sPort'. At the bottom is 'mySRInterface', which contains 'boolArg' and 'intArg'.

This section shows the continuation of the code editor from the previous image, displaying the connection lines for the composition:

```
connect comp1.sPort to comp2.rPort
connect comp2.sPort to comp1.rPort
```