CHALMERS

Automatic code generation of AUTOSAR compliant HMI applications

Master of Science Thesis in Applied Information Technology

SAMIRA AFSHOON
MARYAM GHAVIBAZOU

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, February 2013

Automatic code generation of AUTOSAR compliant HMI applications

Examiner: JAN JONSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, February 2013

Abstract

Today in automotive industry the trend is moving toward use of Human Machine Interfaces for a variety of purposes from designing interfaces for infotainment systems till instrument clusters.

Mecel Populus Suite is a complete tool chain for design, development and deployment of the user interfaces for distributed embedded systems [5]. The current version of this product does not support automatic communication with application under AUTOSAR standard.

The objective of this master thesis is to investigate the possibility of auto-generating AUTOSAR compliant interfaces for the Mecel Populus functional units and implementing an appropriate solution which can serve this purpose and provide an easy procedure for data communication between these two.
The investigation phase consists of proposing a simple process for data mapping and data transmission between Populus and AUTOSAR while the implementation phase consists of providing a user friendly plugin for the Mecel Populus Suite which can offer the expected functionality.

Preface

Mecel is a software and system development company that offers solutions and products specifically for the automotive industry. The thesis proposal was made by Mecel AB as a part of the Second Road Fas 1 project which is initially conducted by Volvo Cars. Second Road Fas 1 project aim is to develop a common simulation environment to evaluate and test active safety functions and HMI-interface before building the cars [12]. Adaptation of HMI platform towards AUTOSAR standard is among the deliverables taken by Mecel and proposed in form of a master thesis.
Populus HMI development suit is one of the major products of this company which is the base of provided solution in this thesis.

We were grateful to have the supervision of Henrik Roslund and Ola Edward regarding Populus and Mathias Fritzon regarding AUTOSAR during this thesis.

Contents

Abbreviations

Artop	ATUOSAR Tool Platform
API	Application Programming Interface
ARXML	AUTOSAR XML
AUTOSAR	Automotive Open System Architecture
BSW	Basic Software
BSWD	Basic Software Description
CAN	Controller Area Network
C/S	Client/Server
DD	Dynamic Data
DE	Data Element
DRP	Data Receive Point
DSP	Data Send Point
ECU	Electronic Control Unit
EMF	Eclipse Modeling Framework
FIL	FU Interface Layer
FR	FlexRay
FU	Functional Unit
FUSWC	Funtional Unit Software Component
HMI	Human Machine Interface
IB	Internal Behavior
IDE	Integrated Development Environment
LIN	Local Interconnect Network
MC	Microcontroller
MCAL	Microcontroller Abstraction Layer
ODI	Open Display Interface
OEM	Original Equipment Manufacturer
PIP	Picea Integration Package
RTE	Run-Time Environment
SCG	Source Code Generator
S/R	Sender/Receiver
VFB	Virtual Functional Bus
SWC	Software Component
SWCD	Software Component Description
XML	Extensible Markup Language

1. Introduction

During the last decade group of major OEMs initiated the development of a standard architecture for reducing the complexity of software development for automotive industry. As a result the attempt led to introducing Automotive Open System Architecture (AUTOSAR).
AUTOSAR is an open standard for software architecture which provides the ability to develop software independent of any specific electronic control unit infrastructure. There are many implementations of this standard available in the industry and Mecel Picea Suite is one of these implementations which have been developed by Mecel AB.

Moreover, Mecel Populus Suite is a complete tool chain product that is also developed by Mecel for easy design, development and deployment of HMIs specifically for the automotive industry. This product offers a SDK which interfaces the HMI and provides the necessary functional base for sending and receiving data to and from HMI.
The current version of Populus is not offering an AUTOSAR compliant SDK for automatic data communication between these two. Therefore, it was requiring further configuration and adaptation of the SDK for making this communication possible.
Therefore there appeared a need for offering an AUTOSAR compliant SDK which can provide the required interfaces for transferring data between the produced HMI by Mecel Populus Suite and an AUTOSAR based application.

Accordingly, the master thesis involves investigating the Mecel Populus workflow and providing an applicable solution for sending and receiving data to and from an AUTOSAR based application which correspondingly writes/reads this data to/from a vehicle communication bus. For this purpose, Mecel Populus Suite is being used as the base in the thesis work but the Mecel Picea is used just as an instance of AUTOSAR implementation and the solutions are addressed in a generic way and totally independent of any specific implementation of AUTOSAR.

1.1 Background

1.1.1 Human Machine Interface (HMI)

A Human Machine Interface (HMI) enables user to interact with the underlying application or system through a graphical interface. Therefore it provides an easier and more understandable way of interaction for user.
In every human machine interface two important component needs to be considered. The first is how to receive input from user and the second is how to present the resultant output to it. Besides, designing a functional, accessible and pleasant to use HMI is considerably challenging.
Mecel Populus Suite provides an easy solution for design and implementation of the desired HMIs without having to write any software. It has specially targeted for automotive industry to deliver high-performance user interfaces with short time-to-market and enable efficient software life cycle management [5].

1.1.2 Automotive Open System Architecture (AUTOSAR)

During the last decade the innovation of Electronic/Electrical (E/E) system led to a growing complexity in this area and also an increasing number of software in the area of functionality. Considering these facts the automotive industry was requiring a more flexible, scalable and reliable solution for taking control of the ECUs. These requirements motivated major automotive companies to work together with suppliers and the software industry to develop an open standard for automotive software architecture [1]. This open standard was called AUTOSAR.

The fundamental design of AUTOSAR is the separation between the application and infrastructure. AUTOSAR introduced a standardized layer between application software and the electronic control unit hardware. Therefore software is largely independent of the chosen microcontroller and the OEM [2]. This will result in a simplified development process and a higher flexibility and easy reuse of the application software [2].

AUTOSAR consortium was founded in 2003 by BMW, DaimlerChrysler, Bosch, Continental, Volkswagen and Siemens VDO. Later Ford, General Motors, Toyota and PSA (Peugeot Citroen) joined the consortium as core members [9]. As a result two prime versions of AUTOSAR have been released since then. AUTOSAR 3.x and AUTOSAR 4.x were released at the end of 2007 and 2009 respectively [2]. The recently sophisticated specification of AUTOSAR was published at January of 2012 which has introduced new features in the area of Network, Safety and processing components. This is the 4.0.3 release and the concentration of this project is also on this release.

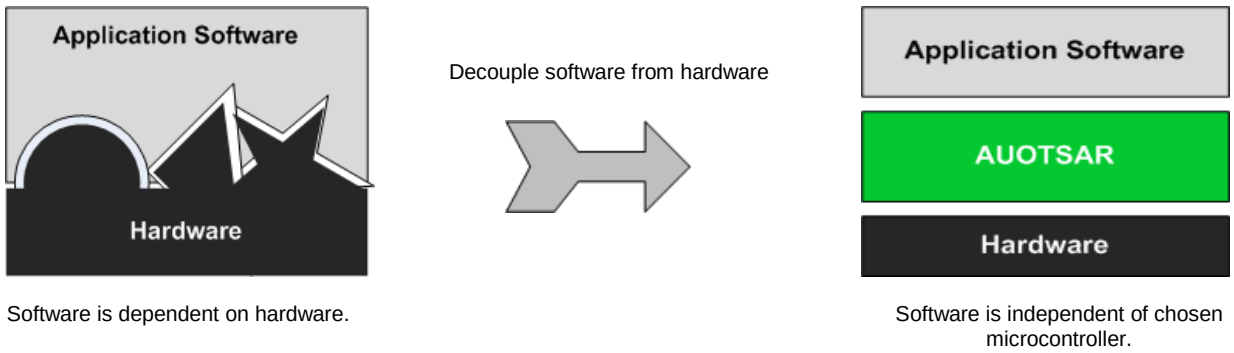

Figure 1.2: AUTOSAR decouples application software from hardware.

1.2 Motivation

From early releases of AUTOSAR a growing cooperation started for development of this standard that is expected to serve as a platform for future vehicle applications [9]. Consequently adding a new plugin to Populus Suite that can support automatic communication with AUTOSAR compliant applications not only elevates the functionality of this produce but also increases the market appeal of it as well.

The new plugin should offer required software for eliminating time consuming manual configurations needed to make this communication possible. For supporting such functionality in Populus, there should be an investigation on possible ways for mapping of data elements within AUTOSAR

application with those in the Populus Suite. Next, a suitable solution for generating the required software elements must be provided as well.

2. Theory

2.1 Populus

2.1.1 Structure

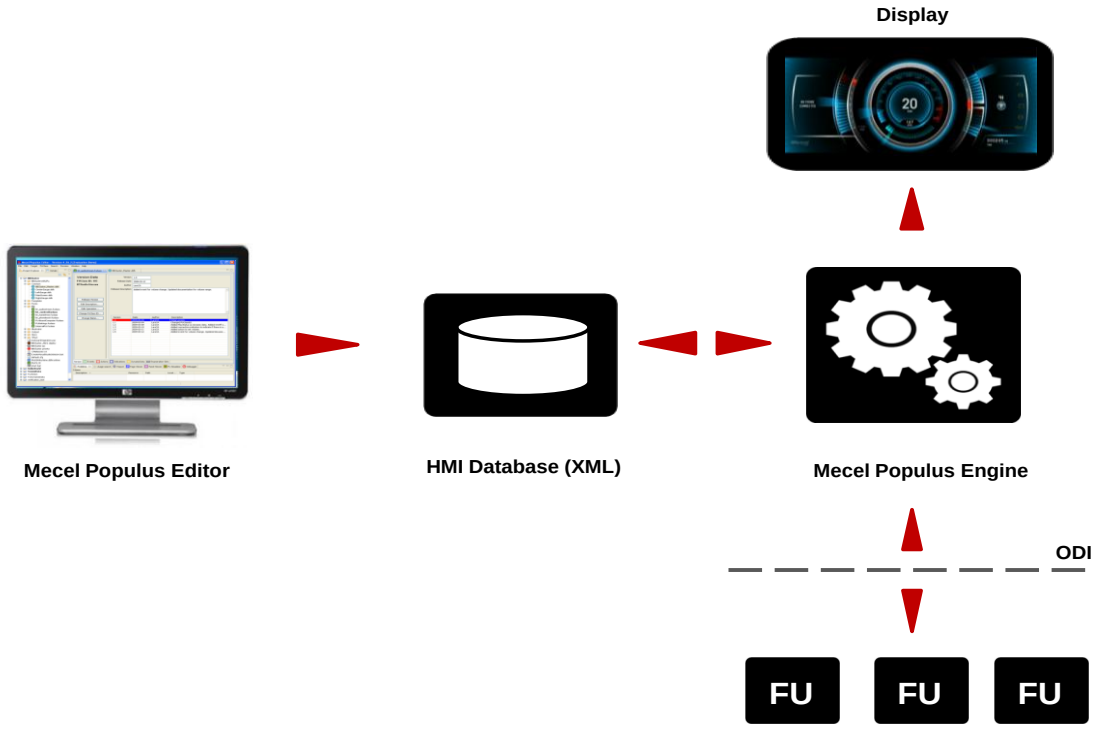

Figure 2.1: Mecel Populus Suite five major components

Mecel Populus Suite consists of five major components. The combination of these components provides the required platform for supporting further functionalities on this base.
As it is illustrated in figure 2.1 these five components are:

- *Populus editor:*
  Through the Populus editor user can design and specify all the required elements and features of the HMI. This includes graphical layout, visual effects, language, content of functional units, required data types etc.
- *HMI database:*
  All the static data from layout to logic of HMI are received from the editor and stored in an XML database.
- *Populus engine*

It executes HMI in run-time [5]. This means for running the HMI, Populus engine fetches required information from the database to render the graphical interface. After that, it receives user interaction with the interface as input and communicates with functional units.

- *Open Display Interface:*
  The ODI communication protocol decouples the functionality of HMI from functional units while they still can communicate with each other.
- *Functional Units:*
  They form the lowest layer of functionality in Populus. In other words, they are the implementation of actual functionality behind the graphical interface.

2.1.2 ODI

Open Display Interface plays an important role within the Populus Suite. The main purpose of this communication protocol is to provide simplicity in communication and flexibility in structure. Through the ODI protocol HMI and the FUs can function independently of each other while they still communicate and exchange information with one another. When it comes to downloading the designed HMI on a target platform the functional units and the displays can be placed on different ECUs. Therefore functional units are absolutely unaware of the number of existing displays and their location on the hardware.

ODI protocol is the only form of communication between Mecel Populus Engine and Functional Units. The protocol is designed for CAN but it is not limited to it and can work with TCP/IP and some other communication templates as well.

ODI messages are divided into four categories

- *Events:*
  FU sends these messages on its own initiative to notify the HMI about an event. Each message contains event ID and FU ID of the functional unit sending this message.
- *Actions:*
  HMI sends these messages to a specific FU. Action messages are divided into two groups of simple actions and value actions. Simple actions contain no data while the value actions carry an additional 32 bits of information. Actions are asynchronous function calls and their corresponding messages contain functional unit ID and action ID.
- *Indications:*
  These are boolean flags which are sent by the FU on its own initiative and broadcasted to all Populus Engine instances. The message contains functional unit ID and a bit field indication for showing true or false.
- *Dynamic Data:*
  These messages are sent to the Populus Engine upon its request or prior subscription. Therefore the engine informs the FU that it is interested in receiving a certain value, in other words the engine subscribes a dynamic data. After that FU sends a response message to the engine every time that particular value changes. The response messages will continue till the engine unsubscribes that dynamic data. A data request message that is sent from engine to a particular FU, contains the FU ID and one to five data IDs which the engine is interested to subscribe or unsubscribe. In return the response message that is sent from a particular FU contains FU ID, data value and data validity state.

According to [11], “ODI is an application layer protocol that runs on top of other communication protocols”. Consequently the message type definitions may differ based on the underlying communication protocols (TCP/IP, CAN, FlexRay and etc.).

2.1.3 Populus workflow

Figure 2.2 illustrates an example of HMI workflow for two types of ODI messages action and dynamic data.

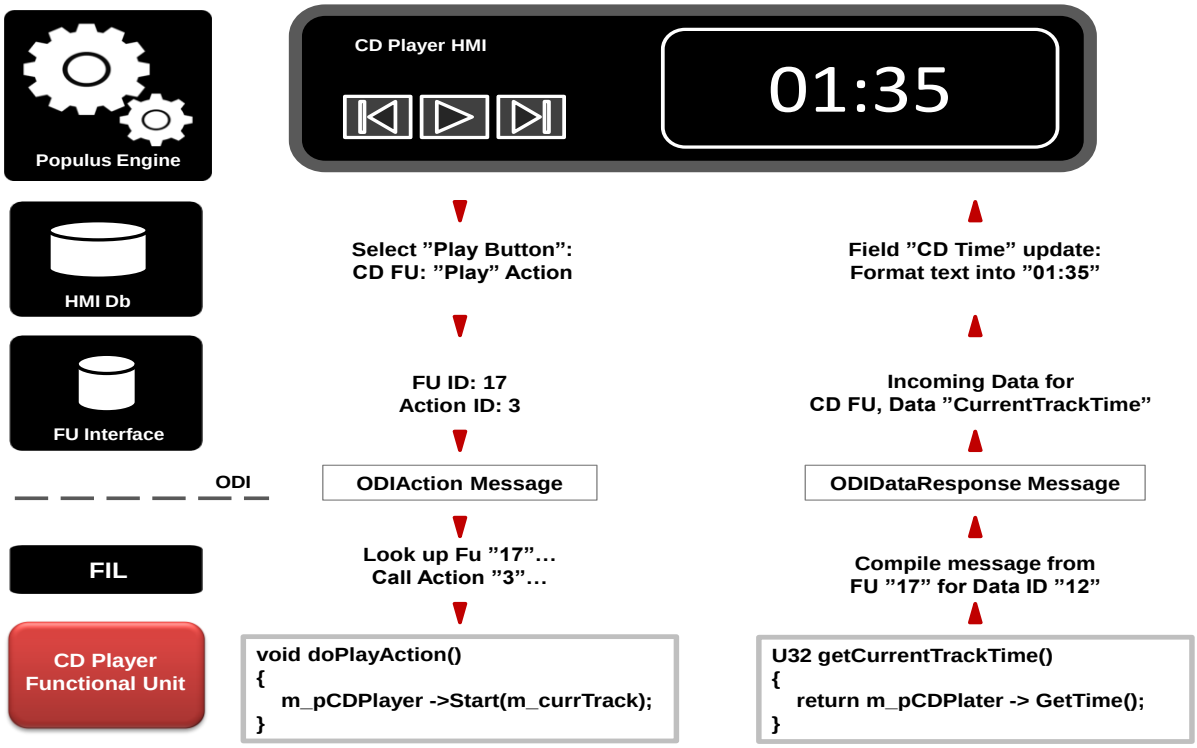

Figure 2.2: Populus workflow for action (left) and dynamic data (right).

2.1.4 Functional Unit (FU)

Functional Units are part of Populus Suite and where the actual functionality behind the visual interface lies. Functional Units and Populus Engine are completely self-contained and they do not have any compile dependencies on each other. Therefore, they can only communication through ODI protocol.

User can design HMIs graphical layout and logic through Populus editor. Here the logic refers to behavior of the designed interface i.e. how the user can interact with it and what HMI does as the result of this interaction. The operation of HMI is defined in Functional Units, but in editor it is only the description of the FUs which are saved in FU Interface (figure 2.2). This means that from the engines point of view FU is just an ID which points to a certain interface. Each Functional Unit can contain the interfaces of several ODI message type e.g. event, action, indication and dynamic data.

Interfaces are identified by a tuple of FU ID and interface ID. For every FU a C++ code is generated automatically from the Populus Editor.

2.1.5 Functional Unit Software Development Kit (FUSDK)

Mecel Populus offers a FUSDK which contains the implementation of ODI and the FUs. The SDK consists of two main parts (figure 2.2) and can be generated in C++, Java or Python through Populus editor after HMI being designed. These two parts are FIL and the FUs.
FIL is the interface layer between ODI and FU implementation and it handles ODI data subscriptions. As an example if FIL receives an ODI message for subscribing a dynamic data value, from the time of subscription it will sends a response message to the Populus Engine upon the value change in that specific dynamic data. In this way it will inform the Engine about the latest value of that dynamic data.
FUs are the lowest layer of functionality in the generated code and they are interfaces of the FUs defined through the Populus Editor. It is upon the programmer to define the functionality of each FU and implement the callback functions (figure 2.2).

2.2 AUTOSAR

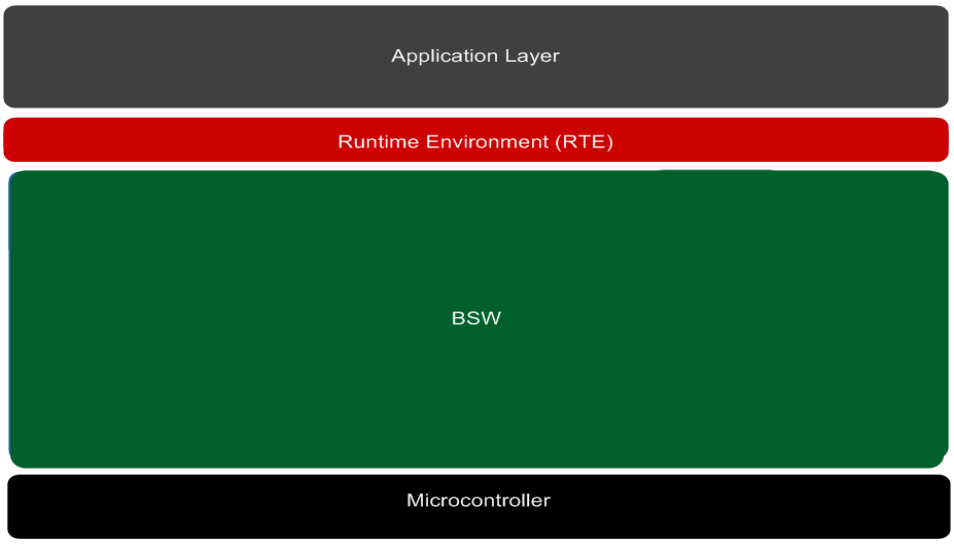

Figure 2.3: AUTOSAR main software layers.

2.2.1 General view

Figure 2.3 illustrates three main software layers of AUTOSAR architecture on the highest level of abstraction [4]. The first layer on top is application layer which encapsulates the functionality of the system. The second layer is RTE which provides the communication services to the application layer [4]. The third layer is BSW which provides basic services for the application layer. BSW is further divided into four sub-layers (figure 2.4).
Microcontroller Abstraction Layer (MCAL) is the lowest layer within BSW and it has direct access to the on-chip microcontroller (MC) and internal peripherals. It makes higher layers independent of microcontroller via specific drivers.

The next sub-layer is ECU Abstraction Layer which provides drivers for external devices and makes upper layer independent of the location of MC and internal/external devices.
Service layer provides basic services for application and basic software modules [4] e.g. vehicle network management, memory management, operating system, diagnostic services and etc.

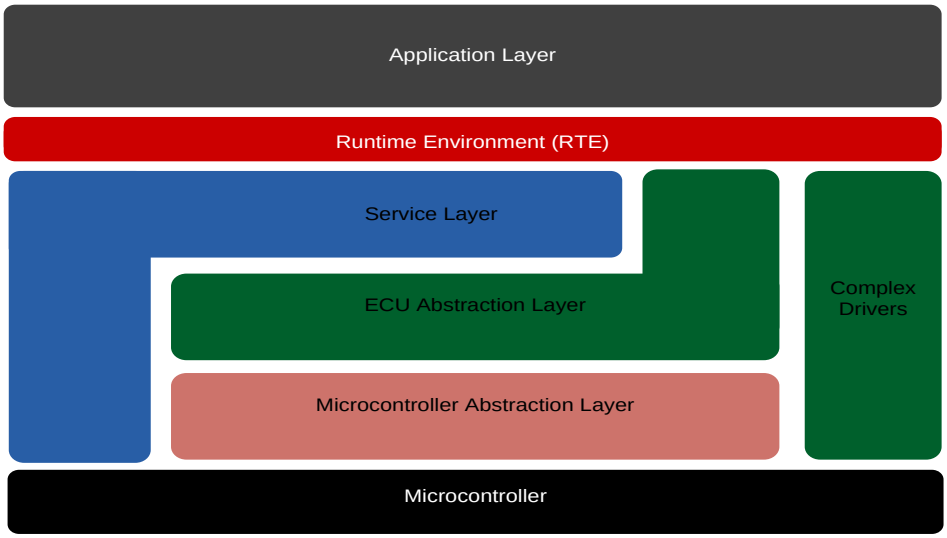

Figure 2.4: BSW sub-layers

Finally, Complex driver gives the possibility of integrating special purpose functionalities which are not defined in AUTOSAR or have significantly high timing constrains [4]. This layer can have direct access to hardware through bypassing the BSW.

2.2.2 Basic concepts

Under AUTOSAR standard the E/E infrastructure in a vehicle is distributed over several ECUs which each of them obey from the AUTOSAR software architecture and are connected through a sort of communication bus to each other. Four basic concepts in AUTOSAR are Software Component (SW-C), Virtual Functional Bus (VFB), Run Time Environment (RTE) and Basic Software (BSW).

2.2.2.1 Software Component (SWC)

The application layer (2.2.1) in AUTOSAR consists of interconnected SW-Cs [1]. SW-Cs are placed on the highest level of abstraction and they are unaware of their location (on a specific ECU) in the system.

Simple AUTOSAR SW-Cs are of type “Atomic Software Component”. This means that each instance of software component can be assigned to just one ECU and cannot be distributed over several ECUs [1]. “Sensor/Actuator” is a special subset of “Atomic Software Component” which contains the application dependencies on a specific sensor or actuator.
AUTOSAR also offers a more generic definition of SW-C in a sense that “a SW-C can be a logical interconnection of other components” [1]. Such a SW-C is called “Composition”. Composition can also be distributed on several ECUs.

AUTOSAR SW-Cs are able to transfer data between each other by using ports and interfaces (figure 2.5). Also, each SW-C in AUTOSAR has a so-called “internal behavior” which consists of one or several “runnables”. Internal behavior describes the scheduling aspects of a component such as activation of the runnables and events that they are responding to [1]. Also runnables are the smallest functions within a software component which have direct access to the SW-C port.

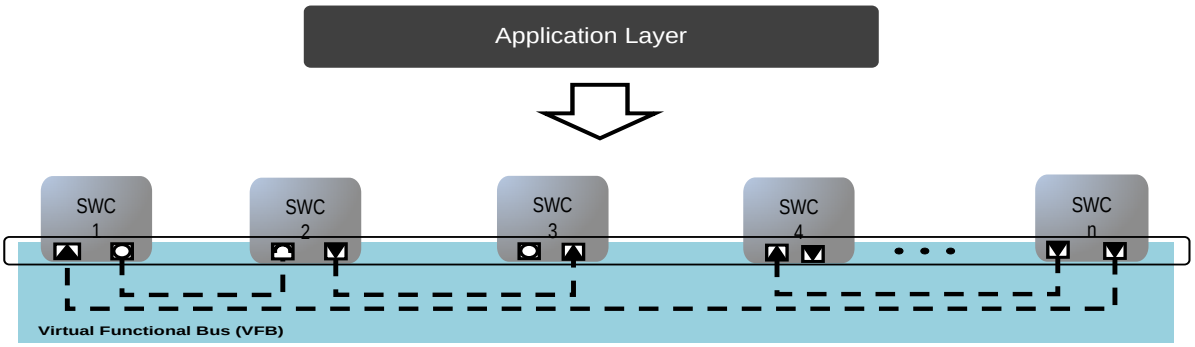

Figure 2.5: Example of SWCs in AUTOSAR

In essence, shipment of a complete SWC in AUTOSAR consists of [1]:

- A SWC description which contains the properties and behavior of a SWC.
- An implementation of the SWC which can be provided either as an object code or a source code

2.2.2.2 Virtual Functional Bus (VFB)

VFB is a communication mechanism that separates SW-C from the underlying infrastructure and provides the relocatability of software in this architecture. In VFB concept, SW-Cs of the entire vehicle can be virtually connected to each other during the design phase. In other words, VFB hides the difficult connection details from the designer. Therefore, SW-Cs can be virtually connected regardless their location on ECU (figure 2.6).

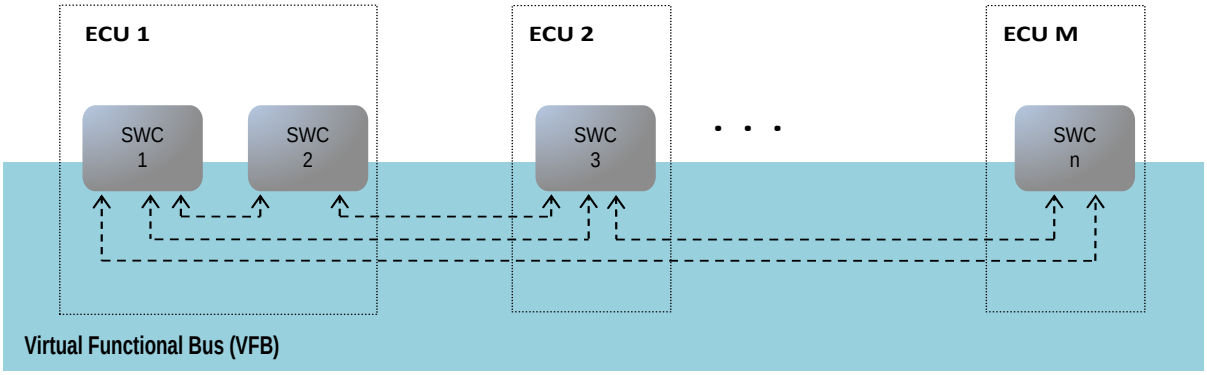

Figure 2.6: SWCs are connected virtually during the design phase

2.2.2.3 Runtime Environment (RTE)

RTE is the implementation of VFB in runtime. It is placed between the application layer and the basic software layer to decouple the functionality of them from each other. RTE hides the network from the application. Therefore, communication between the SW-Cs located on the same ECU is limited to

RTE level. But when it comes to inter ECU communication RTE also offers the necessary interfaces for using COM services which are provided by the BSW. The code for RTE is being generated by RTE generator based on ECU configuration description file (more about this in 2.2.3).

2.2.2.4 Basic Software (BSW)

BSW is a standardized layer which contains ECU specific components. It provides the required services by the software component and does not fulfill any specific functionality on its own [1].
BSW provides:

- Services such as NVRAM, diagnostic protocols, memory management and etc.
- Communication bases (CAN, LIN and FlexRay), I/O management and network management.
- OS service for scheduling
- Microcontroller Abstraction Layer (MCAL) for access to hardware on a higher level.
- Etc.

BSW configurations are stored in an standard AUTOSAR XML file called Basic Software Description (BSWD).

2.2.3 Approach

AUTOSAR approach explains the structure of application software under ATUOSAR standard.
As it is mentioned previously in (2.2.1.1), the application software in AUTOSAR is distributed over smaller units called SW-C. Each SW-C contains a part of the applications functionality and it needs to communicate with other SW-Cs to work in integration as of the initial monolith application.
In general the main specifications of a SW-C can be summarized into:

- *Port:*
  Ports are considered as the communication gateway for SW-C. Each port is one of the provider or required type. A Provider port sends a value out of SWC and a required port receives a value from another port.
- *Interface :*
  Each port in AUTOSAR requires an interface. Interface defines the transferring data elements as well as the semantics of the transfer. Interfaces are of two types of sender/receiver (S/R) and client/server (C/S). A S/R interface is a one way communication between the two sides of transmission while a C/S interface is a two way communication. In C/S client asks for a value and then it will wait for the server response.
- *Data Element (DE):*
  Each port interface contains one or several DE which is the ultimate parameter being transferred between the SW-Cs on the same ECU. Each DE has a data type.
- *Internal Behavior:*
  It defines the behavior of SW-C and also provides the required functions, variables, events and other internal data needed by the SW-C [3].
- *Runnable:*
  A runnable is a schedulable function that has access to SW-C ports. RTE provides the runnable APIs and also triggers it according to a specific schedule defined in SW-C description.
- *Event:*

Each runnable is being triggered by an RTE event. There are different types of RTE events but the general simple form is called “TimingEvent” in which RTE triggers the SW-C component over a constant frequency.

All these information are packed into a SW-C description file (figure 2.7). During early design phase, SW-Cs are connected virtually through VFB. After specifying the connections each SWC will be mapped to an ECU. Depending on whether two SWC are placed on the same ECU or on two different ECUs, they can communicate through RTE or available communication platform.

In AUTOSAR there is a configuration description file for each ECU. This file contains all the required (SW-C, BSW and RTE) configurations for an AUTOSAR ECU. For each newly added SW-C, its description file must be integrated with the ECU description file. The ECU description file is an input for the SCG (Source Code Generator). Having all the necessary configurations SCG will generate the C code of BSW modules configurations and RTE APIs for that ECU. The generated code and the static implementation of BSW modules together will provide a compilable application for this AUTOSAR design.

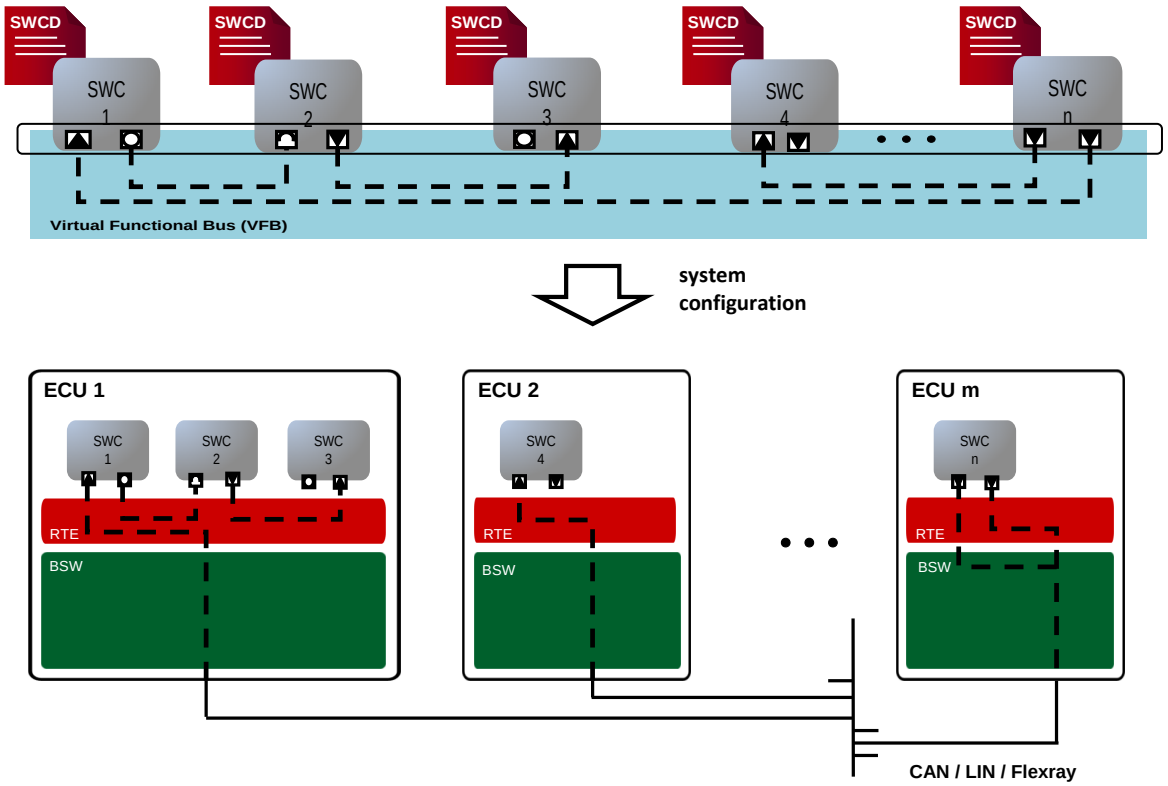

Figure 2.7: AUTOSAR basic approach.

3. Tools

3.1 Mecel Populus Suite

Mecel Populus was the base of provided solution in this project. As the result of solution, a new plugin called “AUTOSAR Integration” is added to this application.
Moreover, for testing the final solution and viewing the result of data transmission between Populus and AUTOSAR we were requiring a HMI for visualizing these results. Therefore, Mecel Populus is used for design and implementation of this HMI as well.

3.2 Mecel Picea Suite

Mecel Picea Suite contains the RTE, BSW and the configuration and integration tool chain. Also it is a fully ICC3 compliant AUTOSAR suite [10] (AUTOSAR ICC3 conformance class contains the most detailed interface granularity).

Picea Suite is consisted of the following main parts:

- *Picea Integration Package (PIP):*
  PIP is the stack implementation of AUTOSAR. It contains both RTE and BSW.
  Picea RTE is a command line code generator which requires the AUTOSAR configuration files as an input. RTE generator reads the configuration file and validates them according to AUTOSAR schema [10] and generates the RTE C code.
  Picea BSW is consisted of SCG and standard software core for each BSW module. Here again the SCG is a command line code generator which receives the ARXML configuration files and generates their corresponding C files.
- *Picea Workbench:*
  Picea Wokbench is consisted of an AUTOSAR authoring tool. Configuration of RTE and BSW is the main use case of this tool. But it can also be used for editing the SW-Cs and building the ECU application layer [10]. More about Picea Workbench authoring tool is in (3.3).

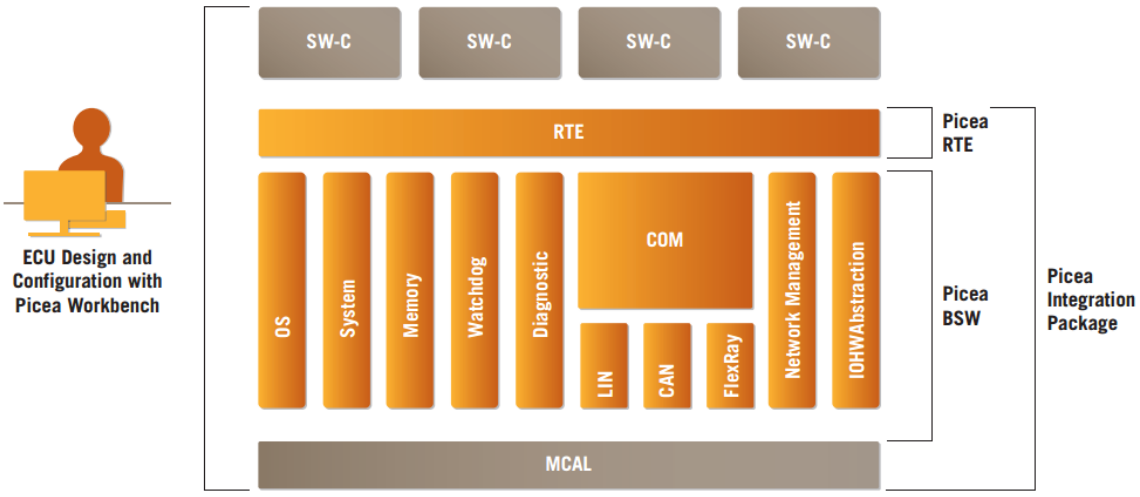

Figure 3.1: Picea Integration Suite (figure from [10])

3.3 Volcano VSX

AUTOSAR standardizes the automotive software architecture but the process of design and deployment of this architecture is a complicated flow. Volcano VSX tool chain offers the necessary tools for an organized design and deployment of AUTOSAR system models. VSA and VSB are two of the VSX tools which are mainly used for the model design and configuration in this project.

3.3.1 Volcano Vehicle System Architect (VSA)

VSA is a tool for design of automotive software and hardware architectures and management of their relations under AUTOSAR standard. System design in VSA includes:

- *SWC definition:*
  Defining required SW-Cs (2.2.1.1, 2.2.3) and their connections with the rest of system.
- *Mapping of SW-C to ECU instance:*
  Each SW-C is mapped to its associated ECU instance. Later, a Communication Matrix (COM Matrix) is defined according to this mapping.
- *ECU topology:*
  It describes the structure for interconnection of ECUs through the underlying physical communication bus [13].
- *Data mapping:*
  SW-Cs located on the same ECU are transferring data between each other using DEs (2.2.3 DE). On the other hand, if they are located on different ECU instances, data transmission is done through the system signal which is sent over the physical communication bus. Data mapping is mapping of data elements (those which are required to be transferred over the communication bus) to the system signals.
- *Etc.*

The result of VSA is an ECU extract for each existing ECU in the system.

3.3.2 Volcano Vehicle System Builder (VSB)

VSB is an AUTOSAR ECU configuration editor that refines the final output (ECU extracts) received from VSA and creates a complete ECU configuration. When VSA main functionality is centered on software architecture and communication design, VSB focuses on applying BSW configurations for a specific ECU.

In general ECU configuration process consists of extracting ECU description system description file for each ECU and configuring each AUTOSAR BSW module according to the special needs of that specific ECU [14].

4. Solution

4.1 Design

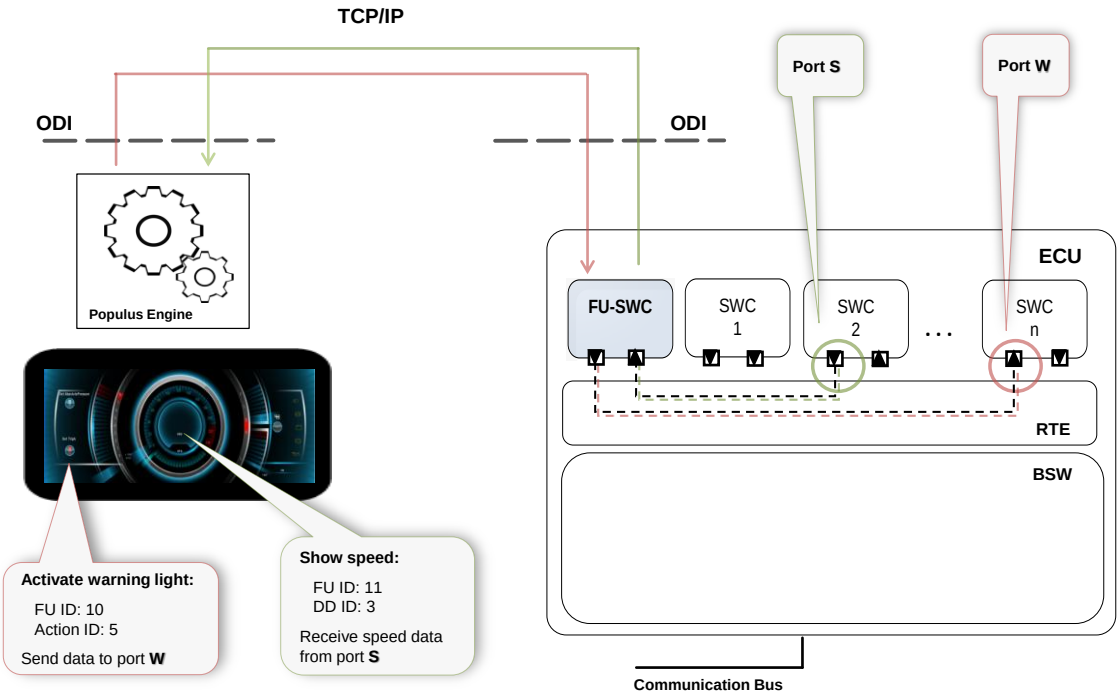

Figure 4.1: Data transmission

Generally the transmitted data between HMI and AUTOSAR can be divided into two categories. The first category contains the data sent from HMI to AUTOSAR. According to section 2.1.2, HMI can send a simple parameter from the HMI engine to the Functional Units in the form of a simple value action message. This message type is one and only proper form for sending data from HMI to AUTOSAR.

The second category contains the data received by HMI from AUTOSAR. According to section 2.1.2 HMI can receive a simple parameter from Functional Units in the form of dynamic data ODI message type. Dynamic data is sent from Functional Unit to the Populus engine upon engines request. This property makes dynamic data a unique and suitable form for our purposes in the current solution.

Although we have found the appropriate ODI message type for our data transmission purposes, the number of parameters used in each transmission can vary significantly between different designs. Thus, placing a new software component within AUTOSAR system that can act as the representative of HMI needs can be a wise strategy for handling these variations (figure 4.1). We name this new software component “FUSWC”.

FUSWC is responsible for collecting the data required by HMI from AUTOSAR and also delivering the data sent from HMI to a targeted port within AUTOSAR system. To enable the FUSWC with such functionality we must define the necessary properties and behaviors for it in its description file.

After that there should be an intuitive way for the user to map each HMI ODI message type to an AUTOSAR data element type. This can be done through a mapping phase where the user can import the ATUOSAR system description file into Populus application and receive all the information about the available ECUs, SW-Cs, ports and etc. Then the user will be able to select the correct ports as a resource or target for each data transmission.

The result of mapping phase will be an input for generation of the FUSWC description. FUSWCD contains all the prerequisites for integration of FUSWC and system description file. By integrating these two we have a new AUTOSAR system file which also contains the representative of HMI as well.

4.2 Implementation

4.2.1 Extracting information

It is upon the user to decide what data transmission needs to take place between the designed HMI and AUTOSAR system. Therefore, user must be informed about the available data in both HMI database and the AUTOSAR system. For this reason we should extract the necessary information from both of these two systems and illustrate them within one view to the user.

4.2.1.1 HMI

All the defined data in Populus are stored in HMI database (2.1.1). The data is placed within Functional Unit and it can be one of the forms of action, dynamic data, event or indication.
The proper data formats for this solution are value action and dynamic data (4.1). Thus, these are the only data structures that the user needs to have access to.
In order to extract the available actions and dynamic data from the HMI database, we have employed Populus own EMF (figure 4.2). This EMF contains a complete model of all data structures defined in Populus and can offer a full functionality for extracting the required information.

4.2.1.2 AUTOSAR

The information that user requires to have from the AUTOSAR system includes all existing ECUs, available SW-Cs in each ECU, existing ports in each SW-C, corresponding interface for each port etc. (figure 4.2). This information can be found in system description ARXML file. But for extracting them we need to employ a suitable modeling framework which can support ARXML 4.0.3 files. We have used Artop 4.0 EMF as the model in this solution.

AUTOSAR Tool Platform (Artop) is a demonstrator which is designed for pars and edition of AUTOSAR XML files. It is also being offered as a plugin in VSX tool (3.3). Artop provides an Eclipse Modeling Framework (EMF) which the programmers can employ to make their desired operation on the standard AUTOSAR XML files. Figure 4.2 shows a general view of extracted information from system description file.

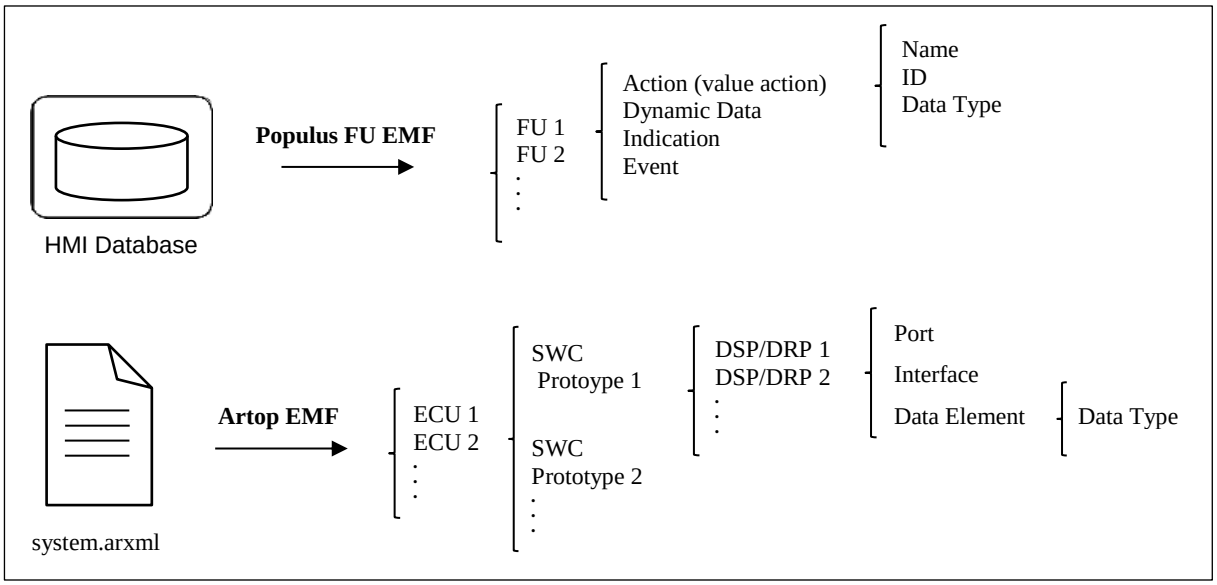

Figure 4.2: Extraction of data from system description files.

4.2.2 Mapping

The User can have access to the extracted information from the Populus and AUTOSAR (4.2.1) through one editor in "AUTOSAR Integration" plugin. This will simplify the mapping process by providing a quick and complete view over all available information at once. User can map each provider port from the AUTOSAR system to an action data structure and also each required port to a dynamic data structure (4.2.1).
A completed mapping will be saved for the generation of FUSWC description in the next step.

4.2.3 SWC description generation

According to the user mapping a complete description of FUSWC can be automatically generated from the editor. For generation of this SWC description the program analyzes user mappings first. By checking this mapping it can find which SWCs, ports, data elements and what data types are involved in this mapping. Therefore it can decide what kind of ports, data elements, data types and etc. the generated FUSWC needs to support as well. Figure 4.3 gives a general view over the information included in FUSWC description.

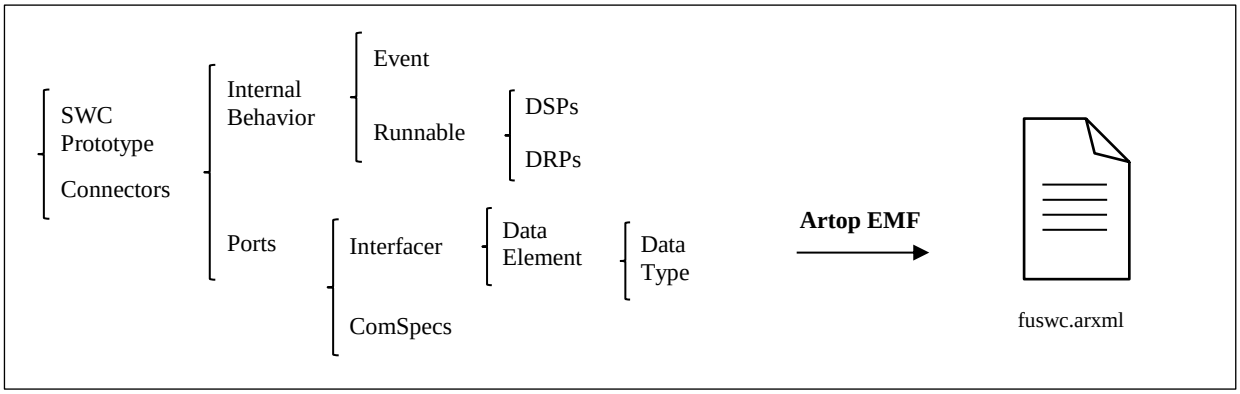

Figure 4.3: Generating software component description.

FUSWC can be connected to the rest of the structure defined in system description through connector. Connectors used for connecting SW-Cs within one ECU are called *assembly connectors*. Each *assembly connector* connects one provider port to one required port. Consequently for Integrating the generated FUSWC with the rest of system we need an assembly connector per each available port in this SW-C. These connectors are also generated as a part of FUSWC description to make the integration process easier for the user.

4.2.4 Generation of AUTOSAR compliant FUSDK

The FUSDK is the implementation of ODI protocol and the FU interfaces (2.1.4). The simple form of generated SDK from Populus contains the interfaces for the FUs but it is upon programmer to define their functionalities by implementing their callback functions (2.1.4).

After mapping of HMI data to AUTOSAR data (4.1, 4.2.2) by the user in the newly added plugin (ATUOSAR integration plugin), an AUTOSAR compliant version of the regular FUSDK can be generated. In this new version the functionality of the FUs are specified and their callback functions are being defined and implemented. All of these are done based on user mappings.
In other words, by checking user mappings, the application will generate the correct RTE read and write interfaces and it will also place them in the correct FU message interface. All generated RTE interfaces belong to FUSWC. The reason for it is that FUSWC is now the Populus gateway for data transmission with the rest of AUTOSAR system. In addition, RTE interfaces are generated according to AUTOSAR standard API reference [8] which makes them completely independent of any specific implementation of AUTOSAR architecture.

Generally RTE APIs for reading and writing a data values differ based on their actual functionality and the type of value they transmit. Hence for employing the correct API we need to identify the factors which are important in this transmission. Considering our solution properties, according to AUTOSAR standard APIs [8] the RTE interfaces for reading and writing of data elements with “data” semantic, over a sender-receiver communication bus, is an appropriate interface that suits our case (appendix A). Each RTE call interface consists of several arguments such as port name, variable data prototype and data type. Therefore, it will be different for each data element on a specific port. To generate the correct interface for each FU call back function we should check both user mappings and FUSWC description to be able to conclude the correct arguments for the corresponding RTE interface.

4.2.5 Generation of SWC implementation and required heading file

The actual functionality for the correct data transmission over RTE is placed within the FU callback functions in the generated AUTOSAR compliant FUSDK (4.2.4). But the activation of the whole FUSDK containing both FIL and the FU interfaces (2.1.4) starts from the FUSWC runnable.

A SW-C implementation C code that contains the FUSWC runnable can be automatically generated from the solution plugin. The signature for the runnable is generated according to standard AUTOSAR runnable signature [8] and it is independent from any specific AUTOSAR implementation (appendix A).
A runnable is a schedulable function (2.2.1) and it can be triggered with a certain period that is defined during the generation of software component description. Therefore, activating the FUSDK

through runnable will change the frequency of ODI message passing. By defining a relatively high frequency for FUSWC runnable in its generated SW-C description we can decrease the possibility of missing ODI messages.

4.2.6 Data type and scaling

In the mapping phase, user should pay attention to data types in both the Populus and AUTOSAR system in order to avoid losing data during transmission. All the related ODI data types and AUTOSAR data element types are extracted and illustrated to the user in the mapping table.

In AUTOSAR there exist no predefined data types. Therefore, each company or OEM must define its own data types.
Generally AUTOSAR data types are defined within three levels of abstraction [3]:

- *Application Data Level*
  This level allows defining data types from the application point of view and it is possible to specify all the types required by VFB on this level. But the data types at this level cannot be used by RTE.
- *Implementation Data Level*
  It is introduced to optimize the data type on the implementation level which is close to the programming language like C [3]. For each data type defined on the Application level, there should be a mapping to an Implementation level data type.
- *Base Type Level*
  It provides the platform dependent part of the Implementation data types [3] which are defined in the form of bits and bytes.

In Populus, ODI data types are in one of the forms Simple Data or Non-numeric Data. Simple Data Types are limited to 32 bit and are always sent using Metric values on the bus (temperature, pressure, speed and etc.). Non-numeric Data Types are in other forms such as String, List, Bitmap and etc. Among these two, Simple Data is the only suitable value type for the data transmission with AUTOSAR.

In the AUTOSAR compliant version of Populus SDK which can be generated from the added solution plugin to Populus, all FU callback functions are implemented so that each contains the correct RTE read/write interfaces for data transmission according to the user mappings. But for generating the correct callback function we should consider the type casting and scaling of data from AUTOSAR to Populus and reverse. Figure 4.5 shows a sample of generated call back functions for dynamic data and value action ODI message. The upper function in this figure belongs to a specific dynamic data in Populus which receives the value of “CoolantTemperature” form AUTOSAR. The generated RTE call for this interface is a RTE read by argument interface. In order to be able to read the value by argument we need to generate the correct data type for using with this RTE call.
As you can see the “coolantTemperature” variable is defined with “uint16” type. Here “unit16” is an implementation data level in AUTOSAR. This data types are being extracted during loading of the ARXML system file into the “ATUOSAR integration” plugin (4.2.1). Also the value of “coolantTemperature” is scaled before returning it to the Populus Engine. This scaling will take place according to the internal scaling of parameters inside the Populus (appendix B).
Likewise, the same procedure is applied for the action ODI interface as well.

```
43
44  U32 BoardComputer::getCoolantTemperature(bool& valid)
45  {
46      valid = true;
47      uint16 coolantTemperature;
48      (void)Rte_Read_R_FuSwcReceiveCoolantTemperature_CoolantTemperature(&coolantTemperature);
49
50      return (U32)coolantTemperature*1024;
51  }
52
53  void BoardComputer::doValueActionResetAction(U32 val)
54  {
55      (void)Rte_Write_P_FuSwcSendTripA_TripA((uint8)val);
56      (void)Rte_Write_P_FuSwcSendAbsolutePressure_AbsolutePressure((uint8)val);
57  }
58
```

Figure 4.5: Generated FU callback function for Dynamic Data (upper function) and Value Action (bottom function).

4.2.7 Integration of generated SWC description with system description

According to user mappings a FUSWC description will be generated from the editor (4.2.3). This new SW-C contains the required properties for data transmission with the rest of AUTOSAR ECU. In order to generate the necessary RTE interfaces for this FUSWC, we need to integrate it with the rest of the AUTOSAR system file. This integration will be done through the VSX tool (3.3). By importing both system and FUSWC description files into the same project, we can connect the FUSWC with the rest of SW-Cs on the ECU. An instruction will be automatically generated with the FUSWC description from the editor. This instruction provides the information for making the correct assembly connector for each connection. For making the connection process easier these connectors are also being generated as a part of FUSWC description and the user can copy the provided instance. Finally the necessary BSW configurations are done for the FUSWC and the system description will be ready for generating RTE.

5. Result

In this section we will review the result of the implementation phase and its final outputs.
Figure 5.1 illustrates the resultant plugin solution in a glance. As is shown in the editor snapshot, user can import the system description file into this plugin and explore the content in a tree view format. The table at the bottom is the mapping table which enables the user to map AUTOSAR data to Populus ODI messages.

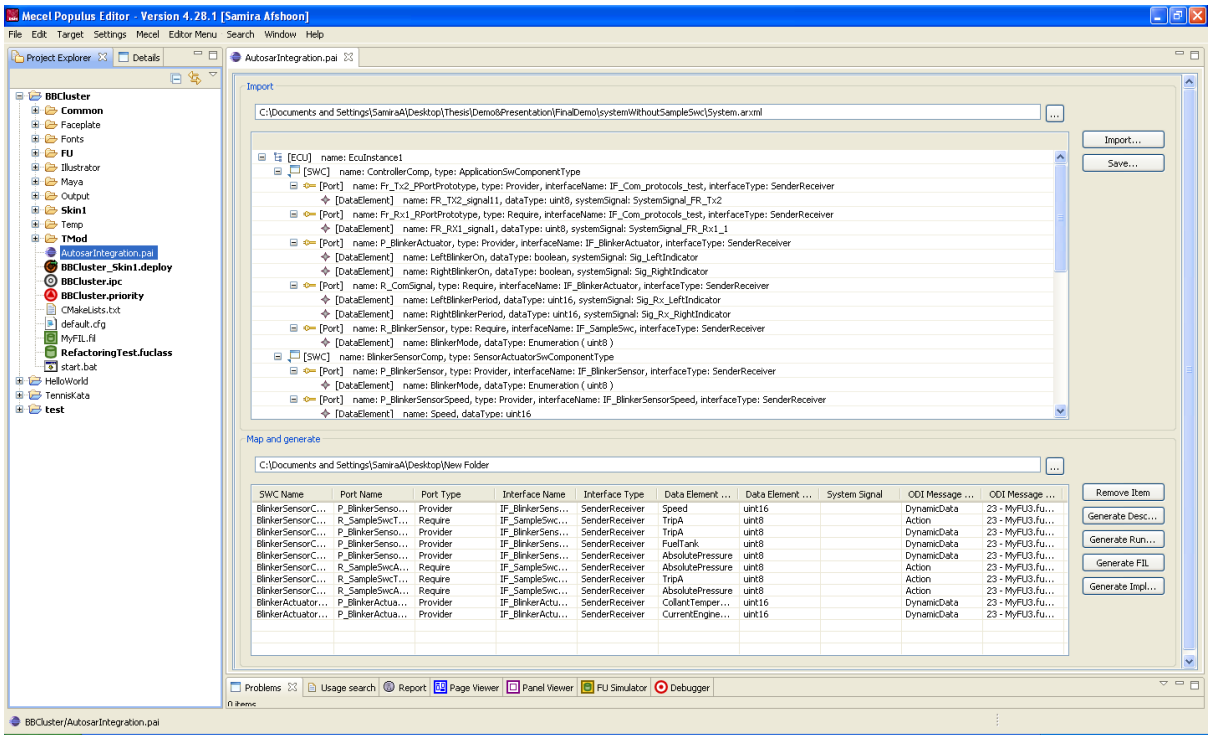

Figure 5.1: A snapshot of the new plugin “AUTOSAR Integration” which is added to Populus Editor as the result of the provided solution in this thesis.

In general the following files can be generated from the solution editor:

- A FUSWC description file and its corresponding integration instruction (4.2.3), (4.2.7).
- A FUSWC implementation (C file) and linking heading files (4.2.5).
- An AUTOSAR compliant version of Populus SDK containing implemented FU callback functions (4.24).

Dispatching of dynamic data ODI request and ODI action message from Populus engine are two main forms of initializing data transmission. Figures 5.1 and 5.2 are illustrating the sequence diagrams of the solution workflow in response to these two ODI message forms.

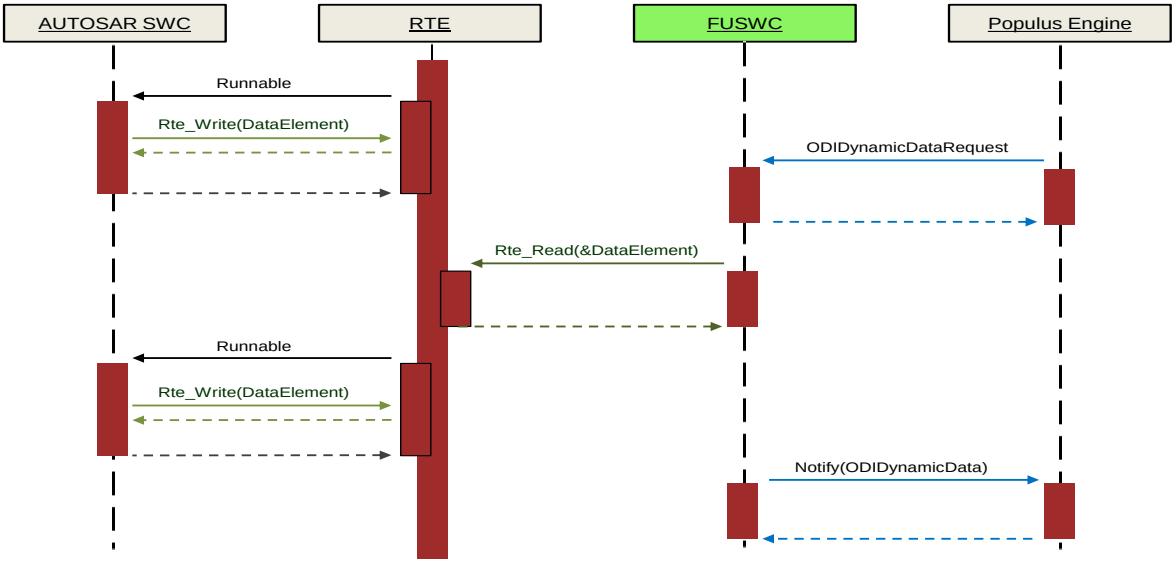

Figure 5.1: Sequence diagram for dynamic data ODI request.

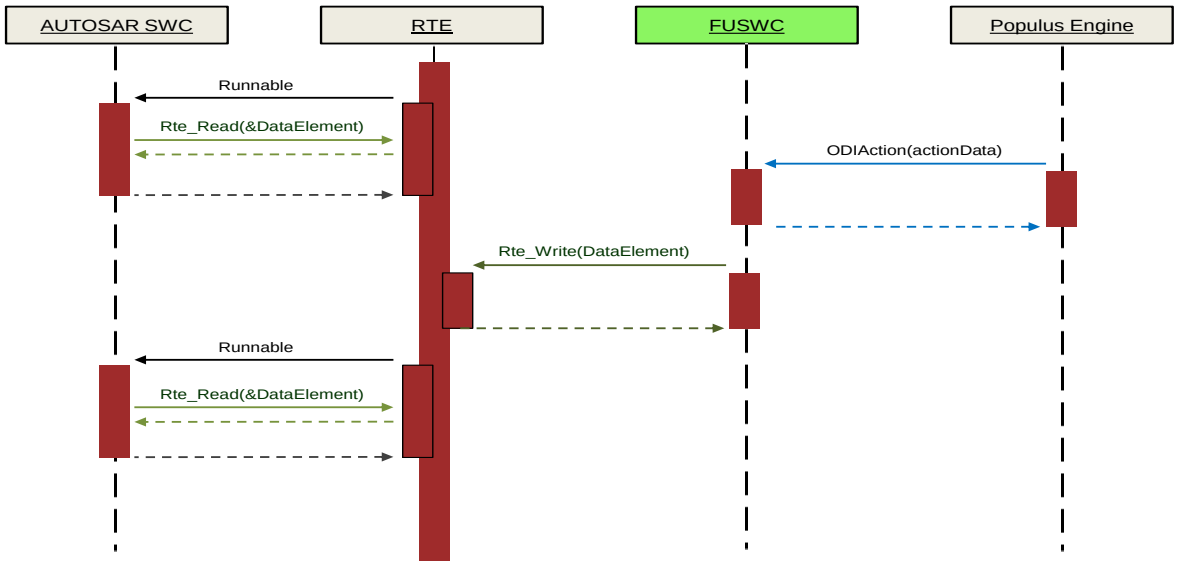

Figure 5.2: Sequence diagram for ODI action message.

In brief, “AUTOSAR Integration” plugin provides all the necessary implementations automatically and reduces the time for making the base of this data transmission significantly. Also it hides most of the implementation procedure from the user and reduces the necessity of having detailed knowledge regarding this process for the user.

6. Discussion and future work

In general the idea of providing the required base for data transmission between HMI and AUTOSAR automatically is in its initial stage. The completion of the solution can be relative to customer needs and use cases.

In the current solution Populus HMI is the starting point for user mappings and code generation. Correspondingly the implementation of solution is partly Populus specific but its overall design can be extended to fit in with other HMI development tools as well.
Besides for accommodating the whole project process into a limited time frame we had to make strict assumption in different stages of implementation. Having a variety of options for implementing the same functionality based on user requirements, we had to choose one of them. Our choices in this area were made based on the customer preferences from Volvo.

In addition, the provided solution is designed specifically for PC testing and the data communication between Populus and AUTOSAR is over TCP/IP. But for testing the solution on a target platform the communication protocol have to change to CAN, LIN or Flexray. In this case, communication is made through the AUTOSAR communication stack which is consisted of three application layers.
At RTE level a data element can be mapped to a signal which transfers the data values through the lower software layers to the communication bus. But this process requires its own configuration in AUTOSAR.

From ATUOSAR perspective two major extensions of the current solution can be:

- Auto-configuration of BSW for the generated FUSWC.
- Auto-generation of required properties and configurations for sending and receiving the ODI messages over CAN/LIN/FR through FUSWC.

Likewise, two major extension of implemented solution from Populus perspective can be:

- Generating a FU definition according to the selected AUTOSAR data by user.
- Investigating the possible use cases for the two other ODI message types, events and indications.

7. Conclusion

The added solution plugin “AUTOSAR Integration” to Mecel Populus HMI Suite eases the processes of integrating designed HMIs with AUTOSAR applications. In other words, the necessary materials for making this integration can take a considerable amount of time and require a high level of knowledge about both systems. But through this plugin these materials from ARXML description files to all the required implementation code can be generated automatically within few seconds. Furthermore this automation can omit the possible errors which may occur during the manual configuration as the result of user mistakes.
As it has been mentioned earlier, the solutions are according to general AUTOSAR standards and completely independent of any specific implementation of this architecture. Although the HMI part of the implemented solution is specific to Mecel Populus HMI suite but the idea and theory of the

solution is addressed in general form and can be further extended and modified for other HMI design tools as well.

References

1. AUTOSAR Technical Overview, AUTOSAR Auxiliary 067, 2008.
2. AUTOSAR. (2012). AUTOSAR General [Online]. Available: http://www.autosar.org/index.php?p=1&up=6&uup=0
3. AUTOSAR Software Component Template, AUTOSAR Standard 062, 2011.
4. AUTOSAR Layered Software Architecture, AUTOSAR Auxiliary 053, 2011.
5. Mecel AB. (2011, May). Product brief – Mecel Populus Suite [Online]. Available: http://www.mecel.se/products/mecel-populus
6. Mecel AB, Populus Specification, 2012.
7. Mecel AB, MecelPopulusTraining: Mecel internal documentation.
8. AUTOSAR Specification of RTE, AUTOSAR Standard 084, 2011.
9. Florian Leitner. Automotive Open System Architecture [Online]. Available: http://www.inf.uni-konstanz.de/soft/teaching/ws07/autose/leitner-autosar.pdf
10. Mecel AB. (2011, May). Product brief – Mecel Picea Suite [Online]. Available: http://www.mecel.se/products/mecel-picea/Product.Brief.Mecel.Picea.pdf
11. Mecel AB, Populus ODI Protocol Specification, 2012.
12. VINNOVA. (2011). Second Road Phase 1 [Online]. Available: http://www.vinnova.se/sv/Resultat/Projekt/Effekta/Second-Road-fas-1---En-gemensam-simulatormiljo-for-aktiv-sakerhet-och-HMI-inom-Volvo-Personvagnar/
13. AUTOSAR System Template, AUTOSAR standard 063, 2011.
14. ATUOSRAR Specification of ECU Configuration, AUTOSAR Standard 087, 2008.

Appendix

A. RTE interfaces

RTE read/write

The RTE interfaces that are generated in the functional units implementation are derived from the standard RTE API reference in AUTOSAR RTE specification document [8].

According to this document the standard interface for explicitly reading a data element by argument on a sender/receiver port interface is:
Rte_Read_<p>_<o>([IN Rte_Instance <instance>] ,OUT <data>**)**
Where <p> is the port name and <o> is the "VariableDataProtoype". "VariableDataProtoype" is the data element short name. The OUT parameter is for receiving the data element through its corresponding argument.

Also the standard interface for explicitly writing a data element on sender/receiver port interface with "data" semantic is:
Rte_Write_<p>_<o>([IN Rte_Instance <instance>],IN <data>**)**
Where again <p> and <o> are representing port name and "VariableDataProtoype" respectively. The IN parameter passes the data element for writing on RTE.

It is worth to know that RTE generates the read/write interface only for those data elements which a corresponding "VaribaleAccess" is defined for them in the DRP/DSP [8].

RTE runnable

Definition of all runnable entities which are triggered by RTEEvent for their execution follows the same following form [8]:
<void|Std_ReturnType> **<name>([**IN Rte_Instance <instance>],[role parameters]**)**
Here <name> is the symbol of the runnable entry point. In case of having several runnable entities within an internal behavior, the runnable which the main timing event is defined on that will be the entry point.

B. Populus internal data scaling

The following table illustrates a subset of Populus data types. This subset contains the valid types for the data transmission in “AUTOSAR Integration” plugin.

Data Type	**Numeric Data Format**
Speed	Type 4
Integer	Type 1
EnumerationValue	Type 1
Decimal	Type 2
TextId	Type 1
BitmapId	Type 1
DistanceLong	Type 4
DistanceShort	Type 4
Volume	Type 4
Temperature	Type 4
Pressure	Type 4
FuelConsumption	Type 4
FuelConsumption Inverted	Type 4
DecimalShort	Type 3
FuelConsumption StandingStill	Type 4
FuelConsumptionCNG	Type 4
FuelConsumptionCNG Inverted	Type 4
FuelConsumptionCNG StandingStill	Type 4
FuelUsedCNG	Type 4

For increasing the resolution and also due to some other internal configurations each of these values are scaled before being saved in the HMI data base. The data values of type 4, type 3 and type 2 are divided by 1024, 10 and 1000 respectively before storing their values into the data base. The data values of type 1 will not be scaled.