POLITECNICO DI TORINO

III Facoltà di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

VIRTUAL PROTOTYPIN IN AUTOMOTIVE EMBEDDED SYSTEMS

Relatore:
prof. Massimo Violante

Candidato:
Ladan Karimzadeh

November 2020

Contents

1 Introduction

1.1 Why AUTOSAR?

Innovative vehicle functions lead to a steady increase in the complexity of the E / E (electrical/electronic) vehicle architecture. The requirements for the development are often contradictory. Additional driver assistance systems should support critical driving maneuvers, while at the same time consumption values should be reduced and environmental standards should be adhered to. The ever deeper integration of infotainment and communication with the direct vehicle environment and online services result in further challenges.

In order to continue to meet these requirements in the future, a new technological approach is required for the software architecture of the control units. Otherwise, the increasing demands on the part of customers, but also of the legislature, cannot be met.

1.2 AUTOSAR Tool

Software tool which supports one or more tasks in the AUTOSAR methodology. Depending on the supported tasks, an AUTOSAR tool can act as an AUTOSAR authoring tool, an AUTOSAR converter tool, an AUTOSAR processor tool or as a combination of those.
According to the AUTOSAR-paradigm "Common standard, concurring implementations", several software suppliers offer software implementations of the AUTOSAR standard. Some of the suppliers of AUTOSAR standard software are:

Table 1-1 several software suppliers offer software implementations of the AUTOSAR standard[1]

Implementer	BSW/MCAL Implementation	BSW Configurator	RTE Generator	System Tooling
ArcCore	Arctic Core - Open source AUTOSAR	BSW Builder	RTE Builder	SWC Builder and Extract Builder
COMASSO_eV	BSW	BSWDT	No	No
Continental Engineering Services	Yes	Yes	Yes	Yes
dSPACE	No	No	SystemDesk RTE Generator	SystemDesk
wikipedia:Elektrobit	EB Tresos AutoCore	EB Tresos Studio	EB Tresos studio	No
ETAS	Yes	Yes	RTA	ISOLAR-A

Freescale	Yes http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=AUTOSAR_CS&tid=vanAutoSAR	No	Yes	Unknown
Dassault Systèmes	No	GCE	RTEG	AAT
Hyundai-Autron	Yes	Yes	Yes	Yes
wikipedia:KPIT Technologies Ltd.	K-SAR Suite	K-SAR Editor	Yes	K-SAR Editor
wikipedia:Mecel	Yes	Yes	Yes	Unknown
Mentor Graphics	Volcano VSTAR	Volcano VSTAR	Volcano VSTAR	Volcano Vehicle Systems Architect
OpenSynergy	COQOS (OS and BSW Scheduler only)	COQOS	COQOS	No
wikipedia:Renesas Electronics	Yes	No	No	No
wikipedia:see4sys	Yes	Yes	Yes	ECU-Designer
Vector Informatik GmbH	MICROSAR	DaVinci Configurator Pro	MICROSAR Rte Generator	PREEvision / DaVinci Developer

The purpose of this thesis is to evaluate Volcano (Mentor graphics) by Creating a simple model using AUTOSAR standard with Mentor Graphics Volcano (VSx), Matlab and Simulink.

The starting system is based on implementation of a Mathematical function, using the Model Based Software Design Techniques.

- This thesis is focused on the following steps:
  - **FIRST PART**
    - Creating a simple filter in Simulink
    - Defining the structure and the elements of new AUTOSAR Architecture (Software Components, Interfaces, Ports, Data Types, etc.)
    - Creating missing parts in the starting model (timers, trigger events, etc.)
    - AUTOSAR Consistency check in VSx
    - Exporting the skeleton to Simulink (arxml files)
    - Creating the behavior for the Application Software Components with Model Based Software Techniques using Simulink
    - Validation and Automatic Code Generation in Simulink
    - Export code in VSx (C, C++, h, arxml, etc.)
    - Configuring Runnables

[1] https://automotive.wiki/index.php/AUTOSAR_Tool

- Adding conneXion Software Components
- Creating the missing Application Software Components for using Sender/Receiver with conneXion Software Components (with hand written code for Runnables)
- Configuring the conneXion SWCs using Module Configuration Parameters
- Creating the Plant in Simulink (adding and configuring the conneXion blocks)

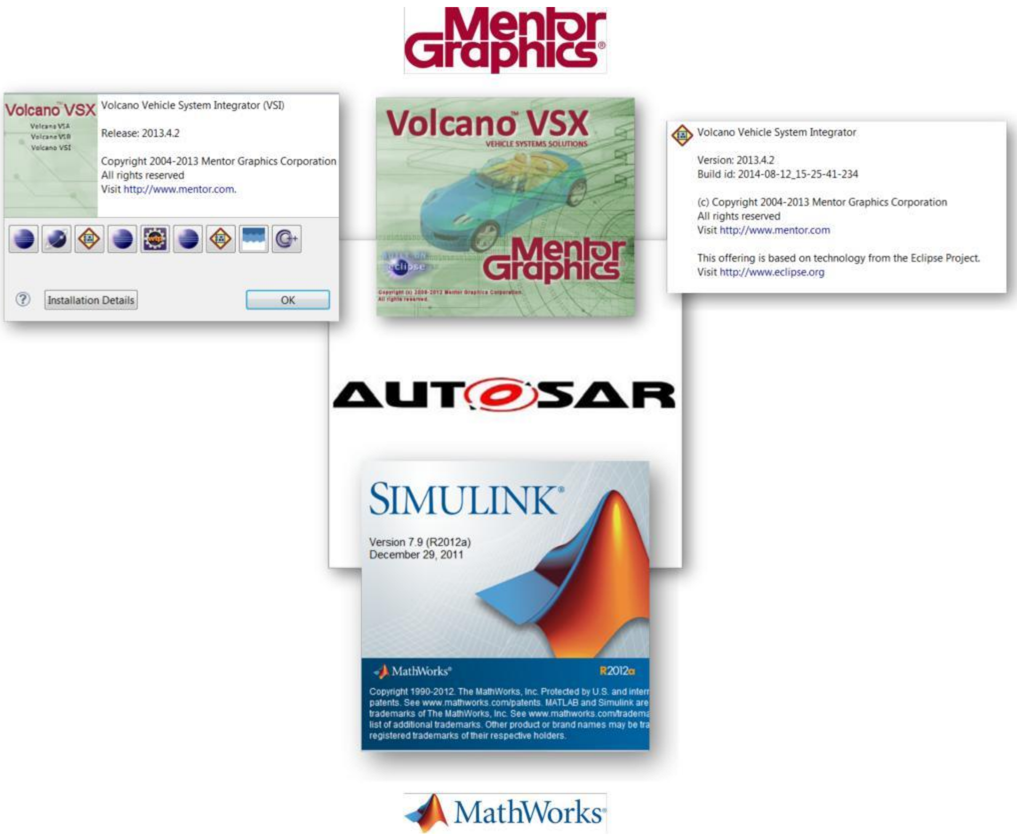
- **SECOND PART**

  - Integrating the two systems in the Plant (VSx model and Starting model)
  - Importing the old test cases in the new model (script code and model optimization/integration)
  - Building the model (virtual RTE and OS will be generated)
  - Debugging using VSI (on Architecture elements and/or on code)
  - Start Simulating with both models
  - Comparing testing results

2 Technical Details

The following tools are used:

- Matlab - Version 7.9 (2012a)
- Volcano Vehicle System Integrator (VSI) - Release 2013.4.2
  using AUTOSAR 4.0.3



Volcano Vehicle System Integrator (VSI)

Release: 2013.4.2

Copyright 2004-2013 Mentor Graphics Corporation
All rights reserved
Visit http://www.mentor.com.

Volcano Vehicle System Integrator

Version: 2013.4.2
Build id: 2014-08-12_15-25-41-234

(c) Copyright 2004-2013 Mentor Graphics Corporation
All rights reserved
Visit http://www.mentor.com

This offering is based on technology from the Eclipse Project.
Visit http://www.eclipse.org

Version 7.9 (R2012a)
December 29, 2011

Copyright 1990-2012. The MathWorks, Inc. Protected by U.S. and intern
patents. See www.mathworks.com/patents. MATLAB and Simulink are
trademarks of The MathWorks, Inc. See www.mathworks.com/tradema
list of additional trademarks. Other product or brand names may be tra
registered trademarks of their respective holders.

3 AUTOSAR

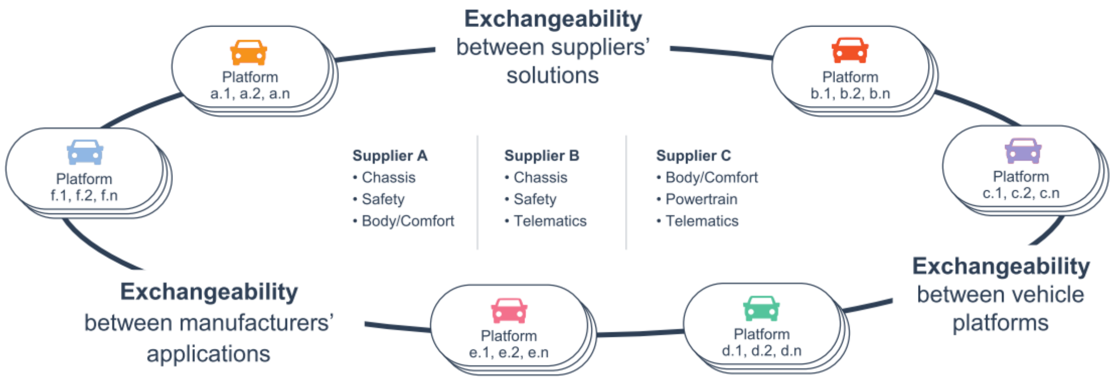


Figure 1.2-1 AUTOSAR exchangeability

3.1 AUTOSAR Introduction

In Automotive electronics, the growing increase in the number of ECUs and network components has led to an increase in the overall complexity of the hardware, thus in-creasing the costs related to hardware and introduction of new features.

The goal is to reduce these costs and **AUTOSAR** (**AUT**omotive **O**pen **S**ystem **AR**chitecture) is an attempt to solve this kind of problems in automotive.

Also with the market full of different car makers, there is the urge to use a common architecture to share systems that can be used as platforms for future applications.

In 2003, vehicle manufacturers and suppliers joined forces to form the AUTOSAR initiative to work on this new approach. The aim of the initiative is to curb the constant new development of the same or similar software components. [1]

AUTOSAR is:

- An open and standardized automotive software architecture [2], that has been developed by a group of automotive vendors and stakeholders
- A framework that helps the software developer to handle the complexity of systems and networks
- A consortium that generated a specification for this software architecture.

[1]https://www.all-electronics.de/ [2]http://www.autosar.org

3.2 AUTOSAR Importance

As we mentioned the increasing complexity of modern vehicles and especially their E/E systems was the main motivation behind the development of AUTOSAR. Furthermore, today's vehicles have more than a hundred ECUs each. Every one of them has thousands of functions. Without this standard, they often have to be completely rewritten when the hardware is changed.

This has made it urgent for automotive manufacturing giants to come together and make software independent from hardware. To make that possible they set AUTOSAR as an industry-wide standard, which was a core solution for software to become reusable.

3.3 AUTOSAR Goals

AUTOSAR aims to standardize the software architecture of Electronic Control Units (ECUs). AUTOSAR paves the way for innovative electronic systems that further improve performance, safety and security.

- Hardware and software widely independent of each other.
- Development can be decoupled (through abstraction) by horizontal layers, reducing development time and costs.
- Reuse of software enhances quality and efficiency

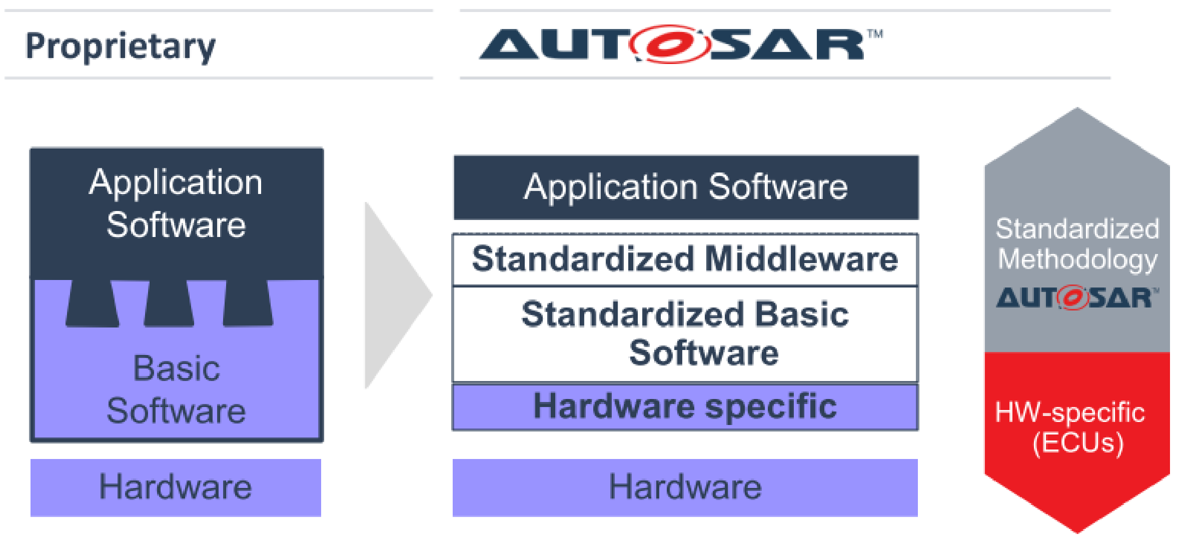


Figure 3.3-1 Autosar

The main goals of AUTOSAR are:

- Standardization
- Flexibility and Maintainability
- Availability
- Safety
- Scalability
- Software updates and upgrades
- Commercial of the shelf increased
- Improve quality
- Improve efficiency
- Preserving competitiveness

3.4 AUTOSAR Model View

Automotive ECUs having the following properties:

- Strong interaction with hardware (sensors and actuators)
- Connection to vehicle network via CAN, LIN or FlexRay
- Microcontrollers from 16 to 32 bit with limited resources of Flash and RAM (compared with Enterprise Solutions)
- Real Time Operating System
- Program execution from internal or external flash memory

AUTOSAR is based on the concept of distinction between Infrastructure (BSW) and Application (SWC), where the infrastructure includes all services for providing an execution environment to abstract HW details, instead the application is the vehicle function of interest.

Infrastructure: Services providing an execution environment to abstract HW details

Application: Vehicle function of interest

An application consists of interconnected Software Components, developed by different software suppliers, they are atomic and cannot be distributed over several ECUs.
The Software Components implementation is independent from the Infrastructure. The AUTOSAR software development is independent from the hardware, but not totally, in fact even if no knowledge is required about the ECU used or the network, the software developer needs to know about Sensors and/or Actuators in the system.

3.4.1 ECU Software Architecture

The ECU software architecture is based on a layered structure, in fact the software stack contains multiple layers, abstracted from the hardware.

In Figure 3.3.1 is possible to see the layered architecture:

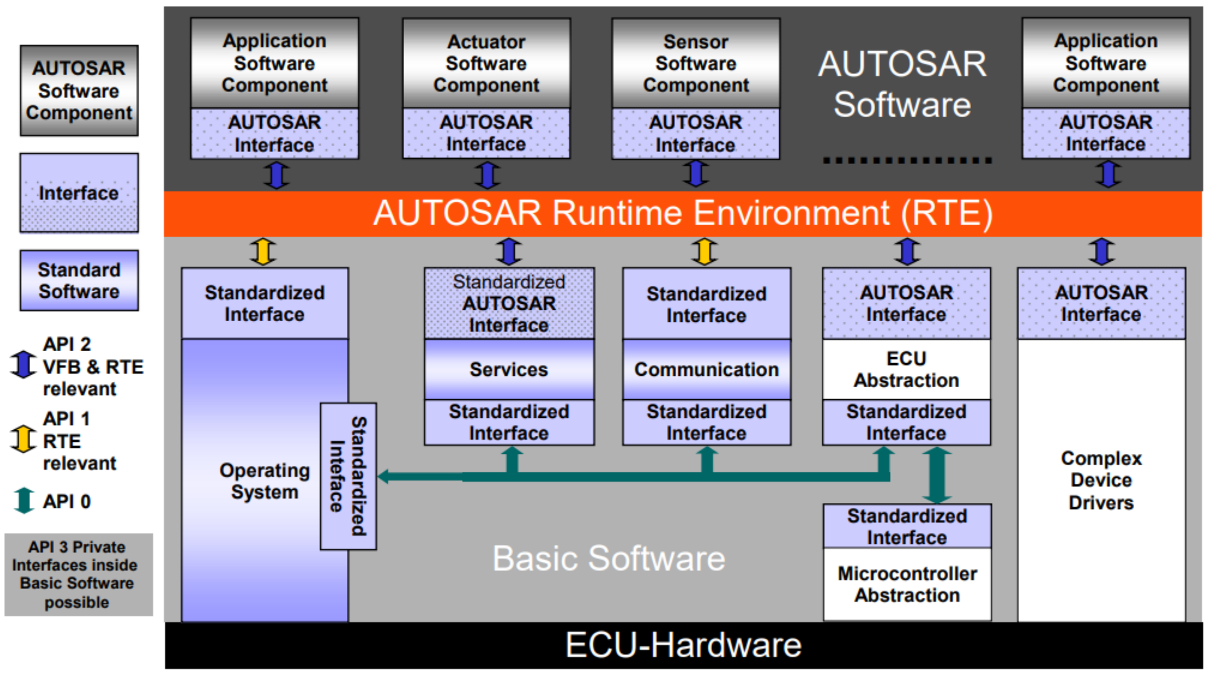


Figure 3.4-1: AUTOSAR Components [from http://www.autosar.org]

Figure 3.4.1: AUTOSAR Components [from http://www.autosar.org]

In which we have:

- Application layer
- AUTOSAR Runtime Environment (RTE)
- Basic Software Layer
- Hardware

The first two important layers are the **Application layer** and the **RTE**.

- The **Application Laye**r is the only layer that is not composed of standardized software, it is also the layer where the actual functionality is implemented. The layer is composed of Application Software Components that interact with the RTE.

- The **RTE** Environment implements a sequence of operations to establish the communication for inter and intra ECU information exchange, using the same interfaces and services for intra and for inter ECU communication. The RTE is able to understand when the communication go through shared memory, pipes or whatever inside the ECU or through one of the vehicle networks to get or deliver the data. The RTE is responsible even to trigger Runnables at events and to provide all the necessary functions for all the configured mechanism. The RTE is generally tool generated and statically configured.

It’s task is to make AUTOSAR Software Components independent from the mapping to a specific ECU.

- The **Basic Software Layer** can be subdivided, and is composed by:
    - Microcontroller Abstraction Layer
    - ECU Abstraction Layer
    - Service Layer

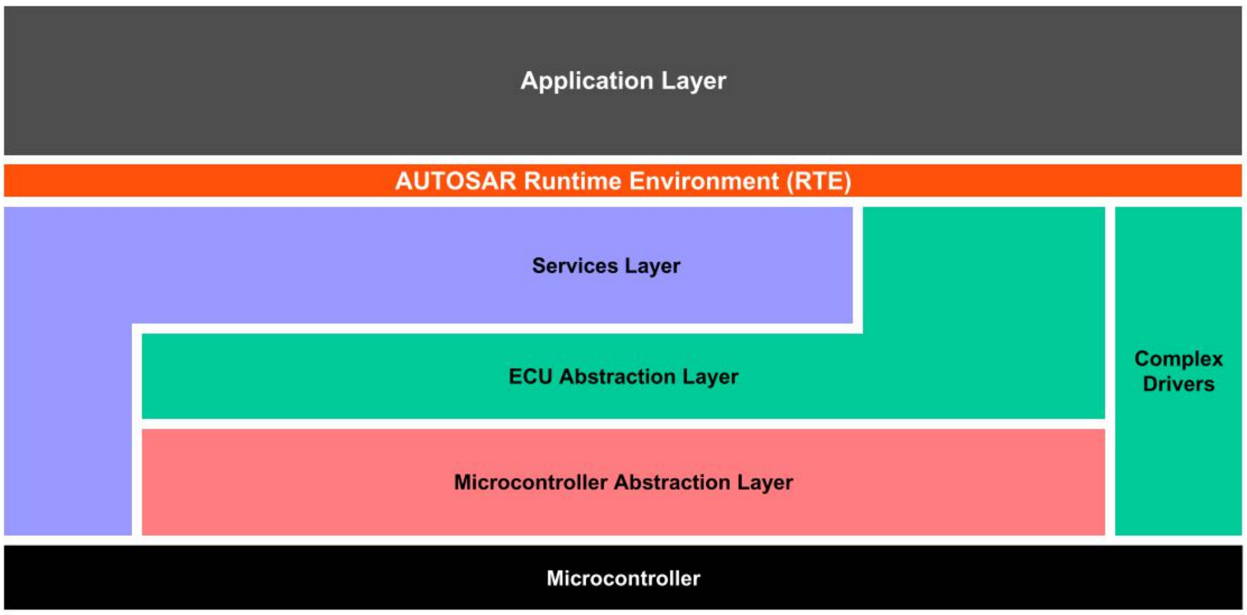


Figure 3.4-2 Basic software layer

- The **Microcontroller Abstraction Layer** contains the internal drivers and handle microcontroller specific operations to provide an abstraction of the microcontroller.
  It’s task is to Make higher software layers independent of μC

- The **ECU Abstraction Layer** interfaces the drivers of the Microcontroller Abstraction Layer. It also contains drivers for external devices. It offers an API for access to peripherals and devices regardless of their location (μC internal/external) and their connection to the μC (port pins, type of interface).
  It’s task is to Make higher software layers independent of ECU hardware layout

- The **Services Layer** is the highest layer of the Basic Software which also applies for its relevance for the application software: while access to I/O signals is covered by the ECU Abstraction Layer, the Services Layer offers:
    - Operating system functionality
    - Vehicle network communication and management services
    - Memory services (NVRAM management)
    - Diagnostic Services (including UDS communication, error memory and fault treatment)
    - ECU state management, mode management

  It’s task is to Provide basic services for application and basic software modules.

The Basic Software can be subdivided into the following types of services:

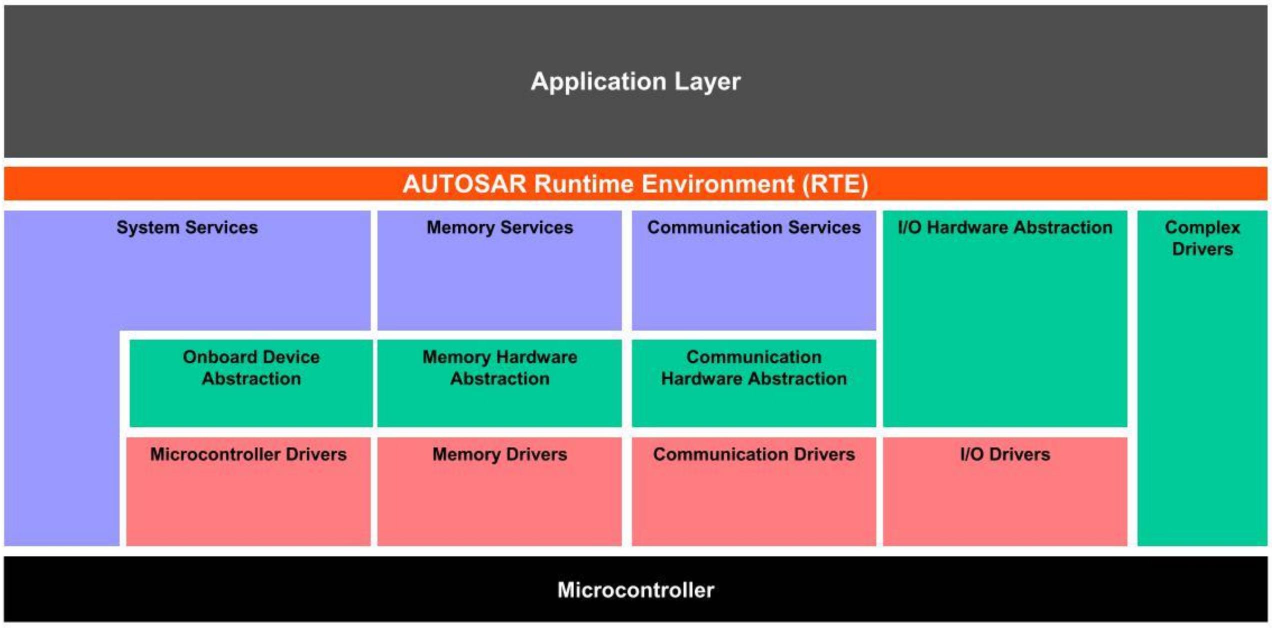


Figure 3.4-3 Basic software layer detailed

- **Microcontroller Abstraction Layer** : Abstract from specific microcontroller on the ECU and provides interfaces for ECU Abstraction layer.

    I/O drivers
    Communication drivers
    Memory drivers
    Microcontroller Drivers

- **ECU Abstraction Layer** : Abstract from the location of the controller

    I/O Abstraction Layer
    Communication HW Abstraction
    Memory HW Abstraction
    Onboard Device Abstraction

- **Service Layer** : Provides basic services for the application

    Communication Services
    Memory Services
    System Services

In this layered vision the developer do not need to know what is happening in the lower levels, using the interfaces provided by them.

It is possible to subdivide the Basic Software layer again in different stacks underlying the software functionality, and more precisely in five different stacks:

- **System stack** : Provides standardized access to services and operating system functions
- **Memory Management stack** : Provides standardized access to memory
- **Communication stack** : Provides standardized access to network system
- **I/O stack** : Provides standardized access to sensor or actuators or to others peripherals
- **Complex Driver stack** : Is used to bypass the basic software layer and to access directly to the hardware

3.5 Software Components

3.5.1 Data types

Before data object creation, it is necessary to create the correspondent data type, that defines the type of the data used.

They are divided into three different level:

- Application data types: Data type description used by the application
- Implementation data types: Data abstraction of the programming code
- Base type: Description of the primitive elements

3.5.2 Interfaces

The interfaces represent the protocol used by the Software Components to communicate through ports.

Different ports can share the same interface and there are different types of interfaces:

- Client/Server: The server provides the data and the client can invoke them.
- Sender/Receiver: The sender distributes the data and one or more receiver can gets them.
- Parameter Interface: For accessing to parameter data: volatile, constant or fixed
- Non volatile Data Interface: to access to non volatile data.

- Trigger Interface: To trigger the execution of another Software Component.
- Mode Switch Interface: To notify the mode to a Software Component.

3.5.3 Ports

As AUTOSAR application is an interconnection of Software Components, we need ports to provide connections.
The ports are the point of contact for a Software Component with others Software Components or with Basic Software modules and they can be:

- PPort (Provide port)
- RPort (Require port)

The PPort provides the service of the correspondent interface, instead the RPort requires the service described in the interface, and depending on the interface used there are different kind of ports:

PortType	InterfaceUsed
PPort	Sender/Receiver
RPort	Sender/Receiver
PPort	Client/Server
RPort	Client/Server
PPort	Parameter
RPort	Parameter
PPort	Sender/Receiver (Service)
RPort	Sender/Receiver (Service)
PPort	Client/Server (NV)
RPort	Client/Server (NV)
PPort	Trigger (Source)
RPort	Trigger (Sink)
PPort	Trigger (Source - Service)
RPort	Trigger (Sink - Service)
PPort	Mode Switch (Manager)
RPort	Mode Switch (User)
PPort	Mode Switch(Manager - Service)
RPort	Mode Switch(User - Service)

Table 3-1 AUTOSA Ports types

3.5.4 Software Components

Software Components are atomic and cannot be distributed over several ECUs.

The generation of the Software Components is not specified in the AUTOSAR standard, in fact it can be written or auto generated using a tool like Simulink.

The AUTOSAR standard is composed by many types of Software Components:

Table 3-2 SWC types in AUTOSAR

Name	Description
Application Software Component	Atomic Software Component that implements the vehicle function of interest (part of an application)
Sensor-Actuator Software Components	Atomic Software Component that acts like a sensor or an actuator, reading a sensor or setting the state of an actuator
Parameter Software Components	Provide parameters data (variable, fixed o const)
Composition Software Components	Encapsulates a collection of Software Components that communicate with the outside world using the delegation connectors
ECU-Abstraction Software Components	Provides access to the ECU IO capabilities, using a client/server PPort
Complex Device Driver Software Components	It is a generalization of the ECU-Abstraction Software Component, but it can defines ports for communication
NVBlock Software Components	Used for access to non volatile data from the other Software Components

In Figure 3.6 is shown a detailed view of the Software Component in VFB View and RTE View, where it is possible to see the difference between VFB and RTE, the mapping of the Software Components in the ECUs and the communication bus between the ECUs.

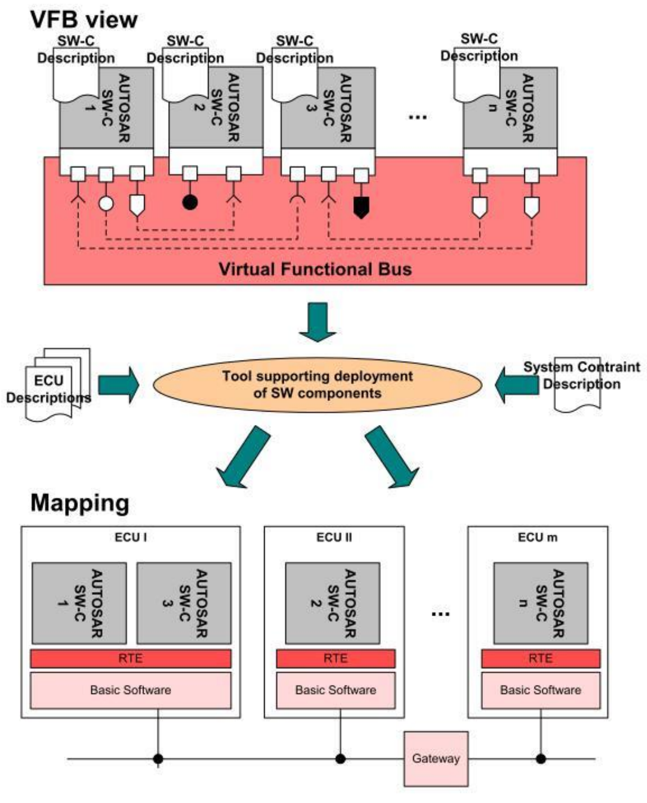


Figure 3.5-1 Software Components detailed view

3.5.5 *Software Components shipment*

A Software Component Shipment is a phase in which SWC will be released, consist of:

- SWC-implementation
- SWC-description

3.5.5.1SWC-implementation

Represents the source code or the object code that will be created.

This code is independent from the underlying hardware in which the component is mapped, it is AUTOSAR duty to provide a standardized view of the hardware to the Software Component.

The code is independent from other components location and from the number of times it will be instantiated in the system or in the same ECU.

Generally the programming language used for SWCs is C/C++ and inside there are all the functions that are mapped with the related Runnables Entities.

3.5.5.2 *SWC-description*

The SWC-description describes how the infrastructure should be configured for the Software Component.

There are three different Software Components description levels:

- VFB level: Communication properties and how the components are connected
- RTE level: Behavior of the single Software Components
- Implementation level: Behavior of each Runnable

3.5.6 Virtual Functional Bus

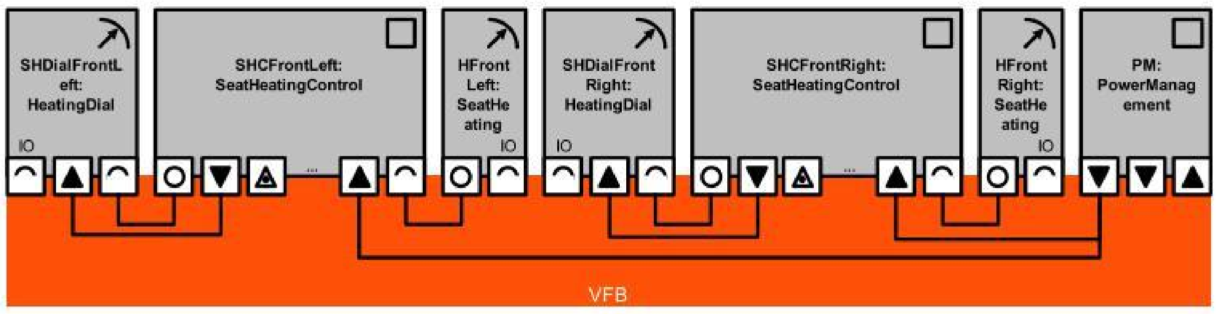

SHDialFrontLeft: HeatingDial
IO
SHCFrontLeft: SeatHeatingControl
HFront Left: SeatHeating
IO
SHDialFront Right: HeatingDial
IO
SHCFrontRight: SeatHeatingControl
HFront Right: SeatHeating
IO
PM: PowerManagement
VFB

Figure 3.5-2 VFB view

The Virtual Functional Bus is a virtual bus that contains the description of all the connection for all the components and how the components are connected between them.

It is just a virtual representation and by means of this, the Software Components can exchange data.

Is a logical entity that:

- Provides a virtual infrastructure independent from the underlying hardware
- Permits interactions between Software Components, using various services
- Separates the Application Layer from the Software Layer

3.5.7 Runtime Environment System (RTE)

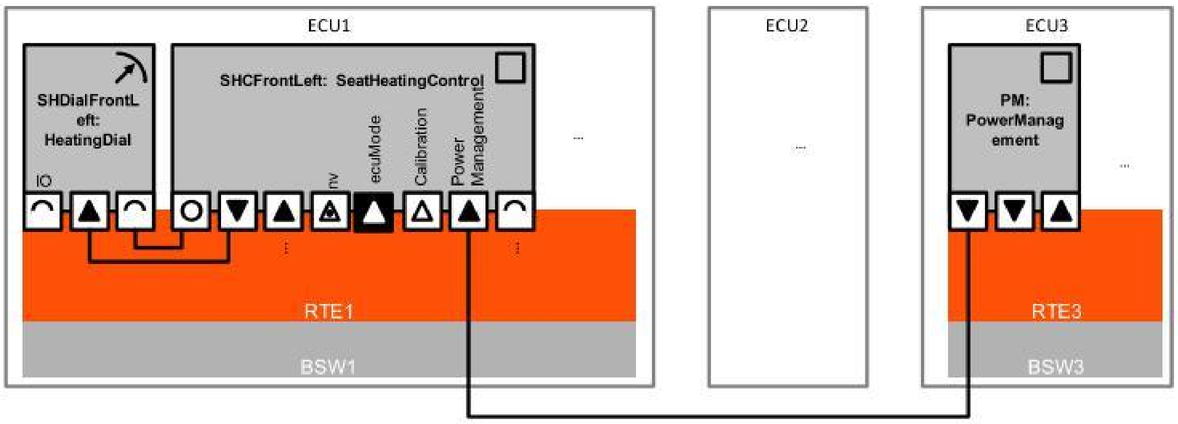


Figure 3.5-3 RTE view

The Runtime Environment System is the real communication implementation of the topology and Software Components interaction, for one single ECU.

Instead of having a single RTE for all the ECUs, like in VFB, there is one RTE customized for each ECU, which is generated during the ECU configuration phase.

One ECU can contain more than one component, so in that case if the components on the same ECU need to communicate they will use an intra-ECU communication, instead for communication between components on different ECU, an inter-ECU communication will be used, for instance a communication bus.

3.5.8 Runnables

Software Component behavior is provided by Runnables.

Runnables are the smallest part in an application, a sequence of instructions, and since the unit of execution in AUTOSAR is the OS task, they need to be mapped to an OS task to be executed, and this operation is done during the ECU Configuration phase.

As the SWCs do not have any direct interactions with the underlying hardware, no process or thread artifacts can exist.

An OS task can contain one or more Runnables and they can be executed concurrently.

The scheduling to execute Runnables is done by the OS scheduler, which decides which task can run on the ECU CPU during runtime.

The scheduling strategies adopted can be priority-based preemptive or round-robin or time triggered, etc.

For having a clear vision of the Runnables inside a SWC, imagine a SWC partitioned into threads where each of these is a Runnable.

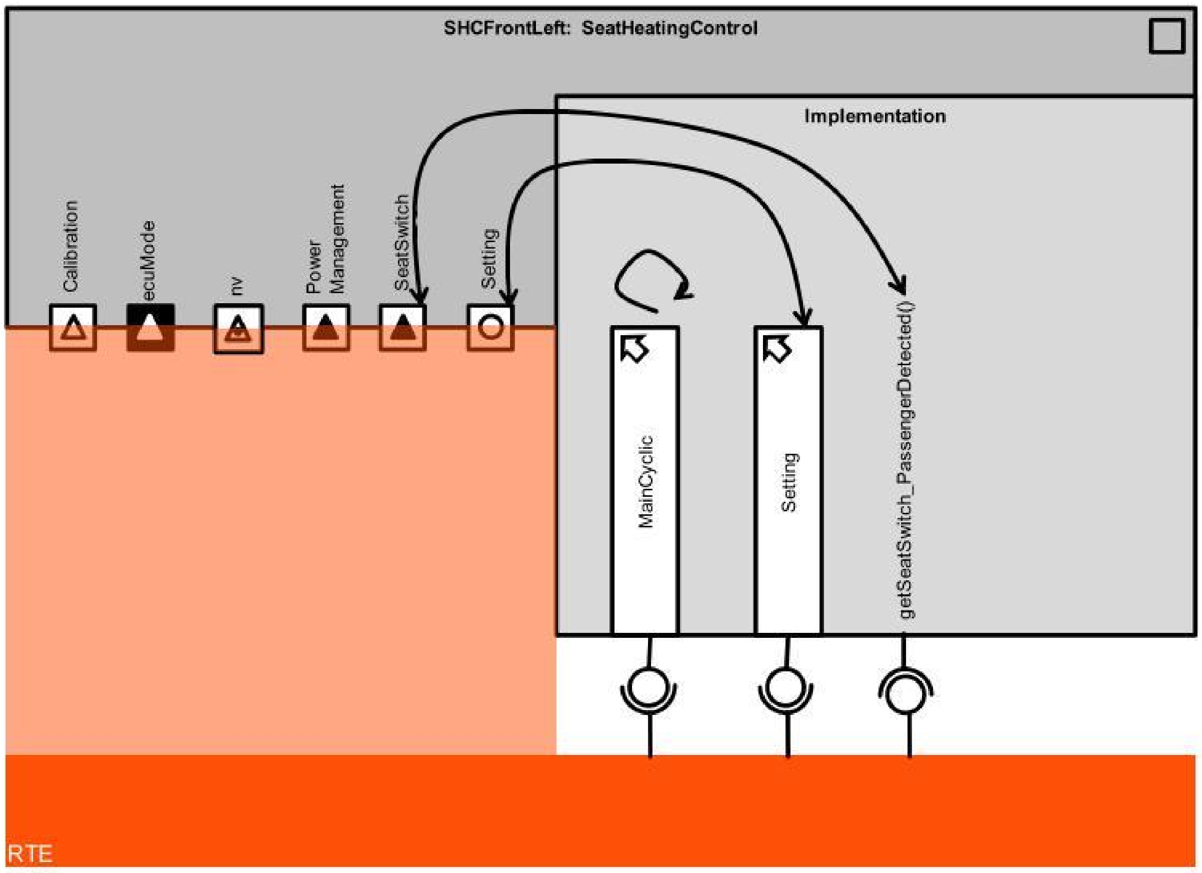


Figure 3.5-4 Runnables

The execution of the Runnable is invoked by the RTE through various different mechanisms:

- Fixed time schedule
- Events related to communication mechanisms
- Events related to physical occurrences

Each Runnable has a behavior that is defined, and the developer can create it manually or using external tools to generate automatically.

Generally the programming language used for Runnable behavior is C/C++.

Since the RTE is responsible for invoking Runnable to execute, some information are needed, which are the requirements for the RTE, that generally are:

- Runnables Event types
- other resources required by the component, such as local memory or AUTOSAR services

This requirements generally are present in the SWC-Description.

Since the RTE is responsible to invoke Runnables, they have to declare the type of defined events.

The events available in AUTOSAR are:

- TimingEvent
- DataReceivedEvent
- DataReceivedErrorEvent
- DataSendCompleteEvent
- OperationInvokedEvent
- AsynchronousServerCall Event
- ModeSwitchEvent

These events are capable of activating a Runnable instance and/or waking it up, that means that the instance has a waiting point for synchronization purposes.

In the generated code, each Runnable has a function, which has the behavior of the Runnable, for example in figure 3.5.7 where the function convGenInteger_run contains the behavior for a Runnable Entity.

```

#include <stdio.h>
#include "Rte_convGenInteger.h"


FUNC( void , RTE_APPL_CODE)
    convGenInteger_run ( P2CONST( struct Rte_CDS_convGenInteger ,
                                  AUTOMATIC,
                                  RTE_APPL_CONST) instance ){

    dt_SVX_Integer_c input;
    //get the input
    Rte_Read_rp_IfConv_op_set (instance, &input);

    //set the output
    Rte_Call_rp_in_ISVXCompSigGenDouble_op_set (instance, input);
}
```

Figure 3.5-5 Example of Runnable behavior code in C

Different aspects of AUTOSAR are involved in Runnable concepts, in fact, there is a different view for each of them: the OS view, the RTE view and the VFB view.

The OS view does not work directly with Runnables, because they are mapped into OS tasks.

These OS tasks can be scheduled by the OS scheduler with the chosen algorithm.

The RTE view is responsible for mapping Runnables into OS tasks.

RTE code manage Runnable inside the OS task.

The VFB view is the design Runnables time view, and in this view, Runnables are considered a sequence of instructions(execution or integration).

Mapping Runnables into OS tasks depend on many factors, like the event type (e.g. Timing or DataReceivedEvent), the type of Runnable and etc.

For example a Timing Event Runnable can be mapped to a single OS task instead of sharing the same OS task with other Runnables [2].

Since the AUTOSAR components are not allowed to communicate with ECU communication facilities at runtime, the RTE Source Code Generator provides an API.

The API syntax follow a general rule for the function name generator, that generally is:

[CommunicationType]_[PortType]_[Value](arg1, arg2, ..., argn)

(e.g. RTE_Read_SignalReading_Value(arg1)).

The available communication types can be:

- RTE_Read
- RTE_Write
- RTE_Invalidate
- RTE_Receive
- RTE_Send
- RTE_Feedback
- RTE_IRead
- RTE_IWrite

2 [6, Robert Warschofsky, AUTOSAR Software Architecture, Hasso-Plattner-Institute fur Softwaresys-temtechnik]

3.5.9AUTOSAR Methodology

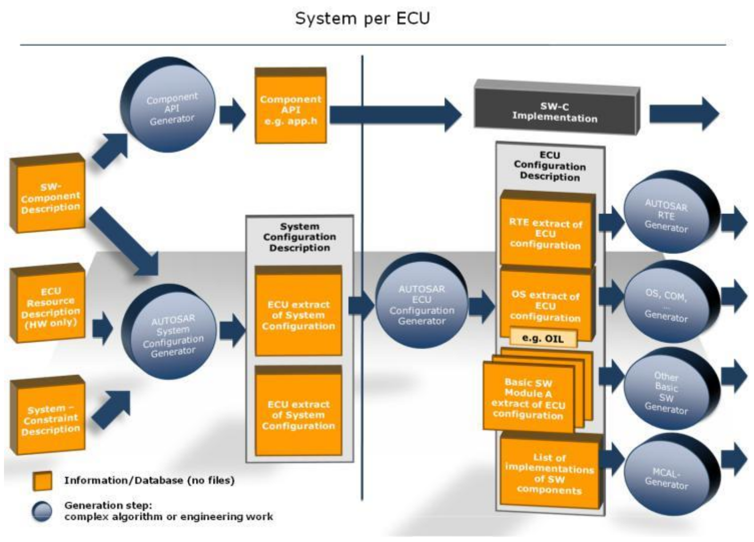

System per ECU
Component API Generator
Component API e.g. app.h
SW-C Implementation
SW-Component Description
ECU Resource Description (HW only)
System - Constraint Description
AUTOSAR System Configuration Generator
System Configuration Description
ECU extract of System Configuration
ECU extract of System Configuration
AUTOSAR ECU Configuration Generator
ECU Configuration Description
RTE extract of ECU configuration
OS extract of ECU configuration
e.g. OIL
Basic SW Module A extract of ECU configuration
List of implementations of SW components
AUTOSAR RTE Generator
OS, COM, ... Generator
Other Basic SW Generator
MCAL-Generator
Information/Database (no files)
Generation step: complex algorithm or engineering work

Figure 3.5-6 AUTOSAR Methodology [from http://www.autosar.org]

Up to know, there is not any common methodology to develop AUTOSAR systems.
The most common method is composed of four steps that starts from the software and hardware description to generating executable for each ECU.

The alternatives in methodology are in the ECU configuration and in the code generation.

The four main steps in the most common methodology are:

- Configure the system
- Extract ECU Specific Information
- Configure ECU
- Generate executable

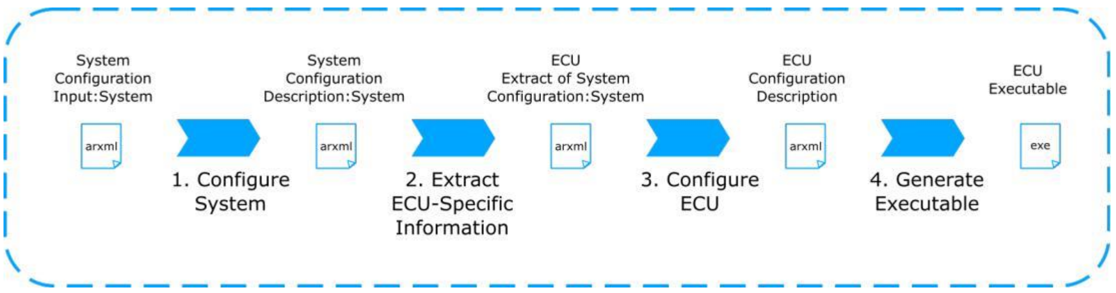

System Configuration Input:System
arxml
1. Configure System
System Configuration Description:System
arxml
2. Extract ECU-Specific Information
ECU Extract of System Configuration:System
arxml
3. Configure ECU
ECU Configuration Description
arxml
4. Generate Executable
ECU Executable
exe

Figure 3.5-7 AUTOSAR Methodology Steps

- The first step, “Configure the system”, is related to the whole system.

The input is an instance of the System template called System configuration input, that contains info about the Software Components, the hardware and about all the system constraints.

The output will be an instance of the System template which is called System Configuration Description that contains SWC to ECU mapping and information about bus topology.

- The next step is to Extract ECU-Specific information, that generates information for a specific ECU.

The Output will be the ECU Extract of System Configuration that is considered as an input for the next step Configure ECU for configuring RTE and Basic Software.

The output from this step will be used for building the ECU software, that will be generated in the next step Generate executable in which there will be code generation, code compilation and code linking phases.

4 VSx - Volcano (Mentor Graphics)

4.1 Principles of the Mentor VSx Tool Suite

- Design Process covered by a set of point tools, deployable step by step, according to customer's needs.
- Supporting:
  - early verification and late binding of solutions as opposed to the more common "early binding / late verification" style.
  - Real industrial development process
    - Parallel Processes
    - Iterative development
    - Distribution of Roles and Responsibilities in today's multi-company, modular supply chain
- Based on standards as AUTOSAR and EAST-ADL
- Enable shifting validation effort from physical prototypes to a virtual environment

4.2 Implementation aspects

- Based on the Eclipse framework
  - All VSx tools are Eclipse "plugins"
- The VSx infrastructure adds AUTOSAR awareness on top of Eclipse.
- Open interfaces to enable integration with other tools, thirdparty or customer specific.

4.3 VSx Platform

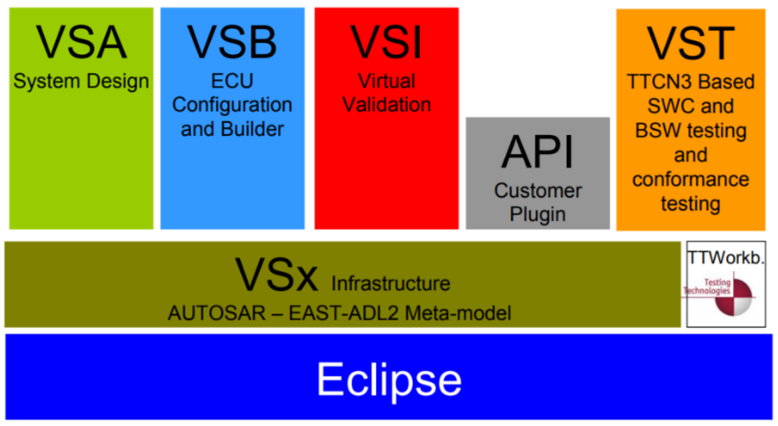

VSA
System Design
VSB
ECU Configuration and Builder
VSI
Virtual Validation
API
Customer Plugin
VST
TTCN3 Based SWC and BSW testing and conformance testing
VSx Infrastructure
AUTOSAR – EAST-ADL2 Meta-model
TTWorkb.
Testing Technologies
Eclipse

Figure 4.3-1 AUTOSAR Methodology Steps

VSx is a tool suite composed by:

- **VSA** (Volcano Vehicle System Architect): Authoring tool for design part

- **VSB** (Volcano Vehicle System Builder): ECU software configuration tool
- **VSA** Com Designer (Volcano VSA Communication): A network design tool
- **VSI** (Volcano Vehicle System Integrator): Simulation environment for software design
- **VSTAR** (Volcano VSTAR): AUTOSAR 4.0 basic software for ECU integration

Volcano VSI is a part of VSx, focusing on the simulation part, having the possibility of 3 validation levels: xml, model and code.

The main VSI goal is to provide the validation without using a real hardware, by the AUTOSAR RTE generated code.

The resultant application will be 100% AUTOSAR compliant and with this approach the developers can focus on developing the application instead of focusing on AUTOSAR software architecture rules and tools.

Generally the main procedure for creating an AUTOSAR application is to start with VSA for creating the architecture and then pass to VSI for:

- Set breakpoints on architecture elements
- Inspect data and control execution
- Build the system
- Other debugging features

While debugging the C/C++ code, it is even possible to step through the execution for inspecting purposes.

This tool family permits the developer to create an automotive architecture using AUTOSAR standard, starting from scratch or using other external tools, and then through the internal debugging system, validate the compliance of the system specifications.

There are other tools on the market with the same functionality, but this tool is focused on the debugging part and the most interesting part is in fact the possibility to run the developed software on a simulated RTE and OS, without the necessity to use the real hardware.

To do that we can even use the build model with some external tools, for example it is possible to use Matlab/Simulink or Labview, or even compiled applications developed in programming languages like C/C++ and others, for providing stimuli for the model.

The AUTOSAR application is composed of Software Components interconnected between them, and each Software Component has a behavior.

For defining this behavior there are two main ways:

1. Writing the code manually.
2. Using external tools.

In this thesis, both approaches are tested and used.

In the first method the developer has the full control of the system, and the simpler the system is, the better this solution works.

The other solution is the Model Based Design approach in which even complex system can be simply developed using the graphical editor, offered by the tool, and the code will be generated automatically by the tool.

Sometimes the generated code is difficult to read, and if some changes are required it is necessary to do the whole process again (modifying the model in the external tool and importing back again in the AUTOSAR architecture system).

This is only the first part of the whole workflow with Volcano, in fact next steps are:

- Defining the topology of a network
- Mapping the Runnables to the correspondent OS tasks
- Proceed with the integration part in the real hardware.

But in this thesis I have focused only on the creation of the architecture and on debugging and validate the system, using not all the tools in the VSx family, but only VSA for authoring and VSI for simulation and debugging/testing tool.

4.4 VSI (Virtual Systems Integrator)

The main goals of VSI are:

- Validate SWC/composition behavior at VFB level
- Design verification and validation of embedded software from multiple sources in distributed systems
- AUTOSAR is initial target
- IDE, debugger and profiler in AUTOSAR environment, etc.

4.4.1 Why an AUTOSAR System Level Simulator?

- AUTOSAR SWCs coming from different sources must be integrated to perform vehicle functionality
- Allows verification of SWC compositions long before ECUs are available
- Models use code generated from BridgePoint UML, Simulink, etc. or hand coding.

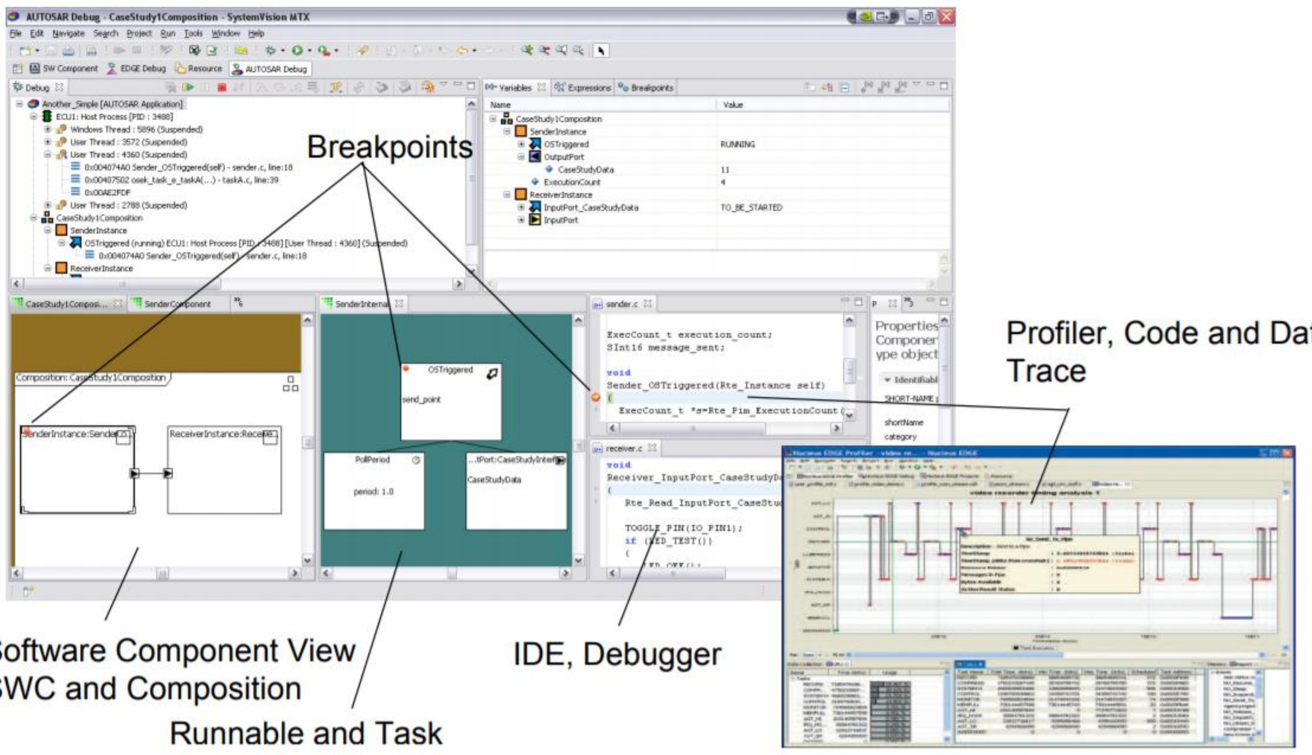


Figure 4.4-1 VSI Tool Suite

4.5 VSx Workow

Generally AUTOSAR software development workflow is:

1. Creating an AUTOSAR architecture
    a) from scratch (using VSA)
    b) from existing model/code
2. Then pass the AUTOSAR system (application) description to VSI, that automatically map it to an ECU and creates the required RTE, OS and BSW. In the end it will generate an executable application.

After creating the described software system the VSI workflow can starts, and in the end an executable will be generated that can be execute in VSI or even in a standalone build like in an interactive console or in a batch.

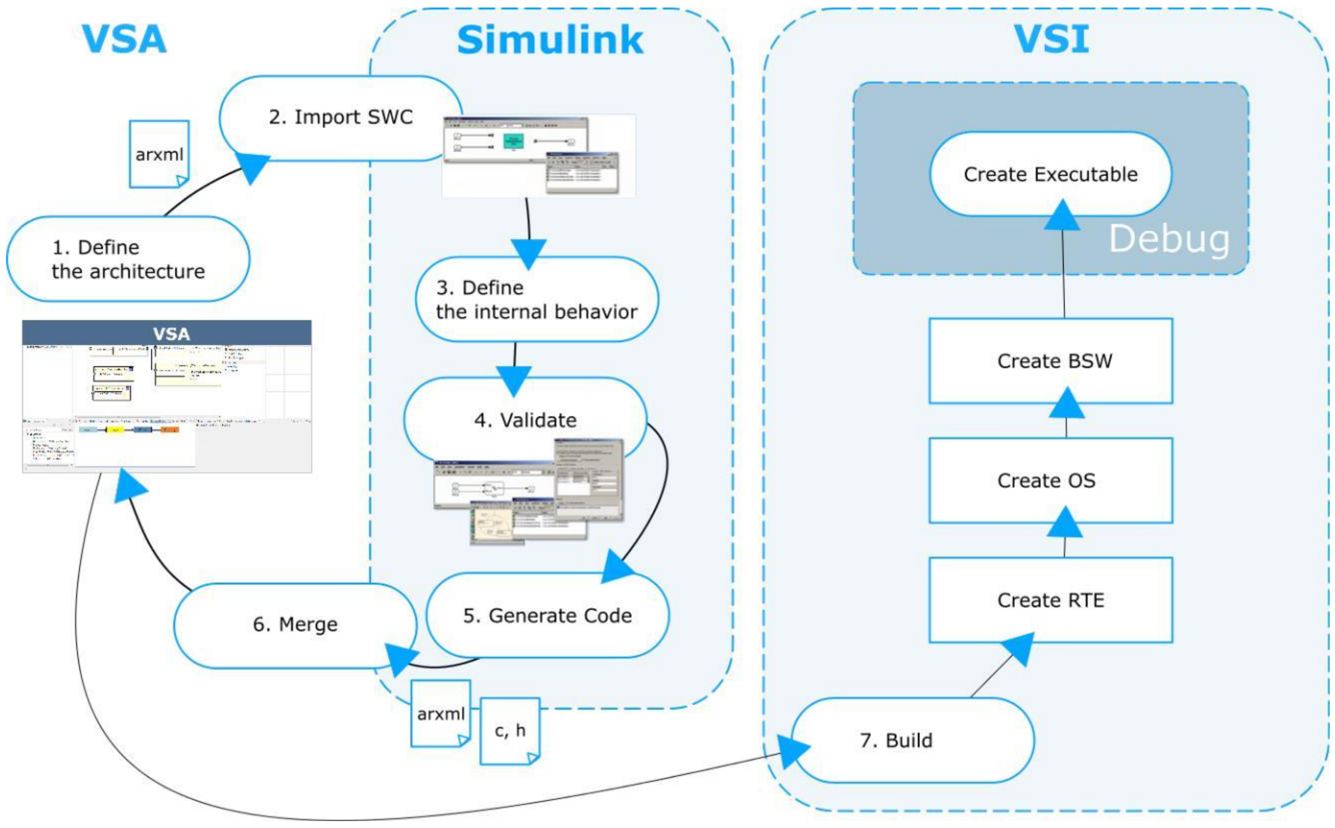


Figure 4.5-1 VSx workow

VSI uses some builders during all the workflow:

- VSI ECU Extract Builder
- VSI Default ECU Configuration Builder
- VSI RTE Builder
- VSI OS Builder.

The VSI ECU Extract takes as input the generated system (AUTOSAR files and generated code), and generates the ECU Extract with information about mapping of the software system.

The VSI Default ECU Configuration Builder takes the ECU Extract and returns the ECU Configuration.

Next is the RTE Builder that using the RTE Generator and taking the ECU Configuration as input in the end produces a VFB.

The last step is the OS Builder using the OS Generator for producing an Operating System for execution of AUTOSAR application.

4.6 VSA

VSA is the authoring tool and give permission to:

- Create the architecture of the system in terms of Software Components and connections between them.

- Define data types, ports, interfaces and to assign those element to the right Soft-ware Component for creating the AUTOSAR application.

The architecture created with VSA is only a skeleton, and it lacks the behavior.

In this skeleton there are only a set of components with ports and correspondent interfaces associated and connected between them.

4.7 Using VSI with conneXion

conneXion is a tool that allows the communications between different models and/or applications (e.g. Java, C/C++) and giving the permission to use external models with the VSI generated models.

The external model can be a sequencer or an acceptor, and communicates with the VSI model during execution.

For adding the conneXion Software Components to the VSI model we need the Configuration phase to define the below information:

- Channel name
- Signal name

These Software Components are all inside an ARPackage (**svx_connexion**) available in VSx, and the developer can use it by importing the package, or starting a new Project in VSI and selecting the correspondent item.

With conneXion is possible even to stop the execution of one model, for example the Simulink model, change something, and then start again the execution with the changes for viewing the results.

conneXion is available to use with different external software like Matlab Simulink or Labview.

In this thesis the focus is on conneXion for Simulink.

4.7.1 conneXion for Simulink

conneXion as a tool that allows the communications between different models, in this case we use conneXion for communication between a model in VSX and other one in Simulink (the Plant).

It works like a virtual communications bus between parts, as we can see in the example in figure 4.4 where the Simulink model is a Sequencer and the VSx model is an Acceptor.

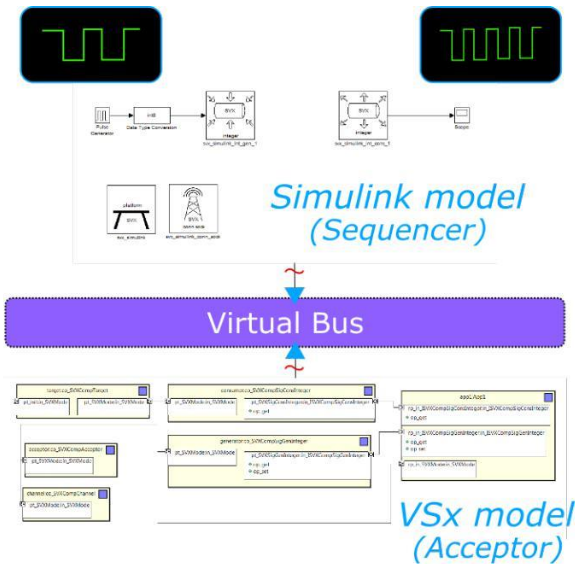


Figure 4.7-1 Connexion communications

Using conneXion for Simulink is simple, the developer just need to use the package present in the Simulink library named SystemVision SVX.
See 7 for more info.
The data types available for use in the SVX library are:

- Boolean
- Integer
- Real
- Unsigned

It is possible to use these data types for both block types: Consumer and Generator. For using the SVX blocks in a model it is necessary to configure them, and in particular the Channel, the Communications block (Acceptor or Connector) and the SVX block to configure the role of the model whether it is a Sequencer or not.

5 The Starting Model

5.1 General view

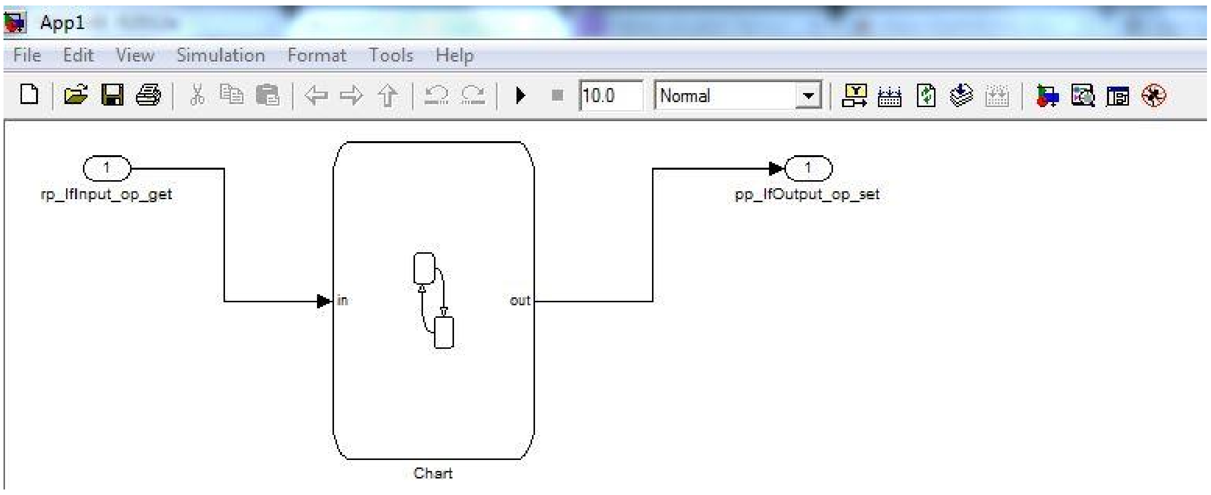


Figure 5.1-1 Starting model

As explained in the previous chapters, the work of this thesis is to implement an AUTOSAR version of a very simple filter, and comparing the testing results between the two implementations.

The starting model is created using Simulink with Model Based Design Technique.

6 AUTOSAR Architecture creation using Simulink

6.1 Implementation choices

To create a model using AUTOSAR architecture, it is important to decide in which method you want to create the skeleton of the model.

In this case the chosen tool for building the architecture are VSA and Simulink. either Simulink or VSA can be used as starting point.

- Start on Simulink
- Start on VSA

The created AUTOSAR architecture is based on the model discussed in the previous chapter which is created in Simulink.

To determine the best way to convert this model to AUTOSAR architecture, a precise analysis of elements we need to use is required, for instance the type of Software Components, interfaces, ports etc.

The goal is to make the architecture similar to the starting model, for reusing the modeled system already created.

Since the starting model is only one system, the decision taken was to maintain this structure and create one application Software Components in AUTOSAR that will have the same behavior as the subsystem.

In this way there is no necessity to redo the behavior for the defined Software Component in AUTOSAR.

The same for the used data types, because having the same behavior it comports to have the same input and output ports.

In this case the starting model was using only the uint8 type.

For the interfaces, instead, the solution adopted is to use the sender/receiver type, because the starting system is event driven.

Then the ports used were PPort for outputs and RPort for inputs.

In figures 6.5.1 is shown a general overview of the architecture created in AUTOSAR using Volcano VSx, in which in the end we have a Root Software Component that contains the two Software Components that are the core of the system: Lock/Unlock (6.5.3) and Protocol (6.5.2).

6.1.1 Start on Simulink

Potentially it is possible to start creating a model using Simulink, and defining the behavior of the Application Software Components, using a Model Based Software Design approach and then importing them in VSx.

Simulink can do an initial validation on the model for checking all the AUTOSAR constraints, and can auto generate the code for the behavior of the SWC using the chosen code like for example C or C++.

This kind of approach seems simple, but there is one thing that must be taken in consideration, not all the Simulink blocks can be exported correctly in AUTOSAR, there could be some incompatibilities and in that case it will be necessary to re-analyze the system and to exchange the incompatible block with others.

The validation of the model before the generation of the AUTOSAR code, is extremely useful in which allows the developer to check if the model follows the AUTOSAR specifications and if all the used components are fully compatible.

This work is done completely by Simulink through an interface that shows useful information.

6.1.2 Start on VSA

The chosen solution, instead is to use VSA for creating the starting model, to avoid possible incompatibility problems due to any Simulink block not fully compatible with AUTOSAR standard.

This method is more difficult than the other one, in which there are more steps to do and the procedure is longer, but in the end the result will be a fully compatible model.

The procedure can be divided in two main steps:

1. Creating the skeleton of the model, in which all the needed Data Types, Interfaces, Ports, Software Components and everything related to the main structure of the model will be defined.

2. Defining all the necessities for the right functionality for vehicle function of interest.

6.1.2.1 Exporting From VSx

Once the architecture skeleton is created, it is ready to be exported to Simulink and defining the behavior.

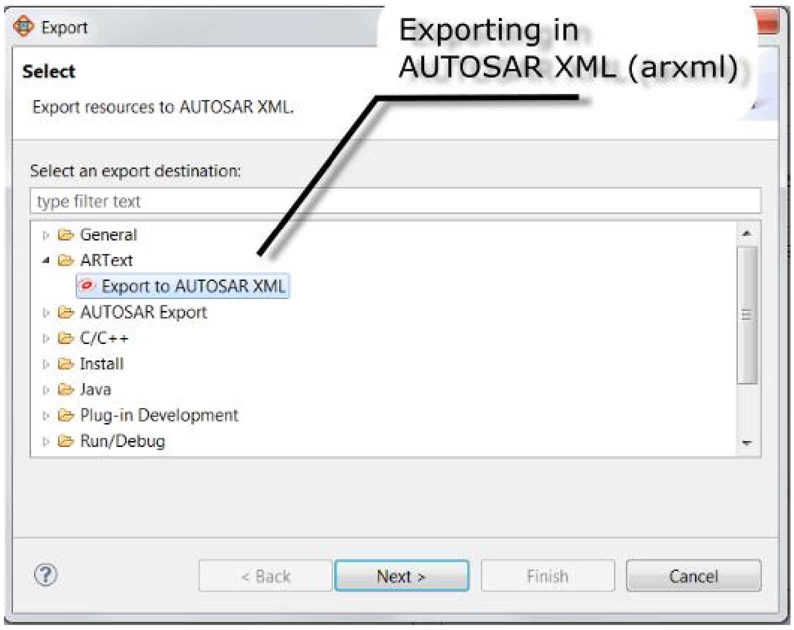
To export the skeleton model in VSx we need to have the data in arxml format, that is an AUTOSAR file format which contains the whole architecture description, in fact it is possible to export the data in arxml files for one or more ARPackages.

An ARPackage contains the description about Data Types, Software Components, Interfaces, Ports, etc.

These files can be imported back in Matlab using the integrated functions for importing the arxml files and the relatives dependencies [1], for recreating the model skeleton in which will be added the behavior.

[1] http://it.mathworks.com/help/rtw/examples/import-and-export-an-autosar-software-component.html

The arxml file is an xml file and it is possible to do an early validation on it before exporting it, for checking the compatibility with the AUTOSAR standard.



Export

Select

Export resources to AUTOSAR XML.

Select an export destination:

type filter text

- General
- ARText
  - Export to AUTOSAR XML
- AUTOSAR Export
- C/C++
- Install
- Java
- Plug-in Development
- Run/Debug

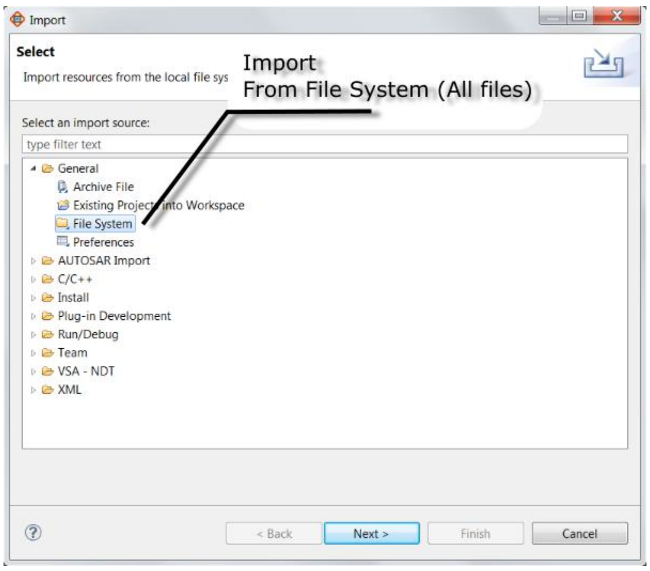
< Back | Next > | Finish | Cancel

Figure 6.1-1 ARPackage exporting from VSI

6.1.2.2 Importing back in VSx

After creating the behavior and the code validation, it is possible to export the code again, using the automated code generation (e.g. using Simulink).

Several files will be generated, like: new arxml files, C or C++ files, and they can be imported back in VSx to integrate them in the previous project.



Import

Select

Import resources from the local file sys

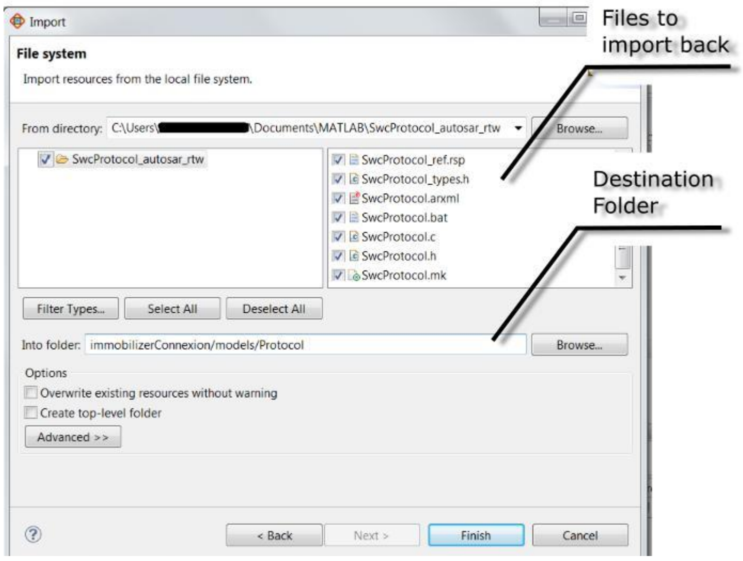
Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- AUTOSAR Import
- C/C++
- Install
- Plug-in Development
- Run/Debug
- Team
- VSA - NDT
- XML

< Back | Next > | Finish | Cancel

Figure 6.1-2 ARPackage importing from external source



Import

File system

Import resources from the local file system.

From directory: C:\Users\[redacted]\Documents\MATLAB\SwcProtocol_autosar_rtw

- [✓] SwcProtocol_autosar_rtw

- [✓] SwcProtocol_ref.rsp
- [✓] SwcProtocol_types.h
- [✓] SwcProtocol.arxml
- [✓] SwcProtocol.bat
- [✓] SwcProtocol.c
- [✓] SwcProtocol.h
- [✓] SwcProtocol.mk

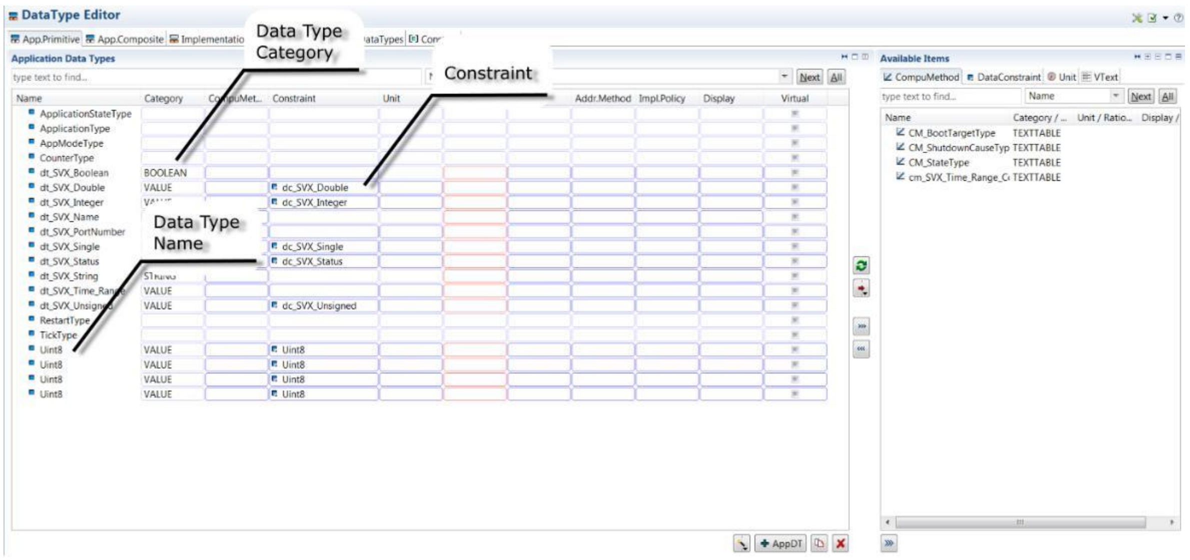
Into folder: immobilizerConnexion/models/Protocol

Options
- [ ] Overwrite existing resources without warning
- [ ] Create top-level folder

Figure 6.1-3 ARPackage importing from external source

6.2 Defining Data Types

Initially it is necessary to study the model to find out which and how many inputs and outputs the system needs, and after that it is possible to create the data types needed with the relative constraints.



Application Data Types

Name	Category	CompuMet...	Constraint	Unit	Addr.Method	Impl.Policy	Display	Virtual
ApplicationStateType								
ApplicationType								
AppModeType								
CounterType								
dt_SVX_Boolean	BOOLEAN							
dt_SVX_Double	VALUE		dc_SVX_Double					
dt_SVX_Integer	VALUE		dc_SVX_Integer					
dt_SVX_Name								
dt_SVX_PortNumber								
dt_SVX_Single			dc_SVX_Single					
dt_SVX_Status			dc_SVX_Status					
dt_SVX_String	STRING							
dt_SVX_Time_Range	VALUE							
dt_SVX_Unsigned	VALUE		dc_SVX_Unsigned					
RestartType								
TickType								
Uint8	VALUE		Uint8					
Uint8	VALUE		Uint8					
Uint8	VALUE		Uint8					
Uint8	VALUE		Uint8					

Available Items

CompuMethod | DataConstraint | Unit | VText

Name	Category / ...	Unit / Ratio...	Display /
CM_BootTargetType	TEXTTABLE		
CM_ShutdownCauseTyp	TEXTTABLE		
CM_StateType	TEXTTABLE		
cm_SVX_Time_Range_C	TEXTTABLE		

Figure 6.2-1 Data types Editor

The data types used in this model, using as reference the uint8 data type, and all inputs and outputs are of the same type with the constraint: Physical [0,255].

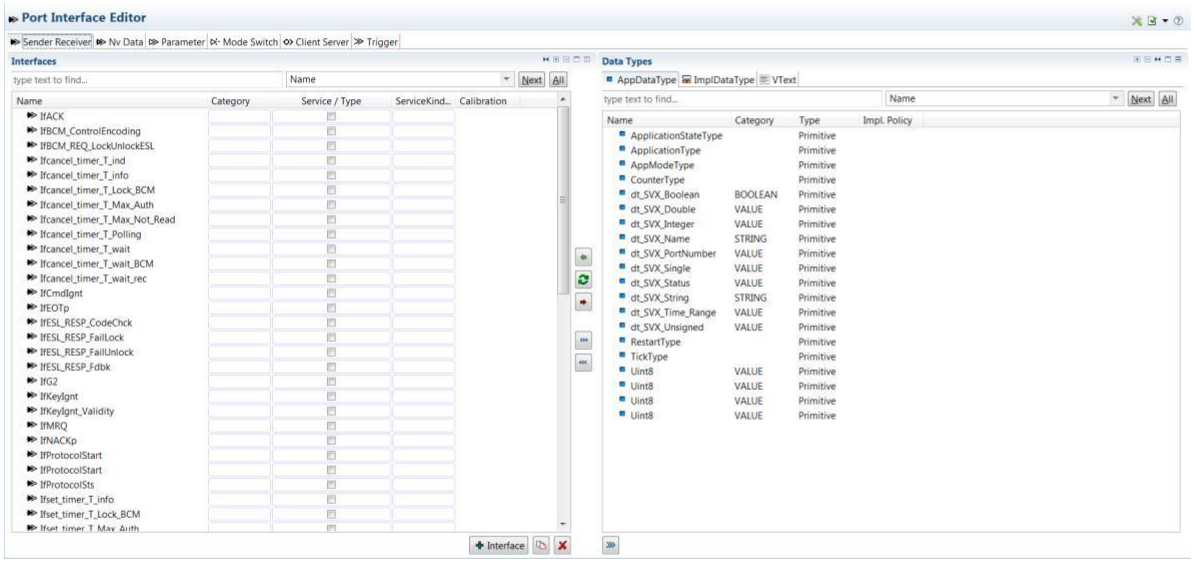
For implementing that in Volcano, it is necessary to open the SWC Architecture perspective and then go to the DataType Editor for adding the new data type.

The important values to set for data types are the Category and the Constraints that characterize the data.

In the DataType Editor are shown all the data types available for the entire project. For using them a Data Type Mapping Set must be created with the associations
between the Application Data Types and the Implementation Data Types.

6.3 Defining Interfaces

It is necessary even to define the interfaces to determine the data types that will be pass through the various ports.



Name	Category	Service / Type	ServiceKind...	Calibration
IfACK		☐		
IfBCM_ControlEncoding		☐		
IfBCM_REQ_LockUnlockESL		☐		
Ifcancel_timer_T_ind		☐		
Ifcancel_timer_T_info		☐		
Ifcancel_timer_T_Lock_BCM		☐		
Ifcancel_timer_T_Max_Auth		☐		
Ifcancel_timer_T_Max_Not_Read		☐		
Ifcancel_timer_T_Polling		☐		
Ifcancel_timer_T_wait		☐		
Ifcancel_timer_T_wait_BCM		☐		
Ifcancel_timer_T_wait_rec		☐		
IfCmdIgnt		☐		
IfEOTp		☐		
IfESL_RESP_CodeChck		☐		
IfESL_RESP_FailLock		☐		
IfESL_RESP_FailUnlock		☐		
IfESL_RESP_Fdbk		☐		
IfG2		☐		
IfKeyIgnt		☐		
IfKeyIgnt_Validity		☐		
IfMRQ		☐		
IfNACKp		☐		
IfProtocolStart		☐		
IfProtocolStart		☐		
IfProtocolSts		☐		
Ifset_timer_T_info		☐		
Ifset_timer_T_Lock_BCM		☐		
Ifset_timer_T_Max_Auth		☐		

Name	Category	Type	Impl. Policy
ApplicationStateType		Primitive	
ApplicationType		Primitive	
AppModeType		Primitive	
CounterType		Primitive	
dt_SVX_Boolean	BOOLEAN	Primitive	
dt_SVX_Double	VALUE	Primitive	
dt_SVX_Integer	VALUE	Primitive	
dt_SVX_Name	STRING	Primitive	
dt_SVX_PortNumber	VALUE	Primitive	
dt_SVX_Single	VALUE	Primitive	
dt_SVX_Status	VALUE	Primitive	
dt_SVX_String	STRING	Primitive	
dt_SVX_Time_Range	VALUE	Primitive	
dt_SVX_Unsigned	VALUE	Primitive	
RestartType		Primitive	
TickType		Primitive	
Uint8	VALUE	Primitive	
Uint8	VALUE	Primitive	
Uint8	VALUE	Primitive	
Uint8	VALUE	Primitive	

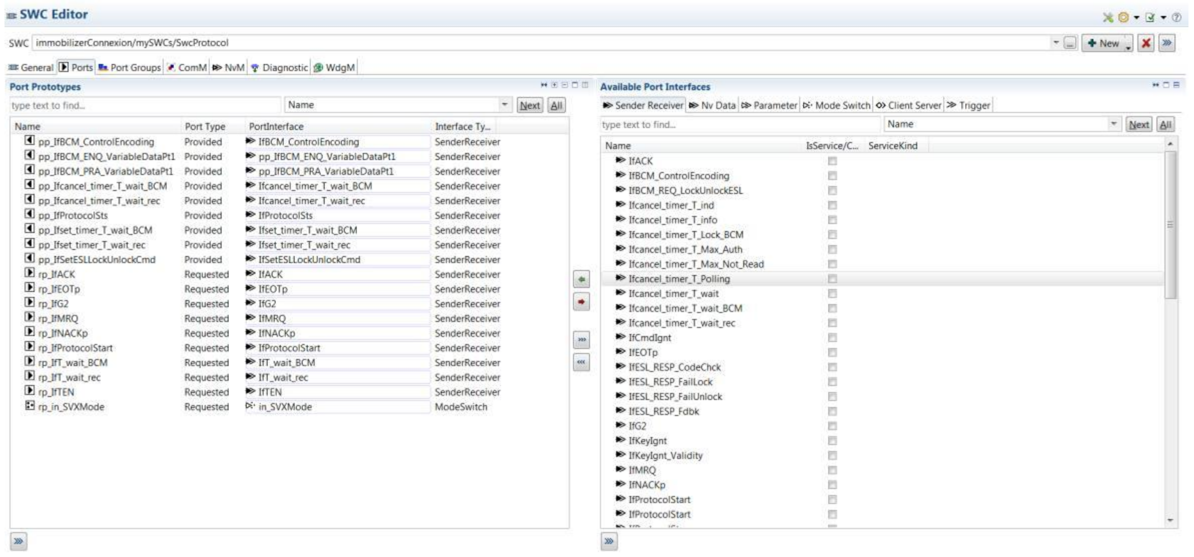
Figure 6.3-1 Port Interface Editor

In my solution I have used many different kinds of interfaces (sender/receiver).

6.4 Defining Ports

The ports are the Software Components point of contact with the outside and it allows the component to send/receive data.

In this model sender/receiver ports are used.



SWC immobilizerConnexion/mySWCs/SwcProtocol

Port Prototypes

Name	Port Type	PortInterface	Interface Ty...
pp_IfBCM_ControlEncoding	Provided	IfBCM_ControlEncoding	SenderReceiver
pp_IfBCM_ENQ_VariableDataPt1	Provided	pp_IfBCM_ENQ_VariableDataPt1	SenderReceiver
pp_IfBCM_PRA_VariableDataPt1	Provided	pp_IfBCM_PRA_VariableDataPt1	SenderReceiver
pp_Ifcancel_timer_T_wait_BCM	Provided	Ifcancel_timer_T_wait_BCM	SenderReceiver
pp_Ifcancel_timer_T_wait_rec	Provided	Ifcancel_timer_T_wait_rec	SenderReceiver
pp_IfProtocolSts	Provided	IfProtocolSts	SenderReceiver
pp_Ifset_timer_T_wait_BCM	Provided	Ifset_timer_T_wait_BCM	SenderReceiver
pp_Ifset_timer_T_wait_rec	Provided	Ifset_timer_T_wait_rec	SenderReceiver
pp_IfSetESLLockUnlockCmd	Provided	IfSetESLLockUnlockCmd	SenderReceiver
rp_IfACK	Requested	IfACK	SenderReceiver
rp_IfEOTp	Requested	IfEOTp	SenderReceiver
rp_IfG2	Requested	IfG2	SenderReceiver
rp_IfMRQ	Requested	IfMRQ	SenderReceiver
rp_IfNACKp	Requested	IfNACKp	SenderReceiver
rp_IfProtocolStart	Requested	IfProtocolStart	SenderReceiver
rp_IfT_wait_BCM	Requested	IfT_wait_BCM	SenderReceiver
rp_IfT_wait_rec	Requested	IfT_wait_rec	SenderReceiver
rp_IfTEN	Requested	IfTEN	SenderReceiver
rp_in_SVXMode	Requested	in_SVXMode	ModeSwitch

Available Port Interfaces

Name	IsService/C...	ServiceKind
IfACK	☐	
IfBCM_ControlEncoding	☐	
IfBCM_REQ_LockUnlockESL	☐	
Ifcancel_timer_T_ind	☐	
Ifcancel_timer_T_info	☐	
Ifcancel_timer_T_Lock_BCM	☐	
Ifcancel_timer_T_Max_Auth	☐	
Ifcancel_timer_T_Max_Not_Read	☐	
Ifcancel_timer_T_Polling	☐	
Ifcancel_timer_T_wait	☐	
Ifcancel_timer_T_wait_BCM	☐	
Ifcancel_timer_T_wait_rec	☐	
IfCmdIgnt	☐	
IfEOTp	☐	
IfESL_RESP_CodeChck	☐	
IfESL_RESP_FailLock	☐	
IfESL_RESP_FailUnlock	☐	
IfESL_RESP_Fdbk	☐	
IfG2	☐	
IfKeyIgnt	☐	
IfKeyIgnt_Validity	☐	
IfMRQ	☐	
IfNACKp	☐	
IfProtocolStart	☐	
IfProtocolStart	☐	

Figure 6.4-1 Defining ports using Software Component Editor for Protocol SWC

6.5 Defining Software Components

The Software Components used in this project are Composition and Application type.

The Application SWCs contain the behavior, that is part of the logic of the application. The Composition Software Components are like a container for all the other types of SWCs.

6.5.1 Composition Software Component

The model is divided into different Composition Software Components, following some screenshots:

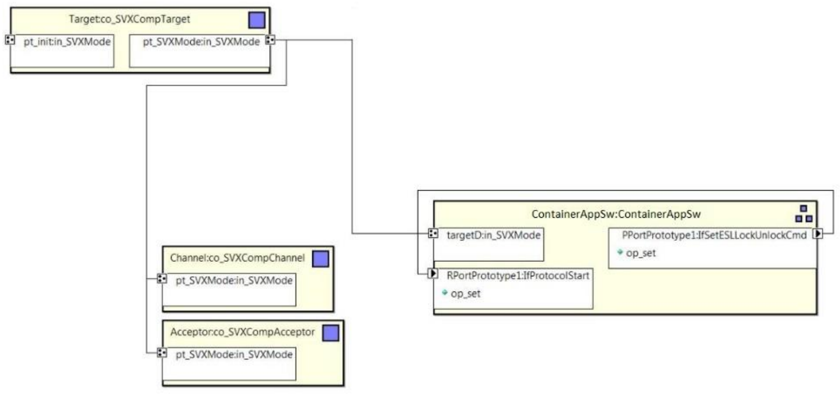

Target:co_SVXCompTarget
pt_init:in_SVXMode
pt_SVXMode:in_SVXMode
ContainerAppSw:ContainerAppSw
targetD:in_SVXMode
PPortPrototype1:IfSetESLLockUnlockCmd
op_set
RPortPrototype1:IfProtocolStart
op_set
Channel:co_SVXCompChannel
pt_SVXMode:in_SVXMode
Acceptor:co_SVXCompAcceptor
pt_SVXMode:in_SVXMode

Figure 6.5-1 Root Composition Software Component

The Root Composition Software Component contains not only the Composition SWCs, but also some other SWCs necessary for conneXion, like the Target, the Channel and the Acceptor, because in this case the Plant in Simulink is the sequencer.

This Composition SWC contains the main Application Software Component Protocol, in which most of the inputs/outputs are connected to others Composition Software Components that contain the conneXion Consumers and Generators.

These Consumers and Generators are connected through a delegation to the Target in the upper level.

Different kind of delegations are connected to the associated ports, for input/output, to make the communication between two Application Software Component.

6.5.2 Applications Software Components

In this project different Application Software Components are used.

Other SWCs are used for example to handle the port connection between the conneXion SWCs and the other SWCs.

In the next picture a collection of some Application Software Components used are shown.

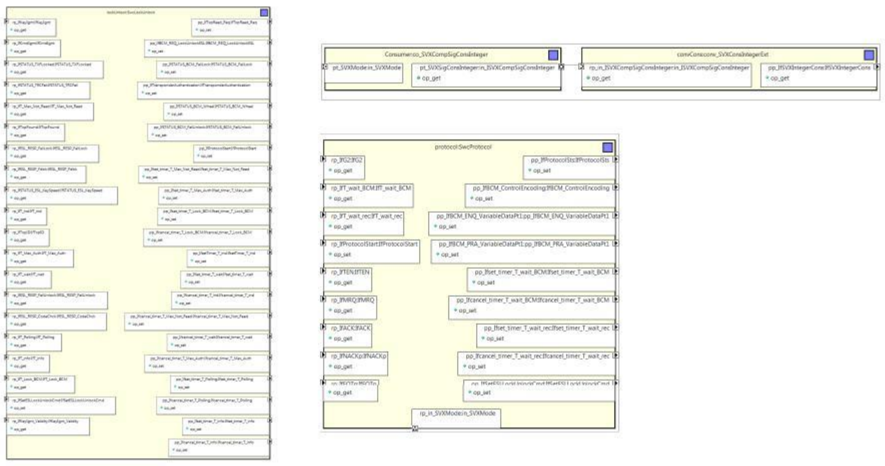

Consumer:co_SVXCompSigConsInteger
pt_SVXMode:in_SVXMode
pt_SVXSigConsInteger:in_ISVXCompSigConsInteger
op_get
conv:Consconv_SVXConsIntegerExt
rp_in_ISVXCompSigConsInteger:in_ISVXCompSigConsInteger
pp_IfSVXIntegerCons:IfSVXIntegerCons
protocol:SwcProtocol
rp_IfG2:IfG2
rp_IfT_wait_BCM:IfT_wait_BCM
rp_IfT_wait_rec:IfT_wait_rec
rp_IfProtocolStart:IfProtocolStart
rp_IfTEN:IfTEN
rp_IfMRQ:IfMRQ
rp_IfACK:IfACK
rp_IfNACKp:IfNACKp
pp_IfProtocolSts:IfProtocolSts
pp_IfBCM_ControlEncoding:IfBCM_ControlEncoding
pp_IfBCM_ENQ_VariableDataPt1:pp_IfBCM_ENQ_VariableDataPt1
pp_IfBCM_PRA_VariableDataPt1:pp_IfBCM_PRA_VariableDataPt1
pp_Ifset_timer_T_wait_BCM:Ifset_timer_T_wait_BCM
pp_Ifcancel_timer_T_wait_BCM:Ifcancel_timer_T_wait_BCM
pp_Ifset_timer_T_wait_rec:Ifset_timer_T_wait_rec
pp_Ifcancel_timer_T_wait_rec:Ifcancel_timer_T_wait_rec
op_set
rp_in_SVXMode:in_SVXMode

Figure 6.5-2 Example of Applications SWCs Used

6.6 Defining the Behavior for the Application Software Components

As mentioned before, we used the approach of creating the model skeleton in VSA, export it in AUTOSAR xml, and then import it in Simulink to define its behavior using Model-Based Approach.

Once the system architecture is created, the next step is to create the behavior of the two main Software Components.

Because the architecture created in VSA is only a skeleton and the Application Software Components do not contain the behavior.

The way to define this behavior is to write it manually or to use external tools to auto generate the code.

In this case I used Simulink to create the behavior of the Application Software Components using the Model Based Software Design technique and do an Early Validation before auto generate the code.

6.6.1 Using Simulink for a Model Based Approach with Auto-Generated Code

Using Simulink makes it possible to define the behavior of the Application Software Component, without writing the code manually, using the auto code generation tool.

In Simulink the support for AUTOSAR Software Component is offered by the Embedded Coder Support Package for AUTOSAR Standard.

It is possible to use Stateflow and then exploit the concept of states and transitions of its FSM, in order to:

- Facilitate the work to the designer.
- Provide easier maintainability
- Avoid possible errors through the validation and testing phase, without having to write any line of code.

This is more useful in complex and articulated systems, allowing even faster editing of the model.

As mentioned before, after the data is exported from the architecture created in VSI, in this case arxml files can be imported in Matlab, and according to Matlab/Simulink documentations, the main steps are:

```

>> imp = arxml . i m p o r t e r ( 'FILENAME. arxml ' )

imp =
          The  f i l e   " SwcAppSw . arxml"  c o n t a i n s :
          1  A p p l i c a t i o n  Software  Component  Type :
                              ' /mySWCs/APPLICATION_SWC_NAME'

                    0  Sensor  Actuator  Software  Component  Type .
                    0  CalPrm  Component  Type .
                    0  C l i e n t  S e r v e r  I n t e r f a c e .

imp . s e t D e p e n d e n c i e s ( 'DEPENDENCIES_FILENAME. arxml ' )

>> imp . createComponentAsModel ( ' /mySWCs/APPLICATION_SWC_NAME' )
```

Figure 6.6-1 E.g. importing code in Matlab

The last command will generate the model skeleton in which will be implemented the behavior.

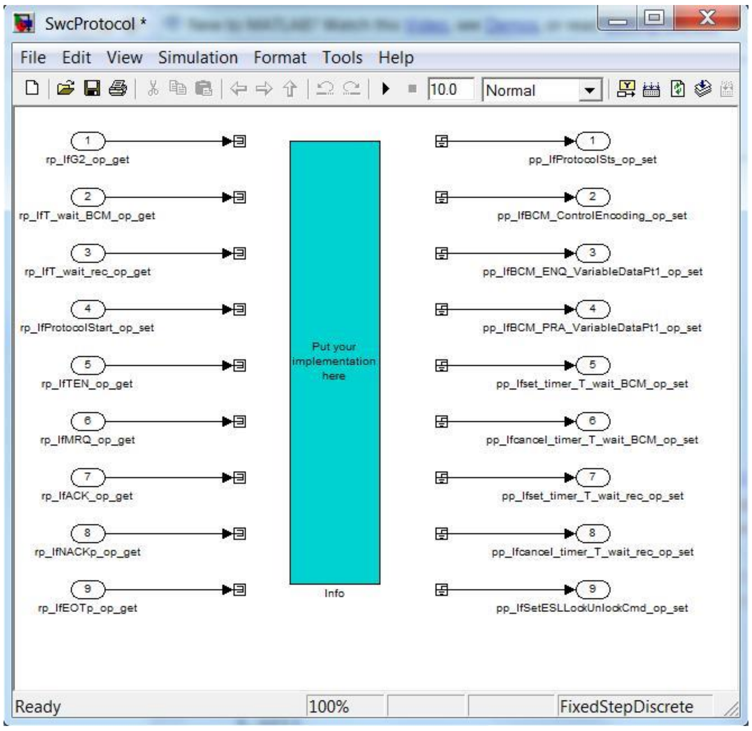

SwcProtocol *
File Edit View Simulation Format Tools Help
10.0
Normal
1
rp_IfG2_op_get
2
rp_IfT_wait_BCM_op_get
3
rp_IfT_wait_rec_op_get
4
rp_IfProtocolStart_op_set
5
rp_IfTEN_op_get
6
rp_IfMRQ_op_get
7
rp_IfACK_op_get
8
rp_IfNACKp_op_get
9
rp_IfEOTp_op_get
Put your implementation here
Info
1
pp_IfProtocolSts_op_set
2
pp_IfBCM_ControlEncoding_op_set
3
pp_IfBCM_ENQ_VariableDataPt1_op_set
4
pp_IfBCM_PRA_VariableDataPt1_op_set
5
pp_Ifset_timer_T_wait_BCM_op_set
6
pp_Ifcancel_timer_T_wait_BCM_op_set
7
pp_Ifset_timer_T_wait_rec_op_set
8
pp_Ifcancel_timer_T_wait_rec_op_set
9
pp_IfSetESLLockUnlockCmd_op_set
Ready
100%
FixedStepDiscrete

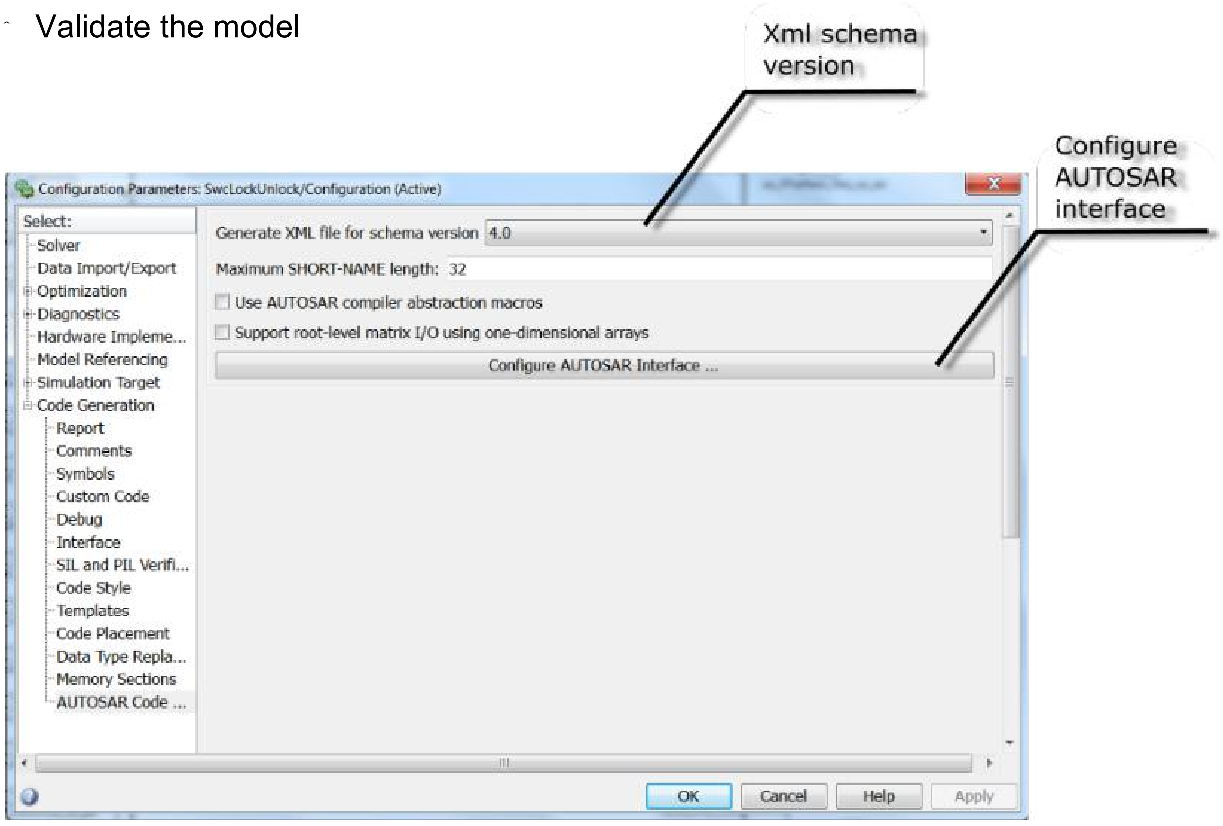
Figure 6.6-2 Example of Imported Architecture

In Figure 6.6.2 it's shown that it is possible to exchange the **“Put you implementation here”** block with the desired behavior, and then connect the inputs/outputs.

After creation of the behavior and all the inputs/outputs ports connections, it is possible to proceed with the automatic code generation, that will generate some files, in which the most important files are the C/C++ and arxml files.

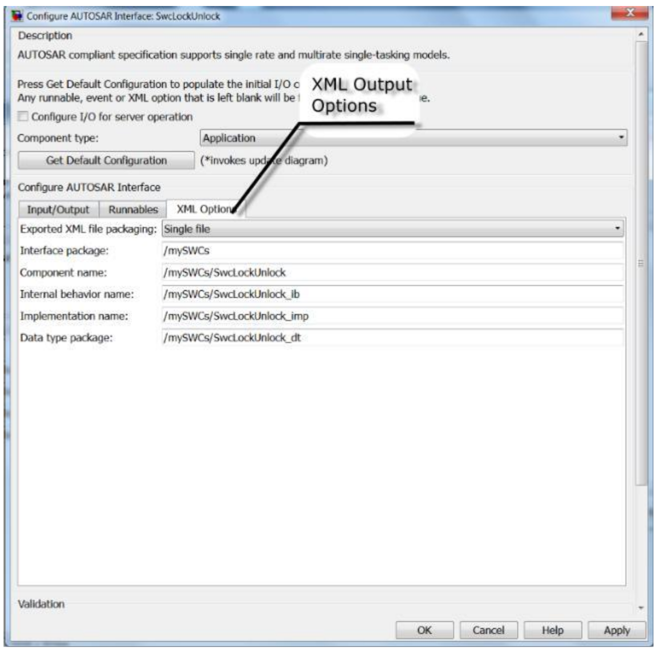
All of them must be imported in VSI, and for doing that it is necessary to set some parameters, showed in the next figures, and in detail:

- Select the schema version for the xml ls that will be generated
- Choose the exporting options (e.g. exporting on single file or on multiple files)
- Validate the model



Generate XML file for schema version: 4.0
Maximum SHORT-NAME length: 32
☐ Use AUTOSAR compiler abstraction macros
☐ Support root-level matrix I/O using one-dimensional arrays

Figure 6.6-3 Select the XML file for schema version


XML Output Options
Configure AUTOSAR Interface: SwcLockUnlock
Description
Get Default Configuration
Configure AUTOSAR Interface
Input/Output
Runnables
XML Options
Validation
OK
Cancel
Help
Apply

AUTOSAR compliant specification supports single rate and multirate single-tasking models.

Press Get Default Configuration to populate the initial I/O c
Any runnable, event or XML option that is left blank will be e.

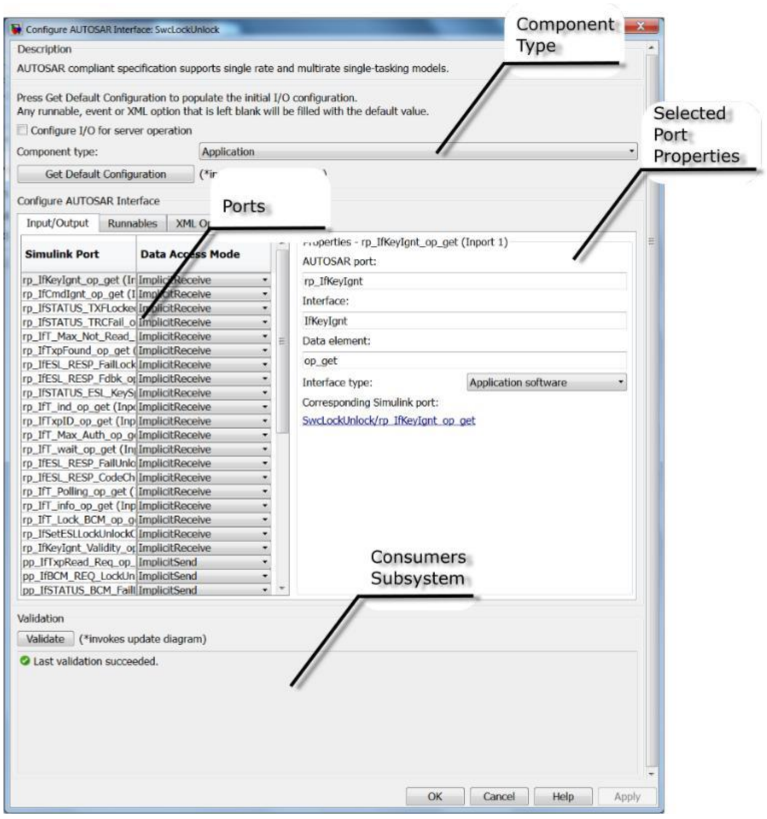
☐ Configure I/O for server operation

Component type: Application

(*invokes update diagram)

Exported XML file packaging: Single file
Interface package: /mySWCs
Component name: /mySWCs/SwcLockUnlock
Internal behavior name: /mySWCs/SwcLockUnlock_ib
Implementation name: /mySWCs/SwcLockUnlock_imp
Data type package: /mySWCs/SwcLockUnlock_dt

Figure 6.6-4 Exporting options



Configure AUTOSAR Interface: SwcLockUnlock

Description

AUTOSAR compliant specification supports single rate and multirate single-tasking models.

Press Get Default Configuration to populate the initial I/O configuration.
Any runnable, event or XML option that is left blank will be filled with the default value.

☐ Configure I/O for server operation

Component type: Application

Get Default Configuration (*ir

Configure AUTOSAR Interface

Input/Output | Runnables | XML O

Simulink Port	Data Access Mode
rp_IfKeyIgnt_op_get (Ir	ImplicitReceive
rp_IfCmdIgnt_op_get (I	ImplicitReceive
rp_IfSTATUS_TXFLocke	ImplicitReceive
rp_IfSTATUS_TRCFail_o	ImplicitReceive
rp_IfT_Max_Not_Read_	ImplicitReceive
rp_IfTxpFound_op_get (	ImplicitReceive
rp_IfESL_RESP_FailLock	ImplicitReceive
rp_IfESL_RESP_Fdbk_op	ImplicitReceive
rp_IfSTATUS_ESL_KeyS	ImplicitReceive
rp_IfT_ind_op_get (Inpu	ImplicitReceive
rp_IfTxpID_op_get (Inp	ImplicitReceive
rp_IfT_Max_Auth_op_g	ImplicitReceive
rp_IfT_wait_op_get (In	ImplicitReceive
rp_IfESL_RESP_FailUnlo	ImplicitReceive
rp_IfESL_RESP_CodeCh	ImplicitReceive
rp_IfT_Polling_op_get (	ImplicitReceive
rp_IfT_info_op_get (Inp	ImplicitReceive
rp_IfT_Lock_BCM_op_g	ImplicitReceive
rp_IfSetESLLockUnlockC	ImplicitReceive
rp_IfKeyIgnt_Validity_op	ImplicitReceive
pp_IfTxpRead_Req_op_	ImplicitSend
pp_IfBCM_REQ_LockUn	ImplicitSend
pp_IfSTATUS_BCM_Faill	ImplicitSend

Properties - rp_IfKeyIgnt_op_get (Inport 1)

AUTOSAR port: rp_IfKeyIgnt

Interface: IfKeyIgnt

Data element: op_get

Interface type: Application software

Corresponding Simulink port: SwcLockUnlock/rp_IfKeyIgnt_op_get

Validation

Validate (*invokes update diagram)

Last validation succeeded.

Figure 6.6-5 Validation Succeeded in Simulink model

In the Figure 6.6.5 you can see that Simulink provides us with a useful tool, it is able to make an initial validation, useful to see if there are any errors in the model, or if the blocks we are used are not compatible with the AUTOSAR standard, and possibly correct them before re-importing in VSI.

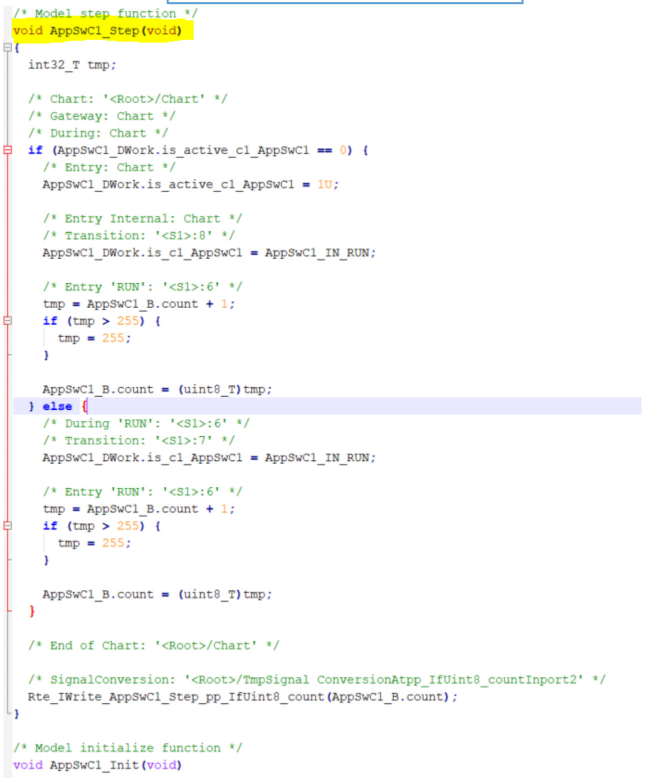
If the validation passed it is possible to generate the code for the behavior.

The Simulink generated les are:

- C/C++, h
- arxml
- other files

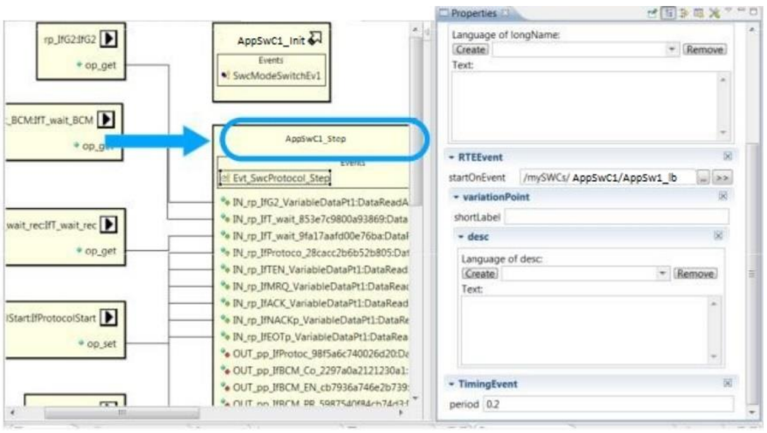
C/C++ les contain the functions with the behavior, and each function's name will be mapped with a Runnable entity in AUTOSAR, for example:

AppSwc1.c



```
/* Model step function */
void AppSwC1_Step(void)
{
  int32_T tmp;

  /* Chart: '<Root>/Chart' */
  /* Gateway: Chart */
  /* During: Chart */
  if (AppSwC1_DWork.is_active_c1_AppSwC1 == 0) {
    /* Entry: Chart */
    AppSwC1_DWork.is_active_c1_AppSwC1 = 1U;

    /* Entry Internal: Chart */
    /* Transition: '<S1>:8' */
    AppSwC1_DWork.is_c1_AppSwC1 = AppSwC1_IN_RUN;

    /* Entry 'RUN': '<S1>:6' */
    tmp = AppSwC1_B.count + 1;
    if (tmp > 255) {
      tmp = 255;
    }

    AppSwC1_B.count = (uint8_T)tmp;
  } else {
    /* During 'RUN': '<S1>:6' */
    /* Transition: '<S1>:7' */
    AppSwC1_DWork.is_c1_AppSwC1 = AppSwC1_IN_RUN;

    /* Entry 'RUN': '<S1>:6' */
    tmp = AppSwC1_B.count + 1;
    if (tmp > 255) {
      tmp = 255;
    }

    AppSwC1_B.count = (uint8_T)tmp;
  }

  /* End of Chart: '<Root>/Chart' */

  /* SignalConversion: '<Root>/TmpSignal ConversionAtpp_IfUint8_countInport2' */
  Rte_IWrite_AppSwC1_Step_pp_IfUint8_count(AppSwC1_B.count);
}

/* Model initialize function */
void AppSwC1_Init(void)
```

Graphical Runnable Editor

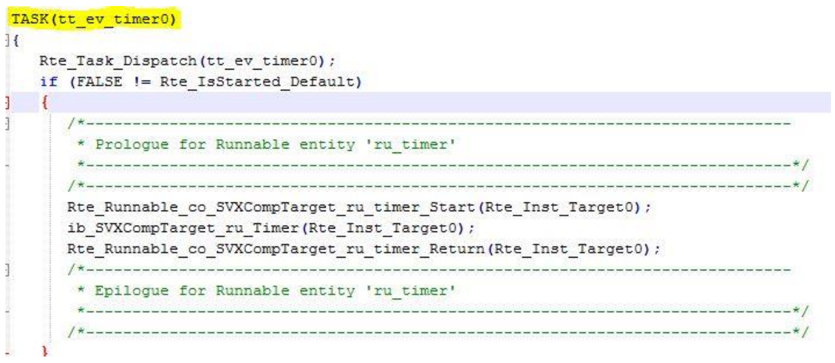

rp_IfG2:IfG2
op_get
_BCM:IfT_wait_BCM
op_get
wait_rec:IfT_wait_rec
op_get
IStart:IfProtocolStart
op_set
AppSwC1_Init
Events
SwcModeSwitchEv1
AppSwC1_Step
Events
Evt_SwcProtocol_Step
IN_rp_IfG2_VariableDataPt1:DataReadA
IN_rp_IfT_wait_853e7c9800a93869:Data
IN_rp_IfT_wait_9fa17aafd00e76ba:Data
IN_rp_IfProtoco_28cacc2b6b52b805:Da
IN_rp_IfTEN_VariableDataPt1:DataRead
IN_rp_IfMRQ_VariableDataPt1:DataRea
IN_rp_IfACK_VariableDataPt1:DataRead
IN_rp_IfNACKp_VariableDataPt1:DataRe
IN_rp_IfEOTp_VariableDataPt1:DataRea
OUT_pp_IfProtoc_98f5a6c740026d20:D
OUT_pp_IfBCM_Co_2297a0a2121230a1:
OUT_pp_IfBCM_EN_cb7936a746e2b739:
Properties
Create
Remove
RTEEvent
variationPoint
desc
Create
Remove
TimingEvent

Language of longName:
Text:
startOnEvent /mySWCs/ AppSwC1/AppSw1_lb
shortLabel
Language of desc:
Text:
period 0.2

Figure 6.6-6 Relation between generated code and Runnable Entities

But because the OS view do not works directly with the Runnables, they have to be mapped into OS tasks 6.6.8, that will be contained in the Rte_Core generated file.

RteCore0.c



```
TASK(tt_ev_timer0)
{
    Rte_Task_Dispatch(tt_ev_timer0);
    if (FALSE != Rte_IsStarted_Default)
    {
        /*----------------------------------------------------------------------------
         * Prologue for Runnable entity 'ru_timer'
         *---------------------------------------------------------------------------*/
        /*---------------------------------------------------------------------------*/
        Rte_Runnable_co_SVXCompTarget_ru_timer_Start(Rte_Inst_Target0);
        ib_SVXCompTarget_ru_Timer(Rte_Inst_Target0);
        Rte_Runnable_co_SVXCompTarget_ru_timer_Return(Rte_Inst_Target0);
        /*----------------------------------------------------------------------------
         * Epilogue for Runnable entity 'ru_timer'
         *---------------------------------------------------------------------------*/
        /*---------------------------------------------------------------------------*/
    }
```

Figure 6.6-7 Runnables mapped into OS tasks

6.6.2 Hand written Code

It is possible to generate the code for the Application Software Components behavior using a manual approach, in fact some behavior in this project are realized using some hand written code (C in this case), only for curiosity than necessity, for exploring different approaches offered by the tool.

```

#include <stdio.h>
#include "Rte_convGenInteger.h"


FUNC( void , RTE_APPL_CODE)
        convGenInteger_run ( P2CONST( struct Rte_CDS_convGenInteger ,
                                                          AUTOMATIC,
                                                          RTE_APPL_CONST) instance ){

        dt_SVX_Integer_c input;
        //get the input
        Rte_Read_rp_IfConv_op_set (instance, &input);

        //set the output
        Rte_Call_rp_in_ISVXCompSigGenDouble_op_set (instance, input);
}
```

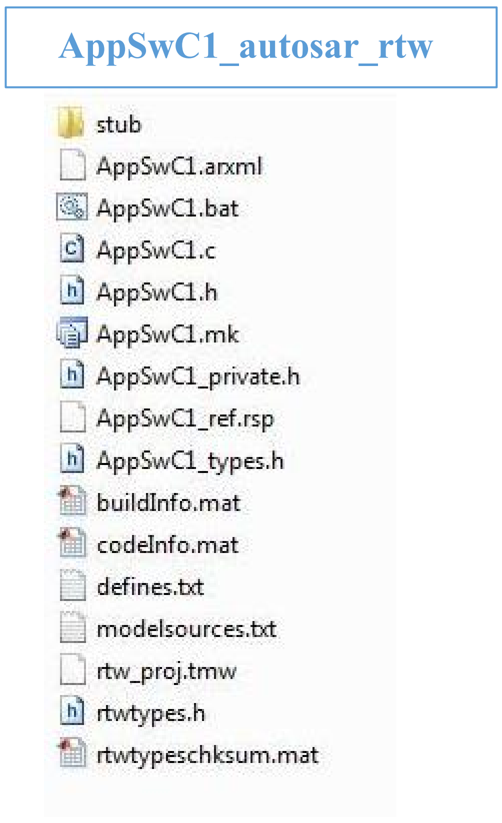
Figure 6.6-8 Example of hand written code in C

6.6.2 Importing Back the Auto-Generated Code in VSI

After the behavior creation phase in Simulink, is possible to re-import all the auto generated files back in VSI.

In this way we will have not only the skeleton model, but even the behavior for the components.

For doing that it is necessary to import all the generated files from Simulink.



AppSwC1_autosar_rtw

stub
AppSwC1.arxml
AppSwC1.bat
AppSwC1.c
AppSwC1.h
AppSwC1.mk
AppSwC1_private.h
AppSwC1_ref.rsp
AppSwC1_types.h
buildInfo.mat
codeInfo.mat
defines.txt
modelsources.txt
rtw_proj.tmw
rtwtypes.h
rtwtypeschksum.mat

Figure 6.6-9 Example of Simulink generated files

After importing back the auto generated code, the new components with new behavior will be available.

With the used version of Matlab (2012a), all the Runnable entities could have only two possible entity event:

- Timing Event
- Data Received Event

Newest version of Matlab introduced other types of entity events, but in this case and for this project only this two types of events are used, considering the constraint on the Matlab version.

The Timing Event repeat the operation based on the configured period, instead the Data Received Event starts the Runnable when an input presents in the associated port.

In VSx is possible to see a graphical representation of the Runnable Entities using the Graphical Runnable Editor in which we can create or modify easily the Runnable Entities.

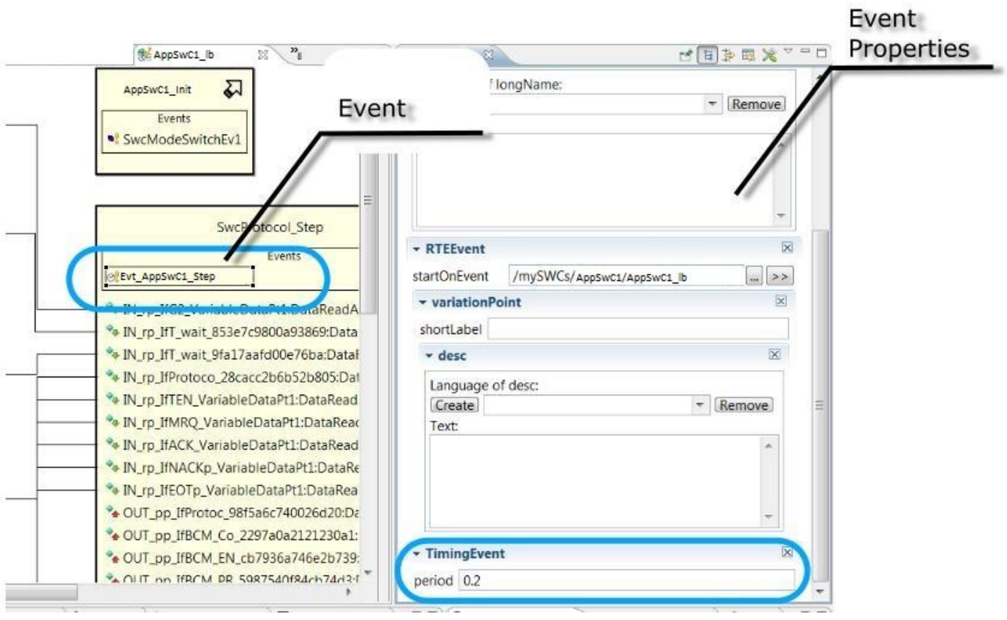
The tools are divided in below categories:

- Communication Among Runnables
- Sender/Receiver Communication
- Client/Server Communication

- External Trigger Event Communication
- Parameter Access
- Mode Communication

In this categories, we can find the tools for creating Runnable Entities and Events like Timing Event, Background Event, DataReceived Event, SynchronousServerCallPoint and even the tools for create connections between the Runnable Entities and Ports.

Each Event can be configured properly to meet the specications, gure 6.6.11, for example for the Timing Event, it is possible to set timing period instead for the DataReceived Event to set the ContextRPort and the targetDataElement.


Event
Event Properties
AppSwC1_Ib
AppSwC1_Init
Events
SwcModeSwitchEv1
SwcProtocol_Step
Events
Evt_AppSwC1_Step
IN_rp_IfT_wait_853e7c9800a93869:Data
IN_rp_IfT_wait_9fa17aafd00e76ba:Data
IN_rp_IfProtoco_28cacc2b6b52b805:Da
IN_rp_IfTEN_VariableDataPt1:DataRead
IN_rp_IfMRQ_VariableDataPt1:DataRea
IN_rp_IfACK_VariableDataPt1:DataRead
IN_rp_IfNACKp_VariableDataPt1:DataRe
IN_rp_IfEOTp_VariableDataPt1:DataRea
OUT_pp_IfProtoc_98f5a6c740026d20:Da
OUT_pp_IfBCM_Co_2297a0a2121230a1:
OUT_pp_IfBCM_EN_cb7936a746e2b739:
longName:
Remove
RTEEvent
variationPoint
desc
Language of desc:
Create
Remove
Text:
TimingEvent

startOnEvent /mySWCs/AppSwC1/AppSwC1_Ib

shortLabel

period 0.2

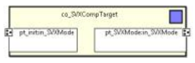
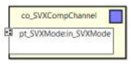
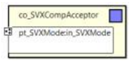
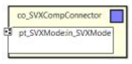
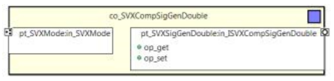
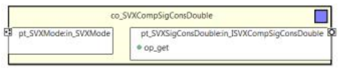
Figure 6.6-10 Example of Timing Event Properties

6.7 Use of the conneXion (SVX) Components in the AUTOSAR Model

For using conneXion it is necessary to introduce some particular Software Components that will manage the communication between the VSx model and the outside world, in this case the Plant created using Simulink.

The main conneXion Software Components are:

Table 6-1 Main conneXion Software Components

	Name	Description
 co_SVXCompTarget pt_initin_SVXMode pt_SVXMode:in_SVXMode	Target	Manage conneXion resources and is responsible of synchronization in distributed applications
	Channel	Used for define the channel name
	Acceptor	Used for specify the role for a node, this in particular passively waits
	Connector	Used for specify the role for a node, this in particular actively connects
 co_SVXCompSigGenDouble pt_SVXMode:in_SVXMode pt_SVXSigGenDouble:in_ISVXCompSigGenDouble op_get op_set	Generator	For generating outputs for the rest of the system
 co_SVXCompSigConsDouble pt_SVXMode:in_SVXMode pt_SVXSigConsDouble:in_ISVXCompSigConsDouble op_get	Consumer	For retrieving inputs from the rest of the system

Each component can be configured properly and in some cases like for the generators or for the consumers, there are several versions of them based on different data types, for having coverage on the major data types used.

6.7.1 Using conneXion Components with Sender/Receiver Interfaces

With the version of VSx used, the only conneXion components available are the ones showed previously in the table 6.1, and all that components for connecting with the application Software Components use the client-server interfaces, and there's no other connection ports available.

There's no way to use components with different interfaces other than client/server. So for instance to use sender-receiver interfaces we need to introduce another Software
Component which behave like a bridge between the two SWCs.

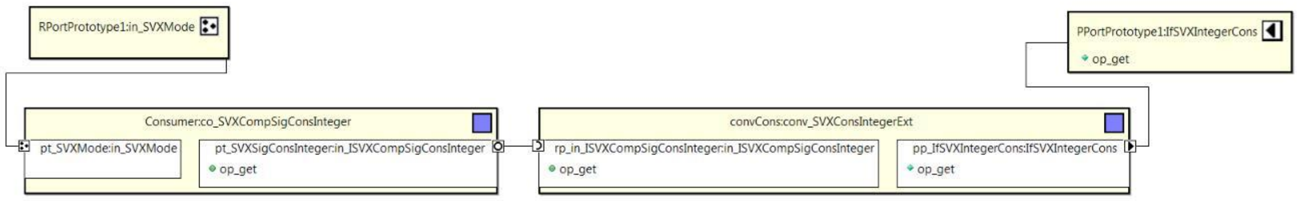

RPortPrototype1:in_SVXMode
PPortPrototype1:IfSVXIntegerCons
op_get
Consumer:co_SVXCompSigConsInteger
pt_SVXMode:in_SVXMode
pt_SVXSigConsInteger:in_ISVXCompSigConsInteger
op_get
convCons:conv_SVXConsIntegerExt
rp_in_ISVXCompSigConsInteger:in_ISVXCompSigConsInteger
op_get
pp_IfSVXIntegerCons:IfSVXIntegerCons
op_get

Figure 6.7-1 Consumer SWC Bridge

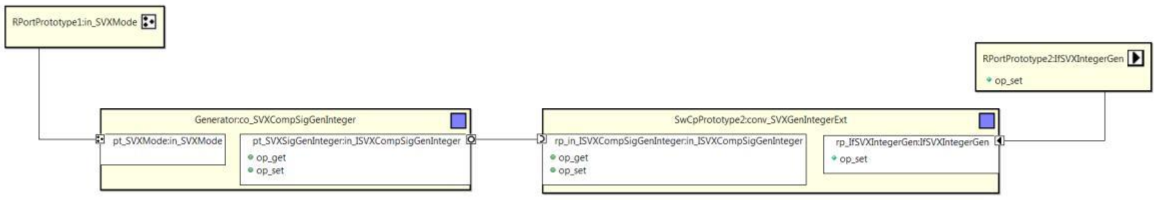

RPortPrototype1:in_SVXMode
Generator:co_SVXCompSigGenInteger
pt_SVXMode:in_SVXMode
pt_SVXSigGenInteger:in_ISVXCompSigGenInteger
op_get
op_set
SwCpPrototype2:conv_SVXGenIntegerExt
rp_in_ISVXCompSigGenInteger:in_ISVXCompSigGenInteger
op_get
op_set
rp_IfSVXIntegerGen:IfSVXIntegerGen
op_set
RPortPrototype2:IfSVXIntegerGen
op_set

Figure 6.7-2 Generator SWC Bridge

The introduced Software Component has a simple behavior, practically takes the data from one port and sends it to the output door for permitting the communication between the two SWCs with ports that using different interfaces.

This surely introduce a delay in the system, particularly in the Consumer, because the system have to request the data from the client-server interfaces, and for doing that a timer event is used.

The timer event have a period t, so the delay introduced is at least equal to t.

The behavior of this Software Component is different in the Consumer and in the Generator, but they are quite similar.

The Consumer is composed by a Runnable entity with a timing event, and in each cycle, the value in the input port is checked, using a client RPort and requesting the data from a Server PPort, so in this case the delay is present.

On contrary in the Generator the Runnable entity does not have the timing event, but the Data Received Event. because the data comes from a sender-receiver port and then it will be sent to the output port that uses the client-server interface.

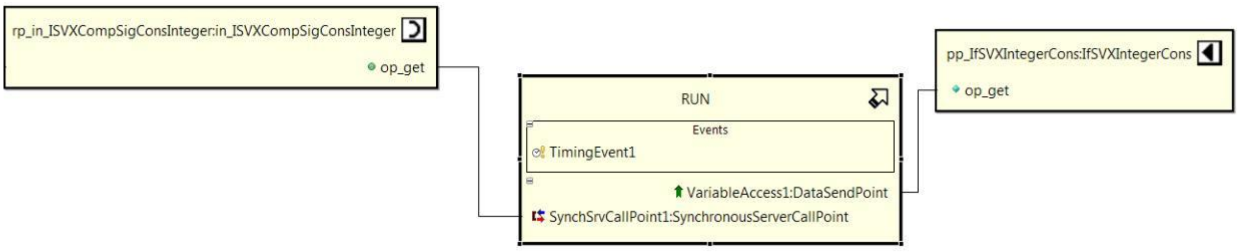

rp_in_ISVXCompSigConsInteger:in_ISVXCompSigConsInteger
op_get
RUN
Events
TimingEvent1
VariableAccess1:DataSendPoint
SynchSrvCallPoint1:SynchronousServerCallPoint
pp_IfSVXIntegerCons:IfSVXIntegerCons
op_get

Figure 6.7-3 Consumer Runnable

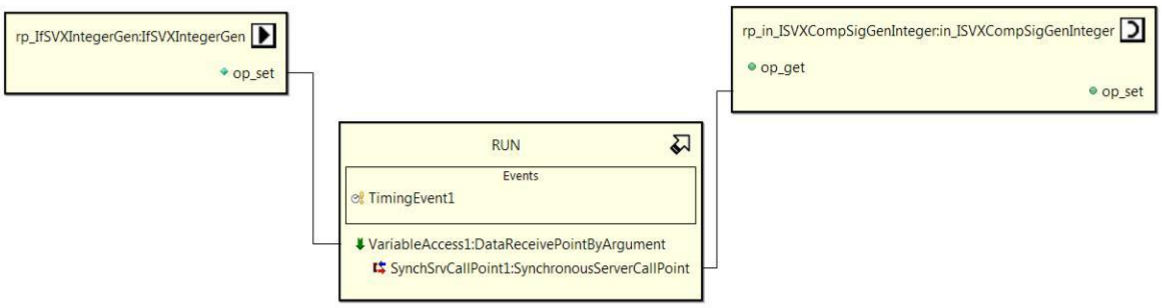

rp_IfSVXIntegerGen:IfSVXIntegerGen
op_set
RUN
Events
TimingEvent1
VariableAccess1:DataReceivePointByArgument
SynchSrvCallPoint1:SynchronousServerCallPoint
rp_in_ISVXCompSigGenInteger:in_ISVXCompSigGenInteger
op_get
op_set

Figure 6.7-4 Generator Runnable

6.7.2 conneXion Components Configuration

After placing and connecting all this components that are necessary for the communication between the Plant in Simulink and the VSI model, we need to configure them.

So for doing that it is necessary to use the Module Configuration Parameters, present in VSI.

Therefore it is necessary to create a new ECUCModuleConfigurationValues to which will be given as definitions the file /pkg_svx_autosar/mdef_SVX, that contains all the definitions of the components used by conneXion.

After adding the definitions it is possible to insert the EcucContainerValues that we need and then configure them using the Module Configuration Parameter Editor.

In this model the following EcucContainerValues are created:

- 1 * svxTarget
- 1 * svxChannel
- 1 * svxAcceptor
- n * svxGenerators
- m * svxConsumers

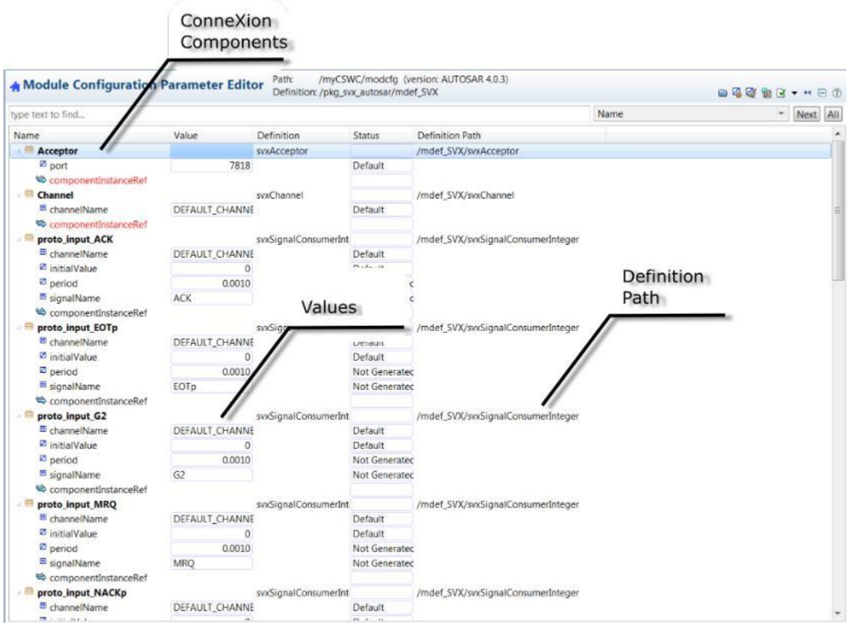
The Target is configured to indicate that the model is an acceptor instead of a sequencer, the Acceptor contains the indication about the port used and the Channel contains the name of the chosen channel.

Also for each inputs and outputs we need to define:

- channelName: channel name
- initialValue: starting value of the signal (default 0)
- period: signal rate
- signalName: unique signal name

In addition to each of these parameters we must also give the reference target, i.e. to whom Component relate, and if necessary to indicate the path, starting from the root component.

Below a screen about the configuration of the used parameters:



Module Configuration Parameter Editor

Path: /myCSWC/modcfg (version: AUTOSAR 4.0.3)
Definition: /pkg_svx_autosar/mdef_SVX

type text to find...

Name

Name	Value	Definition	Status	Definition Path
Acceptor		svxAcceptor		/mdef_SVX/svxAcceptor
port	7818		Default	
componentInstanceRef				
Channel		svxChannel		/mdef_SVX/svxChannel
channelName	DEFAULT_CHANNE		Default	
componentInstanceRef				
proto_input_ACK		svxSignalConsumerInt		/mdef_SVX/svxSignalConsumerInteger
channelName	DEFAULT_CHANNE		Default	
initialValue	0		Default	
period	0.0010		[illegible]	
signalName	ACK		[illegible]	
componentInstanceRef				
proto_input_EOTp		svxSig		/mdef_SVX/svxSignalConsumerInteger
channelName	DEFAULT_CHANNE		Default	
initialValue	0		Default	
period	0.0010		Not Generatec	
signalName	EOTp		Not Generatec	
componentInstanceRef				
proto_input_G2		svxSignalConsumerInt		/mdef_SVX/svxSignalConsumerInteger
channelName	DEFAULT_CHANNE		Default	
initialValue	0		Default	
period	0.0010		Not Generatec	
signalName	G2		Not Generatec	
componentInstanceRef				
proto_input_MRQ		svxSignalConsumerInt		/mdef_SVX/svxSignalConsumerInteger
channelName	DEFAULT_CHANNE		Default	
initialValue	0		Default	
period	0.0010		Not Generatec	
signalName	MRQ		Not Generatec	
componentInstanceRef				
proto_input_NACKp		svxSignalConsumerInt		/mdef_SVX/svxSignalConsumerInteger
channelName	DEFAULT_CHANNE		Default	

Figure 6.7-5 Module Configuration Parameter Editor

7 Plant Creation using Simulink

7.1 Plant Overview

The first image 7.0.1 shown below, is the upper level of the system in which we can see two subsystems, one contains the two models: the conneXion model and the starting model, instead the other subsystem contains the timers.

The other elements are necessary for taking the input data from the workspace do a conversion and send them to the system at a defined rate.

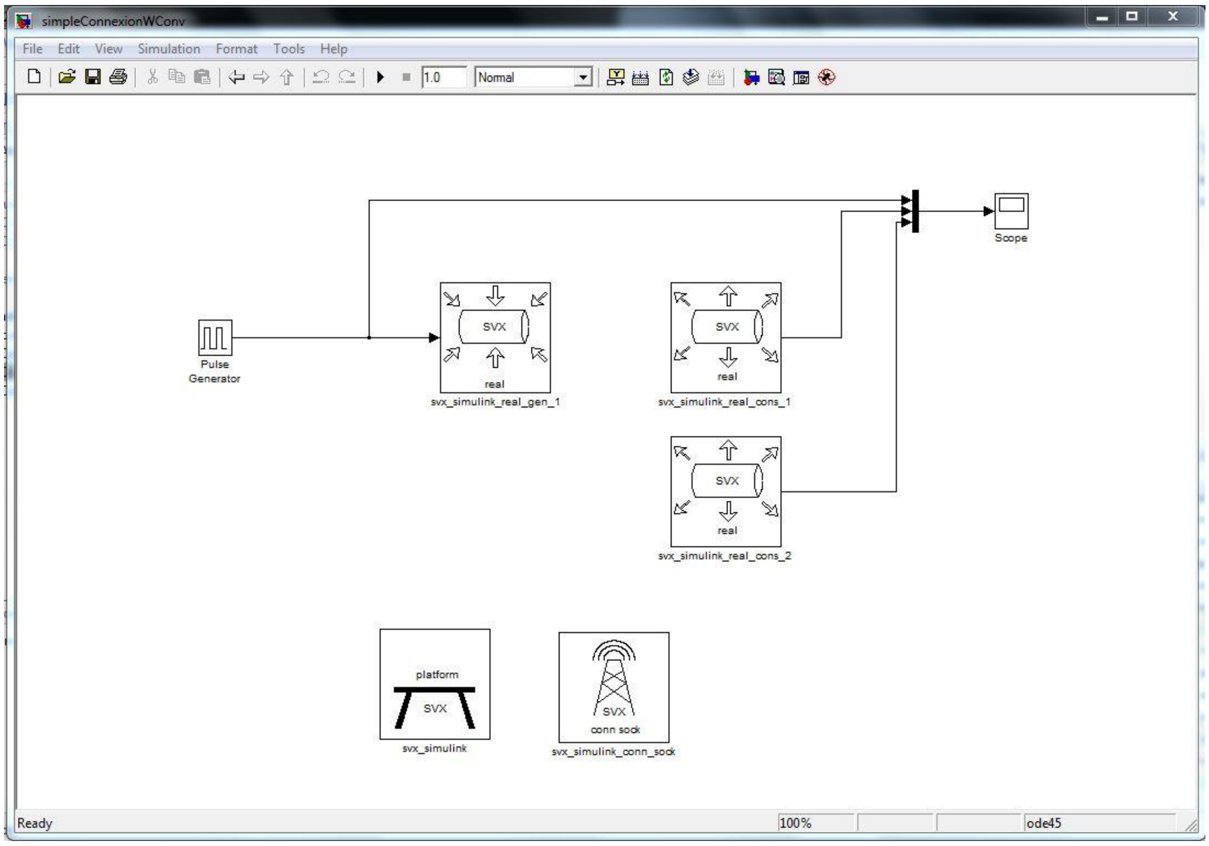


Figure 7.1-1 Plant Overview

- The conneXion blocks to establish the connection with the VSx model
- The Generators subsystem that contains all the conneXion Generator blocks for generating inputs for VSx model
- The Consumers subsystem that contains all the conneXion Consumer blocks for receiving outputs from VSx model
- Scope for comparing tests data from the two systems
- The subsystem that contains the starting model

7.2 Using conneXion Blocks in Simulink

To connect the two models, it is necessary to use the blocks in the SVX library, which as previously seen for SVX for Simulink The main blocks are:

Plant Creation in Simulink

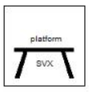
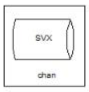
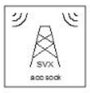
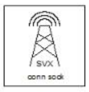
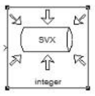
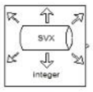
	Name	Description
	Target	Used for specify if the model is a sequencer or an acceptor
	Channel	Used for define the channel name
	Acceptor	Used for specify the role for a node, this in particular passively waits
	Connector	Used for specify the role for a node, this in particular actively connects
	Generator	For generating outputs for the rest of the system
	Consumer	For retrieving inputs from the rest of the system

Table 7-1 Main conneXion Blocks for Simulink

7.3 Configuring the Plant

Obviously even these blocks must be configured, like the others in the VSI model, to permit the communication with the model.

The first thing to do is to decide the role of the model, if is a sequencer or an acceptor, checking the box in the settings of the Target block.

In this case this model is a Plant and it provides the inputs, so it is a sequencer.

After the role of the model is chosen for the connection, it is necessary to add the right block, whether it is Acceptor or Connector and configure these blocks, defining the port number used.

Next it is possible to add the Channel block into the model, to indicate the name of the channel to be used, but this block it can be omitted, because in this case is possible to indicate the name of the channel to be used also in the blocks of Consumer and Generator.

At this point the connection settings are configured and the two models can communicate, it is only necessary to insert the blocks for sending or receiving the data.

The blocks used for this are the Consumers and the Generators, choosing the right data type like the one used in the VSI model.

Like the counterpart in the VSI, there are different types for using in Consumer and Generator:

- Boolean
- Integer
    - int8
    - int16
    - int32
- Real
    - single
    - double
- Unsigned
    - uint8
    - uint16
    - uint32

In every block, it is possible to set the signal precision, the channel name, the signal name, minimum and maximum consumption period and the consumption or generation latency.

Obviously you have to set these values according to the values in the VSI model.

8 Validating VSx

8.1 Debug in VSI

VSI provides the facility to debug the created architecture, using breakpoints on architecture elements or directly on the behavior code and to use the step by step debug execution.

In VSI there are two different types of breakpoints: the AUTOSAR breakpoints and the regular C/C++ breakpoints.

Creating an AUTOSAR breakpoint means that the developer can put a breakpoint directly in the AUTOSAR architecture elements.

For example, it is possible to put a breakpoint on a component or on a port, and generally in any other parts of the system.

During a debug session, if there are AUTOSAR breakpoints, the execution will be stopped when the RTE execution reaches the point in which there is involved a component with breakpoint on it, or another component is triggered by the component with the breakpoint.

What VSI do when there are AUTOSAR breakpoints is to map the breakpoint to one or more C/C++ function breakpoints.

The developer can even sets breakpoints directly in the Runnable code, these are the C/C++ breakpoints.

The C/C++ breakpoints supported by VSI are:

- Function breakpoints
- Line breakpoints
- Address breakpoints

In VSI the breakpoints are divided into implicit and explicit.

The implicit breakpoints are the AUTOSAR breakpoints mapped to C/C++ breakpoints, that appear only during a debug session, and they are placed implicitly by VSI.

The explicit breakpoints are the ones that the developer sets, they exists even outside the debug session and can be AUTOSAR or C/C++ breakpoints.

8.2 Integration of The Previous Validation Model

after integrating the VSI Plant and the starting mode, the output data is compared and the output results are the same in each system, therefore the two systems are similar, in which with the same inputs they show the same outputs, they have the same behavior.

8.2.1 Issues:

- The two output signals were not identically the same, but the one created in VSx using AUTOSAR presents a little delay that can be minimized acting on the various component speeds (e.g. Runnable entities event or period in the conneXion ports configuration).This problem exists mainly because we need to use conneXion instead of a real hardware.

- Another problem with conneXion is the impossibility to connect the Software Components with other kind of ports, different from the Client/Server type. This leads to use other software components for acting like a bridge for taking the input and sending the outputs to the right port. Because the conneXion Software Components are divided in Consumers and Generators, there are two main scenarios using these kind of bridges: Taking the input from the Consumer and providing the output to the Generator.

These two different kind of components have different behaviors, in this case one is implemented using a Timing Event, instead the other one using a DataReceived Event.

For the case of the Consumer, the input comes from a component with a server port that must be connected to a client port that have to ask for the input at a certain rate.

The Generator case is different because in this case the data starts from the model and it can be used with DataReceived Event that sends the data to the right output port when the data is available.

This bridge, particularly in Consumer case, introduces a delay in the system, that in the worst case is equal to period of the Timing Event.

In any case, the delay problem is presented only in the simulation phase without real hardware.

The solution is to increment the speed of the ports and modeling the system taking in consideration the delay problem.

8.3 Test cases

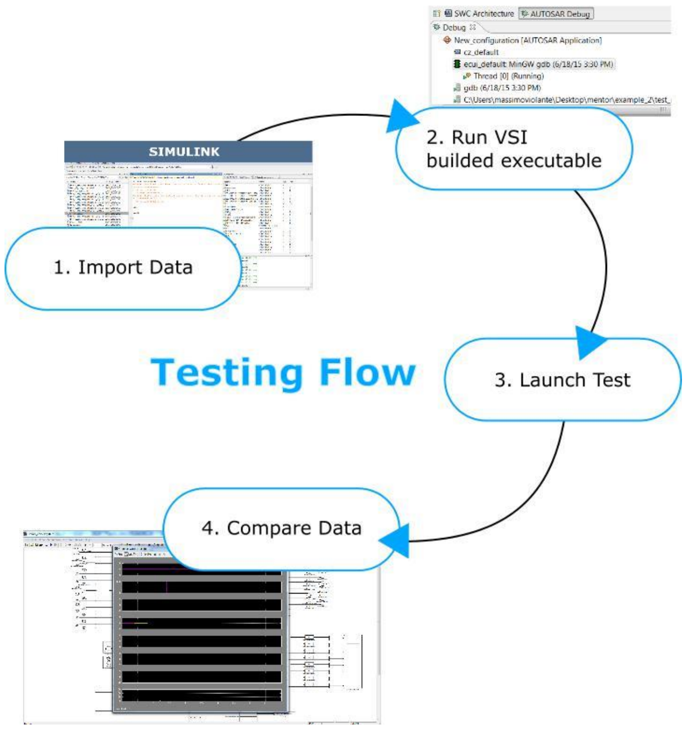

SIMULINK
1. Import Data
2. Run VSI builded executable
3. Launch Test
4. Compare Data
Testing Flow
SWC Architecture
AUTOSAR Debug
Debug
New_configuration [AUTOSAR Application]
c2_default
ecu1_default: MinGW gdb (6/18/15 3:30 PM)
Thread [0] (Running)
gdb (6/18/15 3:30 PM)
C:\Users\massimoviolante\Desktop\mentor\example_2\test_

Figure 8.3-1 Testing Flow

To show the signals better, a delay in the two signals is introduced, like shown in the next figure 8.3.2:

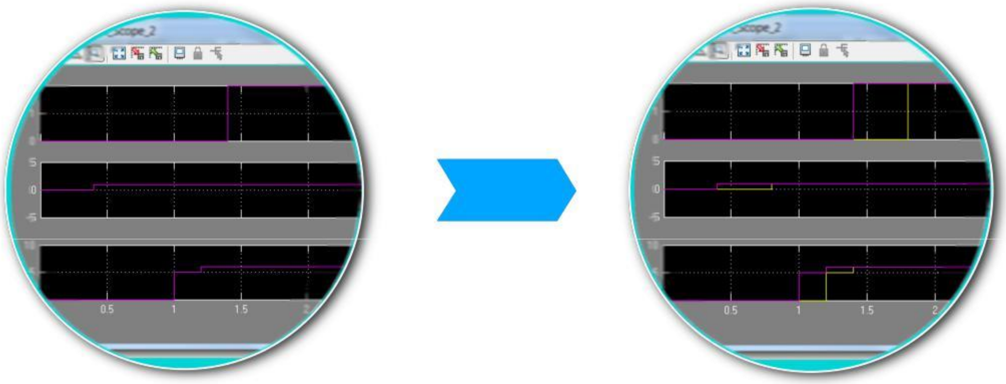


Figure 8.3-2 Delay added for showing the two signals

9 Conclusion

As mentioned before in the introduction, the main goal of this thesis is creation a simple filter using AUTOSAR standard, creating a Plant for sending/receiving inputs/outputs and validation test phase to compare the obtained results with the available ones from the previous implementation.

The obtained results are the same for both models, it means that the two systems are similar and with the same inputs they will generate the same outputs.

The only problem occurred was the delay, introduced by conneXion 8.2, but is solved by increasing the sampling rate for the conneXion signals.

Regarding AUTOSAR, the standardization will result in increasing the quality of the software components and reuse of them.

But even other benefits for manufacturers, suppliers and tool developer [1]:

General benefits:

- Increased re-use of software
- Increased design flexibility
- Clear design rules for integration
- Reduction of costs for software development and service in the long term

Specific benefits for OEMs:

- Functions of competitive nature can be developed separately
- Later sharing of innovations is accessible
- Standardized acceptance tests

Specific benefits for suppliers:

- Reduction of version proliferation
- Development sharing among suppliers
- Increase of efficiency in functional development
- New business models

[1] http://www.autosar.org/about/faq/general/

- Preparation for upcoming increase in software volume

Specific benefits for tool developers:

- Common interfaces with development processes
- Seamless, manageable, task-optimized (time-dependent) tool landscape

In conclusion the Volcano tool is substantially an Eclipse plugin, and have some advantages, for examples the possibility to validate and debug the system and using it concurrently with others models created with different others tools (e.g. Simulink or Labview), but there are also other tools that do the same.

The biggest disadvantage, instead, is that the tool is still in development, so for the moment there are missing functionality and a lot of problems, and in many cases using this tool is more time consuming compared to others.

The obtained results, showing that using this tools is possible, but for the moment is not reliable as it's "unstable" (unhandled exceptions, continuous blocks and subsequent reboots resulting in loss of stored data) , maybe after a stable version will be released, it would be more practical.

10 Definitions, Abbreviations

- BSW: Basic Software
- ECU: Electronic Control Unit
- HW: Hardware
- SWC: Software Component
- RTE: Real Time Environment
- VFB: Virtual Functional Bus
- XML: eXtensible Mark up Language
- VSx: Volcano tool family
- SVX: Old name for conneXion tool

List of Figures

List of Tables

Bibliography

[1] The MathWorks, Matlab Documentation, Import and Export and AUTOSAR Software Component, [http://it.mathworks.com/help/rtw/examples/import-and-export-an-autosar-software-component.html]

[2] Mentor Graphics, Volcano Vehicle System Integrator User Manual

[3] Mentor Graphics, VSx User and Reference Manual

[4] Salvatore Grasso, Model Based Design Of An Immobilizer System, Politecnico di Torino, Torino, Italy

[6] Robert Warschofsky, AUTOSAR Software Architecture, Hasso-Plattner-Institute fur Softwaresystemtechnik

[6] Nico Naumann, AUTOSAR Runtime Environment and Virtual Function Bus, Tech-nical report, Hasso-Plattner-Institute fur Softwaresystemtechnik, 2009

[7] AUTOSAR GbR, [http://www.autosar.org]

[8] Massimo Violante, Introduction to AUTOSAR, Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino, Italy

[9] An introduction to AUTOSAR - [http://retis.sssup.it/sites/default/les/lesson19_autosar.pdf]

[10] Mike Gem¨unde, Evaluation Environment for AUTOSAR - Autocode in Motor Control Units, Embedded Systems Group, Department of Computer Science, Uni-versity of Kaiserslautern

[11] The MathWorks, Inc. 3 Apple Hill Drive, Natick, MA 01760-2098 , RealTime Workshop Embedded Coder 5 User's Guide, (www.mathworks.com, http://www.manualslib.com/manual/392874/Matlab-Real-Time-WorkshopEmbedded-Coder-5.html)

[12] The MathWorks, Inc. 3 Apple Hill Drive, Natick, MA 01760-2098, RealTime Workshop 7 Getting Started Guide, (www.mathworks.com, http://www.manualslib.com/manual/392856/Matlab-Real-Time-Workshop7.html)

[13] Mentor Graphics, VSA Getting Started Guide

[14] Mentor Graphics, conneXion User's Guide

[15] Mentor Graphics, conneXion User's Guide for AUTOSAR

[16] Mentor Graphics, conneXion User's Guide for Simulink