Faculty of Computer Science, Department of Computer Engineering

in cooperation with

In Productline Embedded Software (PES1)

Master Thesis

In

Round-trip engineering concept for hierarchical UML models in AUTOSAR-based safety projects

For fulfillment of the academic degree
M.Sc. in Automotive Software Engineering

Submitted by: Charu Pathni

Matr.-Nr. 355078

12 August, 1991

Bangalore, India

Supervisors: Univ.- Prof. Dr. Wolfram Hardt

Dipl.-Ing. Markus Schwarz

Date of submission: 2 September, 2015

Charu Pathni

Round-trip engineering concept for hierarchical UML models in AUTOSAR-based safety projects

Master Thesis, Faculty of Computer Science, Department of Computer Engineering

Technische Universität Chemnitz, September 2015

Declaration

I hereby declare that this master thesis in topic “Round-trip engineering concept for hierarchical UML models in AUTOSAR-based safety projects”, is entirely the result of my own work and it has been written by me in its totality. Also, I certify that I elaborated this research independently. The work is based on foundation of the information sources and literatures used in the thesis that I have faithfully and properly cited.

Stuttgart, September 1, 2015	
Place, Date	Charu Pathni

Acknowledgement

The contentment and joy that follow the completion of any work is incomplete without expressing gratitude to all those people who helped you make it all possible. There are some special people without whose constant guidance and support, success may not have been achieved with same spirits.

I would like to express my deepest of regards to my professor Univ.-Prof. Dr. Wolfram Hardt and my advisor Dr. Ariane Heller. Their constant enlightenment on the subject and persistent motivation helped through each stage of my master thesis.

My profound thank you to the department of ProductLine Embedded Systems, Vector Informatik GmbH for the opportunity of master internship and master thesis in their teams which lead my way into captivating developments. My manager Mr.Hartmut Hörner, Senior Manager Software Development, Vector Informatik GmbH whose constant expertise on the subject was very valuable for my work during the thesis.

My heartfelt thank you to my mentor, Mr.Markus Schwarz, Senior Software Development Engineer who helped me steer my way through the entire master thesis and help me shape and design my thesis in a very productive manner. I would like to give earnest of regards to Mr.Timo Vanoni, Senior Software Development Engineer for his absolute patience at each stage and for his extremely helpful opinions that helped me look at things with a wider perspective.

Last but not the least, this work wouldn't have been accomplished without continual support from my family: for having my back at every step throughout my life. My friends, who continued helping me fix all my problems at any hour of the day, till the very end: Nitesh, Dhanush, Praveen, Arvind and each one involved at such an intricate level.

A genuine thank to everyone for helping me successfully complete my master thesis into a reality.

Place: Stuttgart Charu Pathni

Date: September 1, 2015

Abstract

Product development process begins at a very abstract level of understanding the requirements. The data needs to be passed on the next phase of development. This happens after every stage for further development and finally a product is made. This thesis deals with the data exchange process of software development process in specific. The problem lies in handling of data in terms of redundancy and versions of the data to be handled. Also, once data passed on to next stage, the ability to exchange it in reveres order is not existent in evident forms.

The results found during this thesis discusses the solutions for the problem by getting all the data at same level, in terms of its format. Having the concept ready, provides an opportunity to use this data based on our requirements. In this research, the problem of data consistency, data verification is dealt with. This data is used during the development and data merging from various sources.

The concept that is formulated can be expanded to a wide variety of applications with respect to development process. If the process involves exchange of data - scalability and generalization are the main foundation concepts that are contained within the concept.

Keywords - software engineering, data model, architecture, requirement, development, component, data exchange, verification, generic

Contents

List of Figures

List of Tables

Abbreviations

AL	Architecture Language
ALM	Application Lifecycle Management
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
CDK	Component Development Kit
Csh	C Sharp
CSS	Cascading Style Sheets
EA	Enterprise Architect
ECU	Electronic control units
E/E	Electronics and Electrical
HTML	HyperText Markup Language
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
OEM	Original equipment manufacturer
OMG	Object Management Group
Perl	Practical Extraction and Reporting Language
PHP	Hypertext Preprocessor
ReqIf	Requirement interchange format
RTE	Run time environment
SPICE	Software Process Improvement and Capability Determination
UC	Understand C
UDB	Understand C Database
UID	Unique identification number
UML	Unified Modeling Language

VCD Vector

VHDL VHSIC Hardware Description Language

V-Model Verification and Validation Model

VSF Vector specification file

XMI XML Metadata Interchange

XML Extensible Markup Language

XPath XML Path Language

XPointer XML Pointer language

XSLT Extensible Stylesheet Language Transformations

1 Introduction

The success of a product, be it construction of a building or producing a motor vehicle, depends on one of its most critical feature - the architecture. The planning involved in building a software follows the same lines of principle. Understanding the usage of any particular product evolves from the knowledge you acquire by understanding the requirements clearly. The following figure 1.1 depicts the basic idea of process of development in every industry.

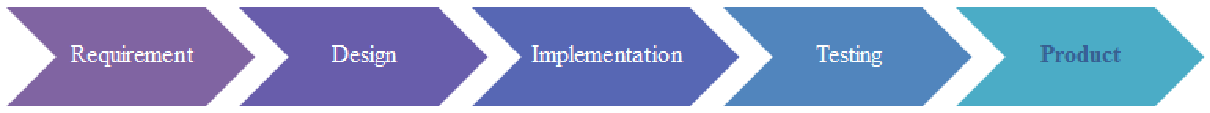


Figure 1.1: Process development - abstract view

Demands of complex requirements are seeing an exponential increase, Automotive Industry in particular. One of the main reasons for this growth is the rise in demand for safety features. And hence, interest was stirred to explore the roots for the same.

1.1 Motivation

While understanding the infrastructure of design and understanding the Automotive Open System Architecture (AUTOSAR) architecture, a particular term repeatedly occurred - Safety. Thus, captivating interest in the existing use of the concept. Automotive industry must comply with the safety standards in order to make sure that the driver is safe from any kind of accident, danger and risk due to the vehicle. This builds the pressure on manufacturers and tool vendors to make sure what they produce is not only safe for the driver but also complies with all the rules stated in the standard.

Vector Informatik GmbH provides the basic software components to the manufacturers while making sure hat the standards of the automotive industry are met.

There are various methodologies when it comes to software engineering. The complete procedure of development can take place in different phases. Some of the methods are:

- Waterfall
- Prototype model

- Incremental
- V-Model
- Spiral, and many more [Rpl13].

This research focusses on Verification and Validation Model (V-Model) development process. As the name suggests, it not only follows a top to bottom development but also goes upwards after the coding phase. The model can be represented as seen in following figure 1.2 of V-Model.

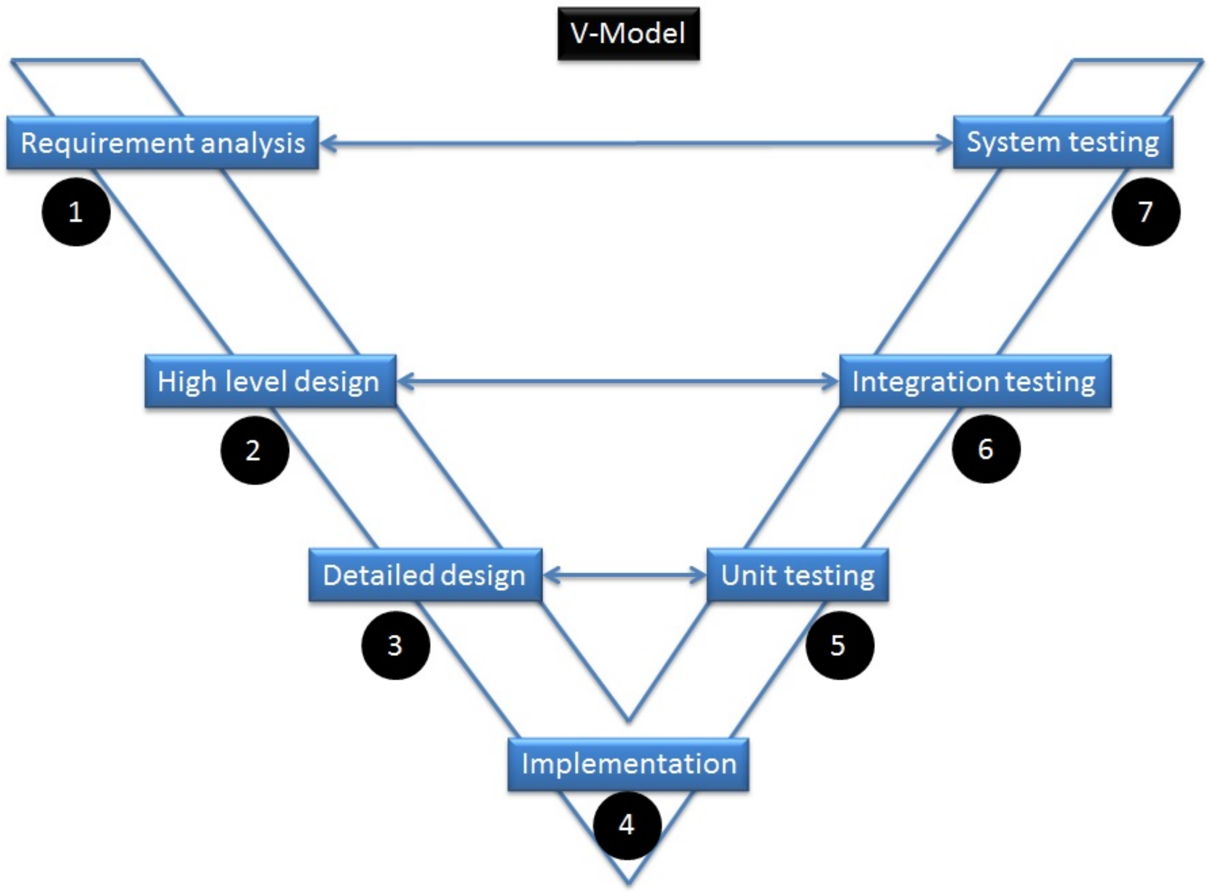


Figure 1.2: V-Model development process

V-Model stands for verification and validation model [Kum15]. As seen in figure 1.2, the first step to developing a product is analysing the requirement clearly. This phase is followed by high level design that is architecture design. In this, the complete product design is made which is further given as input to the next level of development, this is called product design. Then, the product is divided into smaller components and each one has a detailed design of its own, and this stage the architecture of each component is finalized. Finally, these components are implemented. Implementation is incomplete without the step of testing. Each unit/component is tested against the detailed design. Once it is sure that each component works independently as per the design, it is then integrated and tested against the product design/architecture design. And hence, the last step of this process is when the complete system is verified against the system requirements that were gathered at the beginning of the process.

This method is not only easy to understand, but fairly simple to use. The defects can be spotted early during the process and for small systems and it is really efficient to break down into various phases. For example, if there are problems that are detected at stage four of figure 1.2, the changes can be made at that level before moving it to next stage and also, updating this change to stages higher than it. Exchange of information is discussed at length in coming chapters and also, solutions to its limitations will be covered.

But during the course of internship, discoveries were made with respect to the obstacles in this simple process. The data that is exchanged from one level to another is in various formats and the usage at every phase is also variant. Throughout the development, the features of safety has to be maintained. Being one of the crucial points, data gained more significance amongst all other factors.

The focal point of this research is established pertaining to architecture design. This is a combination of the first four phases of the V-Model as seen in figure 1.2. All the developments are based on AUTOSAR platform and this makes it more interesting and complicated to comply with.

This instigated the thesis topic to be chosen and research on the same theme to its lowest register.

1.2 Problem statement

As discussed in the motivation section, architecture is the targeted topic of research.

Taking it another step closer to understanding, the architecture is modelled or designed using Unified Modeling Language (UML) and this is done with the help of a powerful environment called Enterprise Architect (EA). UML is a method used to design in software engineering and provides a standard visualization of the designed system [AW05].

Architecture of the product is thus designed in EA and contained in the EA file. This file is the design input when given to the component developers. The component developers use this file as a reference to understand the big-picture of the product. Although they mainly need the requirements of their own component, this design helps them understand the interaction between various components. This is important because in case of any change at component level, they will have an idea on how their change could affect the other components.

Apart from the use of reference of the enterprise architect file, the component developers use it to refine this design with the information of their concerned component before starting with implementation.

Due to the refinement process of data within these designs, there is a problem that arises - data redundancy. This issue not only arises at this stage, but as the V-Model progresses, it creates complication during verification and validation phases of the process too. To formalise the problem, few main questions will be answered and solved in this Master Thesis and they are as follows:

- What design information is needed by whom?
- How to work efficiently with data?
- What is the "Master" of data?
- How to comply with standards like ISO 26262 and SPICE process assessment?
- How to use this data during verification?
- How to do architecture "round-trip" data usage?

1.3 Thesis structure

The structure of this Master Thesis is organised in the following chapters:

- **Chapter 2** describes the literature research on architecture design in software perspective, requirements, safety standards, technologies and tools used. This chapter also explains the requirement standards, interchange formats, modelling framework, International Organization for Standardization (ISO) 26262 and Software Process Improvement and Capability Determination (SPICE) in particular. Under technologies and tools various platforms and methodologies are discussed.
- **Chapter 3** forms the core of this Master Thesis. This chapter throws light on the state of the art on existing problems and its causes. There were observed and collected by conducting a survey with colleagues of various teams at Vector Informatik GmbH . The possible solutions are discussed as well. Finally, it unfolds a concept and solution for a process that can ensure consistency and nearly redundancy-free design.
- **Chapter 4** outlays the solution provided in this Master Thesis to achieve the defined goal. It explains the concept and strategy to attain the objective of the research. Implementation phase 1 is discussed in this chapter. It covers the formation and usage of the final solution. Also, it discusses the implementation of the mechanism that was designed during the thesis. This involves the explanation of implementation of tooling that was completed.
- **Chapter 5** describes the results obtained by various implementations. Testing is carried out to make sure the extensibility of the solution and also its limitations.
- **Chapter 6** presents the conclusion to this work underlining the goal to be achieved in this Master Thesis. Also, it identifies some key enhancements to the existing solution and projects, the outlook for future development.

2 Literature Research

This chapter explains the the following topics comprehensively as part of the literature research:

- AUTOSAR
- Software architecture
- Requirements
- Standards
- Tools and technologies

These topics are the prerequisites to understanding the importance of safety and problems related to it. It gives a clear picture of the present scenario in all the domains that are mentioned above.

2.1 AUTOSAR

This section introduces AUTOSAR right from the most basic level of understanding. The main goal is to explain why this platform was introduced in the first place.

With the exponential growth of automotive industry and the massive increase in complexity of the vehicles, there was a need to organize this complex system. So, many founders of the leading automotive industry, came together to decide on a common platform. This platform promised the efficiency of increasing the level of complexity and still making it easier to handle. Based on many disadvantages of the existing development process, AUTOSAR was introduced to overcome all the problems. Global partnership of automotive manufacturers, suppliers of tools and software and all other involved like software, electronics and semi-conductor companies in this industry came together to agree and put together this platform for the common growth [Sys15].

2.1.1 Introduction

The complexity of the modern vehicles is increasing day by day. To manage the Electronic/-Electrical complexity of a vehicle was getting very difficult as there are more than a hundred Electronic control units (ECU) in today's cars. Also, the scalability factor becomes very hard to manage. If there are any modifications made to the existing system, the implementation of the change is very tedious. Therefore, to improve the quality and reliability, there was a need for change and introduction of a method that could make all the above mentioned problems really easy to solve and make the system more competent.

Goals - The main goals of AUTOSAR thus focused on making the development process easier. They did not just want to deal on a component level but on system level. Modularity was one of the aims, so that the system can be easily divided and worked upon individually by respective specialists. Now to transfer the characteristics of the system to much bigger system, scalability was important. So the next goal was to make it scalable. For individual systems, the property of transferability was required, so that it could be used to build next better systems, in lesser time. And finally, the goal of re-usability was covered. Some of the components of the system can be used for many years to come. So instead of investing time, to rebuild the basic components again, they worked on the aspect of re-usability.

Members - ([AUT15a]) The core members consist of BMW Group, PSA Peugeot Citroën, Continental, Robert Bosch, Daimler, Toyota, Ford, Volkswagen and General Motors.

Currently there are 48 premium members and some of them are TATA, Renault, Panasonic, Infineon, iAV, Elektrobit Corporation and many more. There are 100 associate members currently and some of them are Alps, Caterpillar, CISCO, HCL, Infosys, Nissan, Toshiba and many more.

Amongst the development members, there are 24 members at the moment and few are listed as follows, Vector Informatik GmbH, Basixworx, ESR Labs, Gliwa GmbH and more.

2.1.2 AUTOSAR Architecture

As seen in figure 2.1 [AUT15b], the architecture of AUTOSAR is divided into 4 main layers. On the topmost layer, the software components are seen. These are application software components that can be easily modularized and re-used. Then the AUTOSAR runtime environment (Run time environment (RTE)) is seen. This environment enables the applications to run on the platform. Below the RTE, the standardized interface that helps in the running of the components above is also noticed. These include the operating system, services, communication, ECU abstraction, Micro controller abstraction and Complex Device Drivers. These interfaces help in the functioning of the applications on top. All of these are placed above the main ECU Hardware.

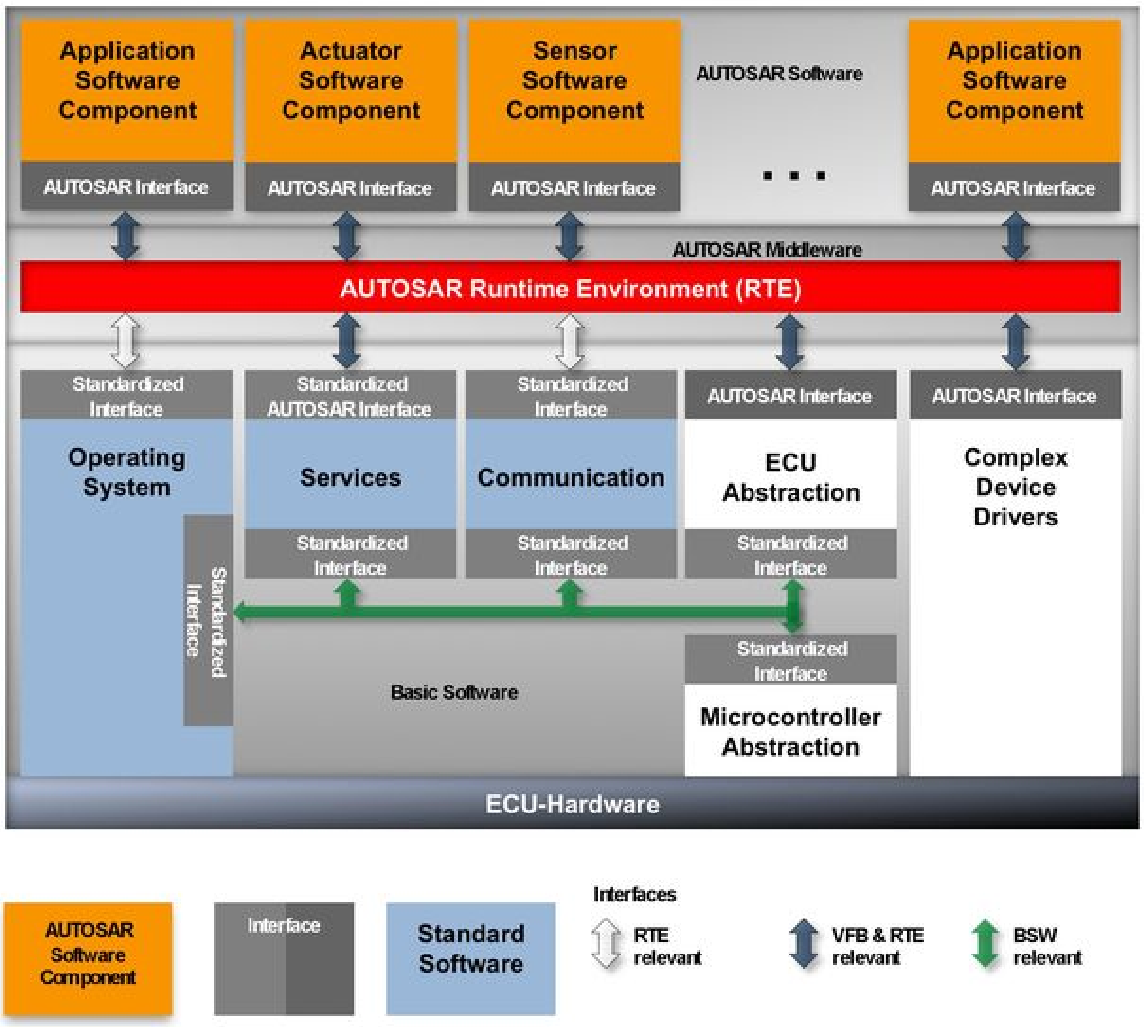


Figure 2.1: AUTOSAR architecture

Features - Based on the above architecture, features can be derived. Most important feature is its modularity and reconfigurability. Each function acts as a separate module and can be detached to configure respectively. This can be done only when there is a standardized interface. This is very important since there is a need for a common platform to work upon. And finally, to implement all the components and make sure the interfaces run in synchronized manner, we have the runtime environment which enables the complete concept of AUTOSAR.

Layered Software Architecture - The layered software architecture is the abstract view of where the application layer, RTE, Service Layer, ECU abstraction Layer, Micro controller Abstraction Layer Complex Drivers and Micro controller are clearly defined and given a boundary. It consists of different kinds of software components that are developed for various purposes and run on AUTOSAR RTE. In figure 2.2 [suc15], AUTOSAR layered software architecture can be seen.

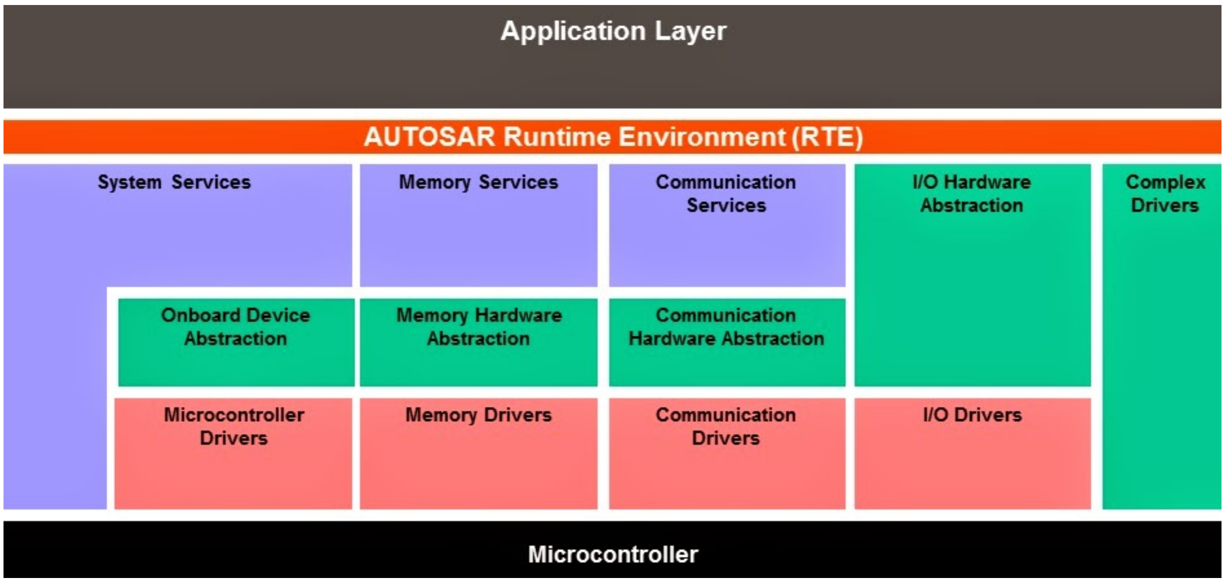


Figure 2.2: AUTOSAR layered software architecture

2.1.3 Application

Application Description - An application in AUTOSAR consists of interconnected "AUTOSAR Software components" which is also referred to as Atomic software Component. AUTOSAR software components are connected via connectors in an application. Each application has a description which contains set of fields that need to be covered for each component. For example, the data elements and operations to be used have to be specified. Requirements of the components based on the infrastructure of the system. The list of resources that will be used by the software components is needed. If there is any specific implementation by the component, that information has to be specified. All the mentioned fields can also be combined together and called as "software component template."

Application Development - Application development is discussed in detail in this section. The method of application development is divided broadly in 2 categories [Aut].
Firstly, there is Model-based development of application software component. This uses simulation method of development. Since, the behaviour of the model being built is verified in early stage, the final code generation will be of a very high quality. And the final output is an Extensible Markup Language (XML) file which contains software component description, as discussed in previous sections. And the code generated is in *.c, *.h format.

Second method is the "Classic way" of software development. This uses the development by hand, contrary to the method before. More effort is required to compare the results and meet the same quality. The output files are same, *.xml, *.c and *.h.

2.2 Software Architecture - Profound understanding

Assuming that a house has to be build, and it is decided not to hire an architect. Questions like 'how to get it build' without the knowledge of distributing the work amongst the builders? Moving to the possibility that it is built, is that house prone to any sort of threats or calamity?

That is exactly how crucial is architecture. When a house or car or a software is planned to be build, it all comes down to the plan of the product. To understand what is needed and what are the exact requirements of the user, is the key to begin the process of development.

2.2.1 Architecture - general outlook

Matthew R. McBride writes, "A software architect is a technically competent system-level thinker, guiding planned and efficient design processes to bring a system into existence. He is viewed by customers and developers alike as a technical expert. The architect is the author of the solution, accountable for its success or failure." [MR04]

Architecture definition can be branched into two major stages:
The first stage is breaking down the product that is being designed into smaller segments and defining its purpose or in technical terms - its functionality.
The second stage is defining the communication or interaction between these segments which is also a crucial part.
If the understating of communication within the product is clearly defined, it makes it easier for making refinements to the segments which will not affect the overall functionality of the product.

Hence, for the architect(s) to design any product, understanding of the requirements is absolutely essential. To understand this a little better, refer to the architecture business cycle that has been depicted in figure 2.3.

The influence of various factors while the architect designs the architecture of the system, is noticed in figure 2.3.

To sum it up in one single statement - 'Architecture is design but not all design is architectural' [Cle+10].

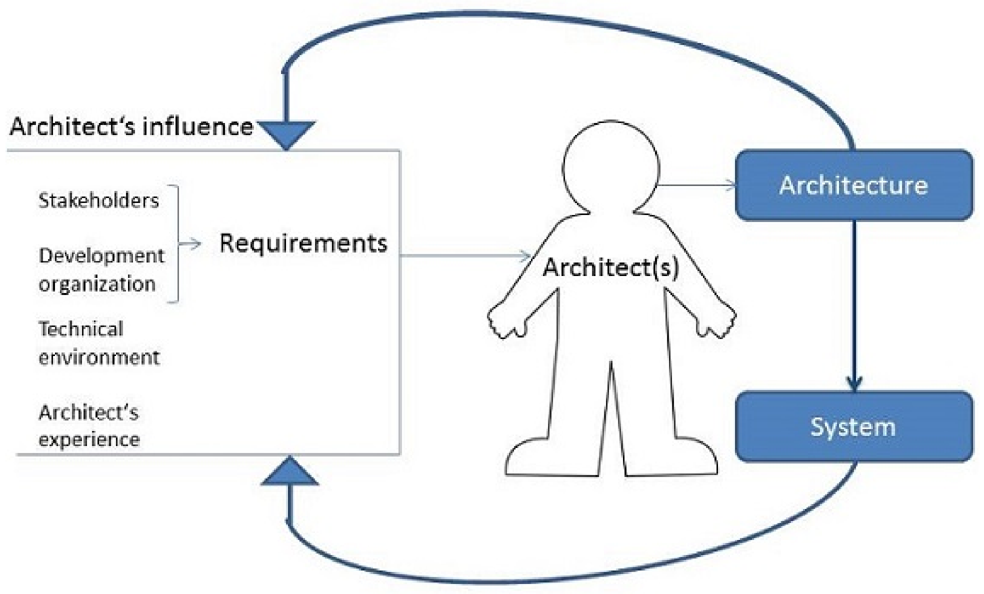


Figure 2.3: Architecture Business Cycle

2.2.2 UML and Views

Architecture has its own design/modeling/description language. Around the world regularly used language is UML. It helps the architect to describe the design in visual form using components like modules, linkers, activities, interfaces and show the communication and interaction between various components.

There are a variety of diagrams that are designed in UML. The figure 2.4 represents the classification of the various types [Gro]:

A view is the composition of various components of the system and also the relationships between them. UML helps in modelling 2 different views of the system [Uml]:

- Static view
- Dynamic view

These models can be exchanged in a format called XML Metadata Interchange (XMI). This format will be discussed in section 2.5.

Different kinds of views are listed as follows [Cle+10]:

- Functional/logic view
- Code/module view
- Development/structural view
- Concurrency/process/runtime/thread view

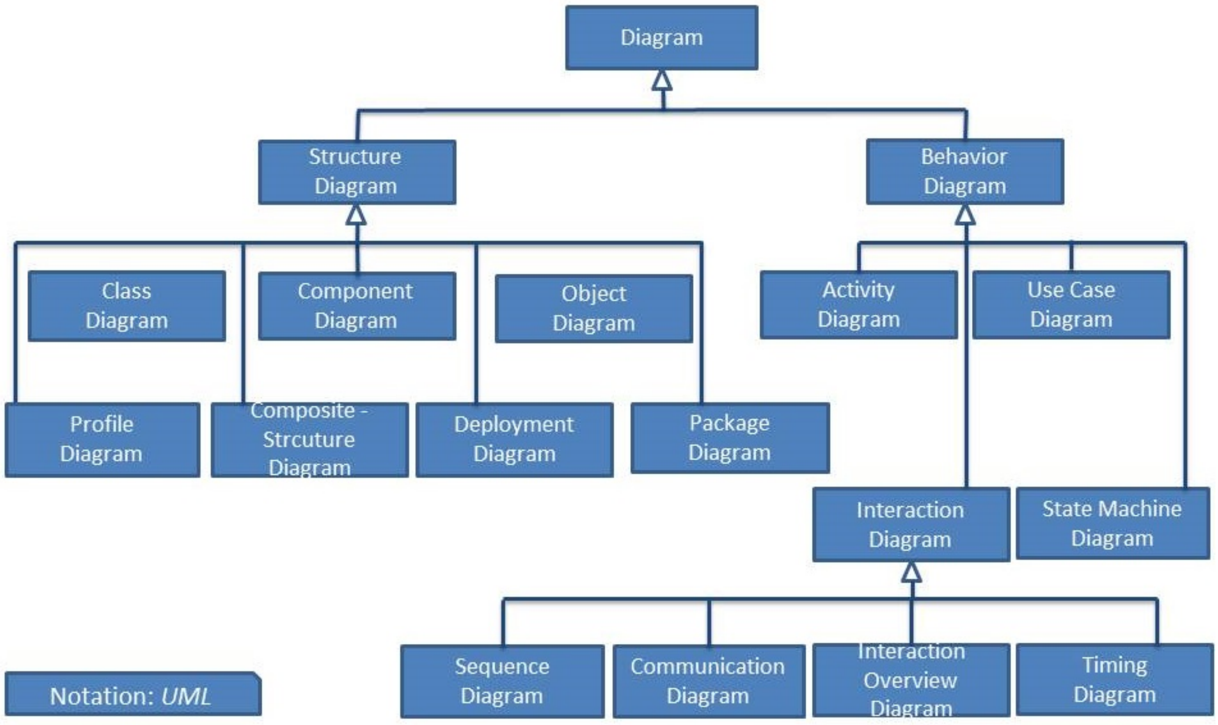


Figure 2.4: Classification of UML diagrams

- Physical/deployment/install view
- User action/feedback view
- Data view/data model

There are several architecture frameworks available along with many other architecture description languages. Some of the frameworks include Reference Model of Open Distributed Processing (RM-ODP) and the Service-Oriented Modeling Framework (SOMF) [Rpl13]. Amongst the various Architecture Language (AL) some of them are ABACUS (UTS), ACME / ADML (CMU/USC), and many more [Mal+15].

After the design is complete, it is not only easy to understand the product, but also goes both way in hierarchy. It becomes really efficient to discuss the design with the stakeholders and once it is finalized, it is easy to give architecture design as an input to the component level for further refinement. The component is defined within the architecture design and then the detailed design is used for implementation.

2.3 Requirements

In day to day activities, how are the needs generally expressed? While ordering food at a restaurant, or deciding which car to buy, or what kind of clothes to buy for a certain occasion or building a house? Decisions are made based on the need of the situations. These needs are technically called requirements.

Figure 2.5: Requirement(Image source: [Pix15])

In software engineering, while developing a product, requirements is a formal definition of all the functions and services it needs to fulfil. So, while designing the product, at the architecture level - it is the document that is used as the guiding path. Requirements are not only used in the beginning for reference, but also at the verification stages to verify at each stage of development. This keeps the product that is being built, well within the scope of its need.

Requirements are branched into 2 fields [CDC15]:

- **Product Requirement** - This defines the overall service provided by the product.
- **Process Requirement** - This definition mainly implies to the organization that is building the product. It defines the methods and constraints to be maintained during its development.

Depending on the phase of development, requirements metamorphose into different types. They are listed as follows [Int15]:

- State/Mode - Specification of number of states and the progression from one to another should be clear
- Functional - This requirement must clearly state the task of the system/product
- Performance - The extent to which the product can be used and push the limits is also defined
- External Interface - This specifies if the system/product can be used in other places/countries
- Environmental - This describes how the system can run and where does it store its sources and so on

- Resource - The amount of consumption by the system/product must be clearly mentioned
- Physical - If we are talking about a system apart from a software, this requirement specifies its physical properties
- Other quality - This reflects any other requirement which is specifically mentioned apart from the above mentioned
- Design - This requirement discuses about the internal and external design of the system/product in detail

Requirements need to follow some rules as well. Since it plays an important role to provide intricate details of the system, it should be:

- Easy to understand
- Granular in nature
- Thoroughly descriptive
- Rationally defined
- Testable at every stage

2.3.1 Requirement Interchange Format

Now that it is clear how essential requirement is, understanding how it can be used and exchanged is also very important. That's where there is a concept of an interchange format of requirements. This was initiated by an international group called the 'Object Management Group (OMG)'. They develop standards that can be used at an international level for diverse technologies and industries.

But before OMG took charge of this, it was initially introduced by a group of German automotive groups and they had named it 'RIF' [Boa15].

After which,Requirement interchange format (ReqIf) was brought into existence by OMG group. This idea was thought of when the exchange of requirements was lossy in nature and that caused a lot of problems during and after the product was developed. When one party(stakeholder) had to exchange ideas, product and requirements to another party(developer/stakeholder), it was soon realized that if the requirements were not exchanged seriously, it caused losses for the stakeholders as well.

ReqIf is a standardized meta-model, in XML schema [Req]. It not only contains information of the product, but also other details like the version, the ID, stakeholder details and so on. This concept is depicted in the figure 2.6:

The reason that it plays an important role in this thesis is due to its benefits. It helps in understanding the significance of the requirement document and also a possible format that can be used while considering the data problems being dealt with.

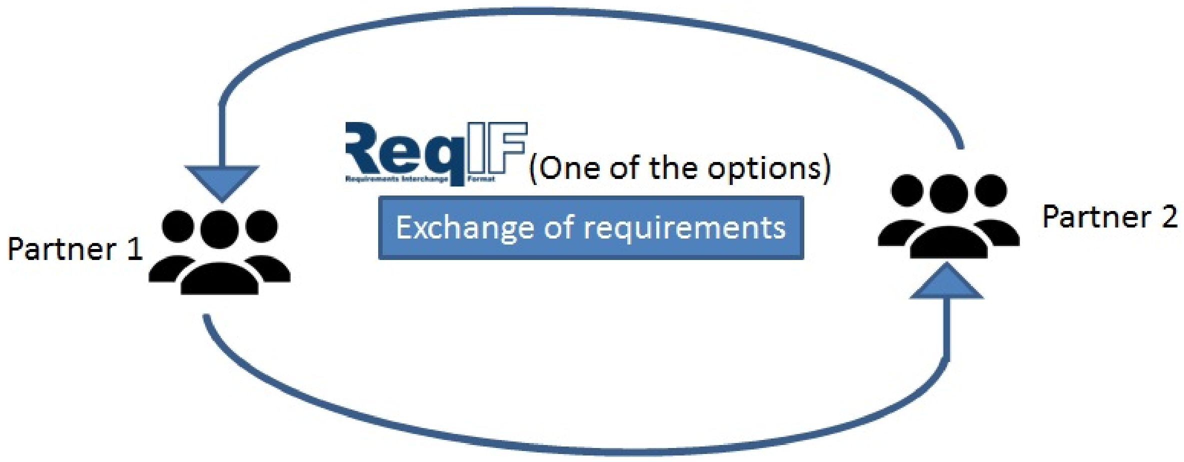


Figure 2.6: Concept depiction - Requirement Interchange Format

The exchange process between 2 partners is shown in figure 2.7:

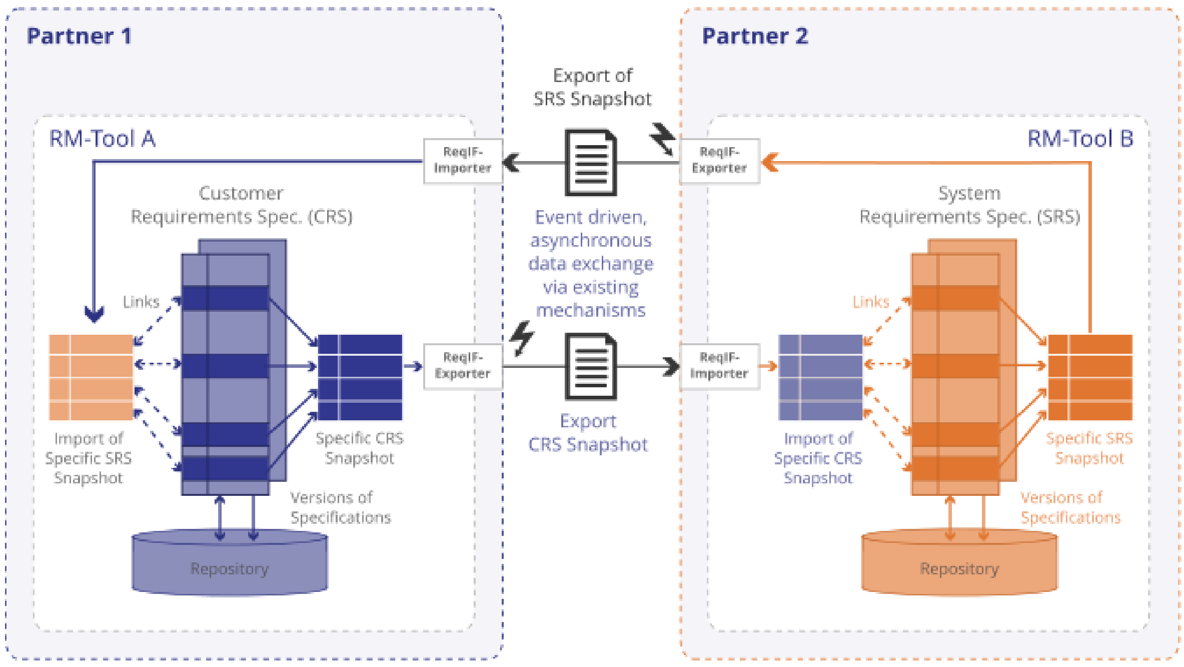

Partner 1
RM-Tool A
Customer Requirements Spec. (CRS)
Links
Import of Specific SRS Snapshot
Specific CRS Snapshot
Versions of Specifications
Repository
ReqIF-Importer
ReqIF-Exporter
Export of SRS Snapshot
Event driven, asynchronous data exchange via existing mechanisms
Export CRS Snapshot
Partner 2
RM-Tool B
System Requirements Spec. (SRS)
ReqIF-Exporter
ReqIF-Importer
Links
Import of Specific CRS Snapshot
Specific SRS Snapshot
Versions of Specifications
Repository

Figure 2.7: Exchange between partners - ReqIf (Image source: [Boa15])

In the figure 2.7 shows that partner 1 wants to get feedback on a requirement specification from partner 2, but only for a certain part of the requirements and sending complete set of requirements would not be necessary. Therefore, only a subset of the requirement specification is sent (depicted in dark blue colour) via e-mail or any direct file sharing portal. Partner 2 already has a set of requirements with them and they import the file into their system. After refinements of the requirements, and once it is finalized, it exports the file back to the Partner 1. Partner 1 then imports it back to its database [Boa15].

This is a fine example of how ReqIf makes the exchange of data(requirements) so smooth and uncomplicated.

The file extension is *.reqif and the underlying format of model is in XML to make the processing easier.

2.3.2 Requirement Modeling Framework

In order to make changes to the ReqIf file and to work with it, an environment is needed that supports this process. To ensure this, a framework is needed and hence the name, requirement modeling framework.

The framework is eclipse based open-source software. This works with the needs based on ReqIf.

The functionality of this framework includes storing, writing, validating and manipulating ReqIf files [Ecl15]. Adding more quality into the product, it can also coordinate with other Eclipse tools. This feature expands the framework a lot more and sophisticated to use. Developments continue to make it more dominant and presently prototypes are being released to note the response from the industries.

2.4 Standards

According to Merriam-Webster, safety is defined as "the condition of being free from harm or risk," and security is defined as "the quality or state of being free from danger" [EBC15]. These feature are used in every industry and here the reasons for it being so essential from automotive perspective as well are discussed.

The following figure 2.8 [car15] shows a disassembled 'Mk2 Golf' that is manufactured by Volkswagen.

The figure 2.8 clearly depicts the complexity of a car. This complexity is exponentially rising as the day passes and that makes it a responsibility of the manufacturers in the automotive industry to make sure of one of the very important aspects - SAFETY.

This had to be implemented by introducing standards at an international level and had to be followed in the automotive industry in order to get necessary permissions before release of any vehicle.

Two such standards that are focused in this master thesis. They are:

- ISO 26262 standard

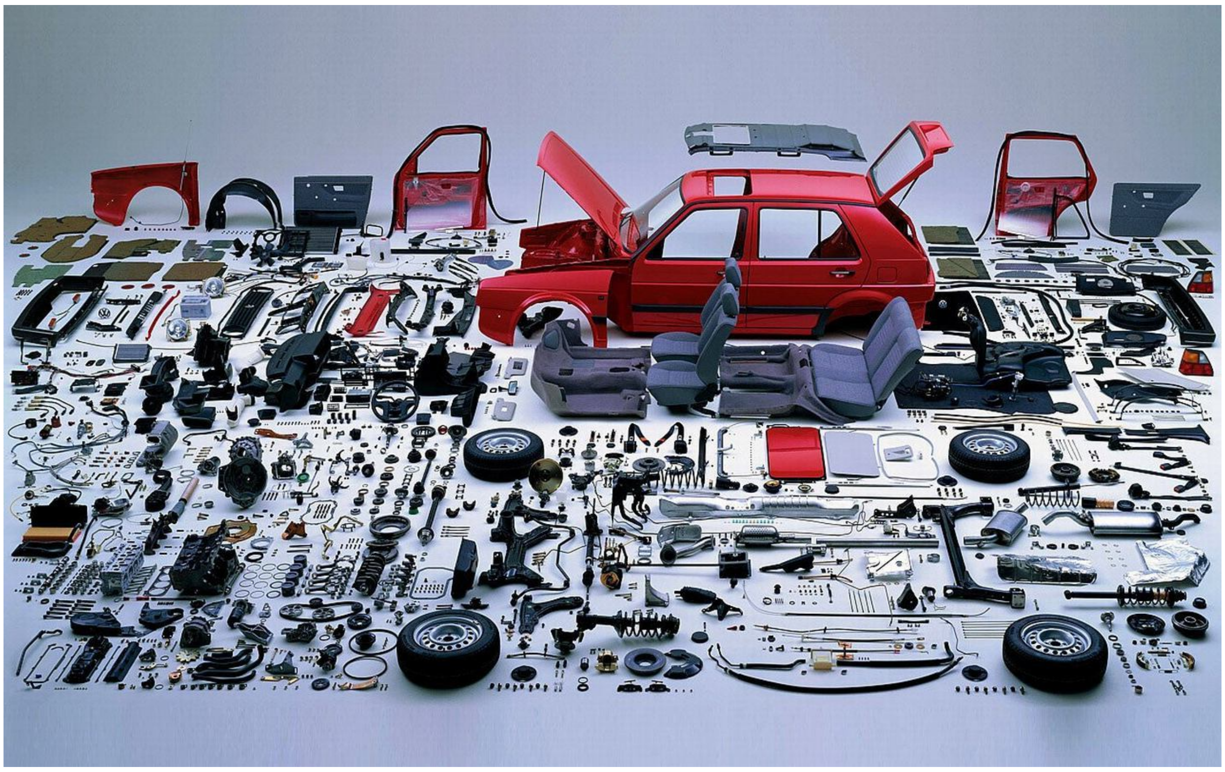


Figure 2.8: Disassembled Mk2 Golf (Image source: [car15])

- SPICE

2.4.1 ISO 26262 Standard

In order to make sure they suffice the requirement, it was decided by the ISO to formulate and release a standard for safety of the automotive Electronics and Electrical (E/E) system. The earlier release of the standard was International Electrotechnical Commission (IEC) 61508. The standard discussed here is ISO 26262 which is a derivative of the IEC 61508. [Ins15].

ISO 26262 defines safety aspects in terms of hardware, software and overall at the system level. The following topics are the general overview of what the ISO 26262 Safety standards covers [Iso]:

- Vocabulary
- Management of safety functions
- Concept phase
- Product Development at system level
- Product Development at hardware level

- Production and operation
- Supporting Process
- Automotive Safety Integrity Level (ASIL) and safety oriented analyses
- Guidance on ISO 26262

The main focus in this thesis, amongst all the topics listed above is **Product development at software level**. This can be summed up under the following topics:

- It provides a safety cycle about all the critical points to be considered during the development
- It also provides various classes to classify different risk levels - Automotive Safety Integrity Levels
- Further more, it uses these levels to determine if the system is acceptable or not
- Finally, it states the requirements for validation that all the standards have been met

Product Development at system level, the module was formulated to make sure safety was at product development at software level. It begins from the requirements understanding at product level. Understanding of requirements from safety point of view is a step before designing begins. This is followed by designing the architecture of the complete product. Then it is itemized into separate components and detail design for each is made. Each unit/component is developed and tested as it is developed. Once all the units are ready, they are combined together to see the functioning of the product as a whole. This must pass the tests at product level and one of the most important requirement is that final product must verifies against the initial requirements given. The product verifies successfully against the initial requirement that was gathered. The safety requirements have to be met once the product is complete and only then it passes this module of the standard.

This makes sure that safety involves at every sub-phase of the development. The depiction of the concept is in the following figure 2.9.

In the figure 2.9 we notice the steps involved at a detailed level. The system design is the abstract overview of the first phase in the V-Model represented. It refers to the tables in ISO part 6. That leads to the architecture of the software. This architecture design is implemented at component level. After the completion of development at component level, it is then tested unit wise. When unit testing is completed, the components are brought together to form the software which is tested after integration against the architecture design. Finally, when we obtain the integrated software, we need to verify against the safety requirements that is mentioned right in the beginning. This process completes the module at focus. After which the software is integrated with rest of the system and tested against the system design.

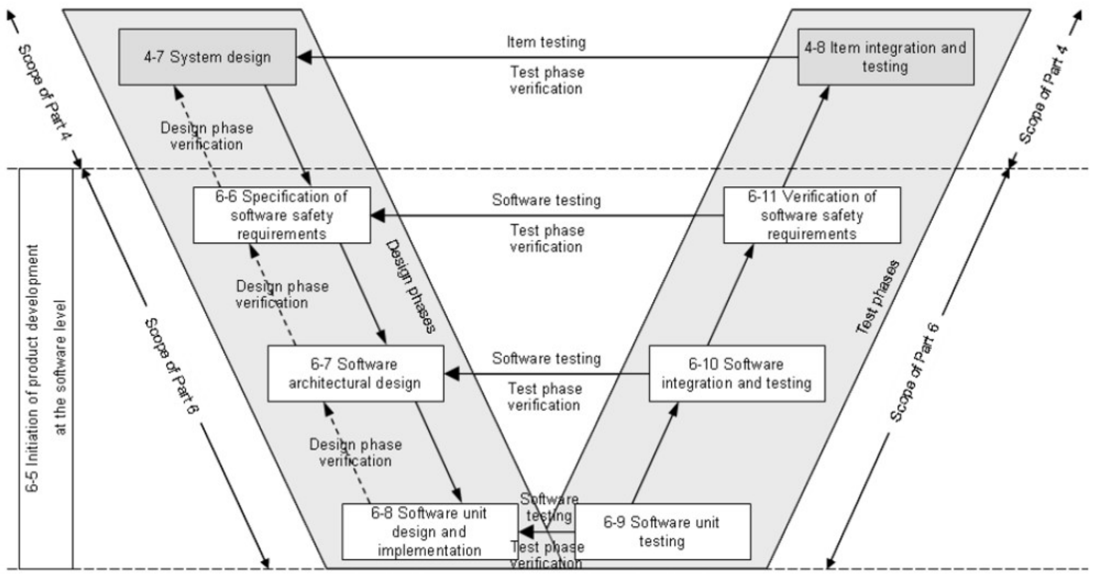

4-7 System design
Item testing
Test phase verification
4-8 Item integration and testing
Scope of Part 4
Design phase verification
6-6 Specification of software safety requirements
Software testing
Test phase verification
6-11 Verification of software safety requirements
Design phase verification
Design phases
Test phases
6-5 Initiation of product development at the software level
Scope of Part 6
6-7 Software architectural design
Software testing
Test phase verification
6-10 Software integration and testing
Design phase verification
6-8 Software unit design and implementation
Software testing
Test phase verification
6-9 Software unit testing

Figure 2.9: Reference phase model for software development (Image source: [Inc15])

2.4.2 SPICE

SPICE is a major international initiative to support the development of an International Standard for Software Process Assessment [Pyh04]. In automotive industry, the software involvement is increasing as the years are passing. It has been found that if the process that is involved in the development could be assessed then the capability of the process can be determined in an easier way along with improvement of the process if needed.
SPICE is based on the standard ISO/IEC 15504. In order to understand the process assessment by SPICE, observe figure 2.10 [Pyh04].

The figure 2.10 represents the process assessment at a high level. Here the focus is on evaluating one process at a time. Using SPICE standards, the process undergoes the assessment. This results in 2 major outcomes - Process improvement and Capability determination. Process improvement leads to identifying the changes needed in the process. Whereas capability determination identifies capability of the process and the risk involved in the process. If there are risks involved, it motivates for process improvement as well. This entire practice not only helps in understanding the process but also detecting risks at an early stage. This identification further helps in reducing the costs involved in improvements at later stages, as compared to cost of improvement in early stage.

Processes are organized into 5 process categories in the model [Pyh04]:

- **Customer-Supplier** - The processes that directly affect the customer and operation of the process.

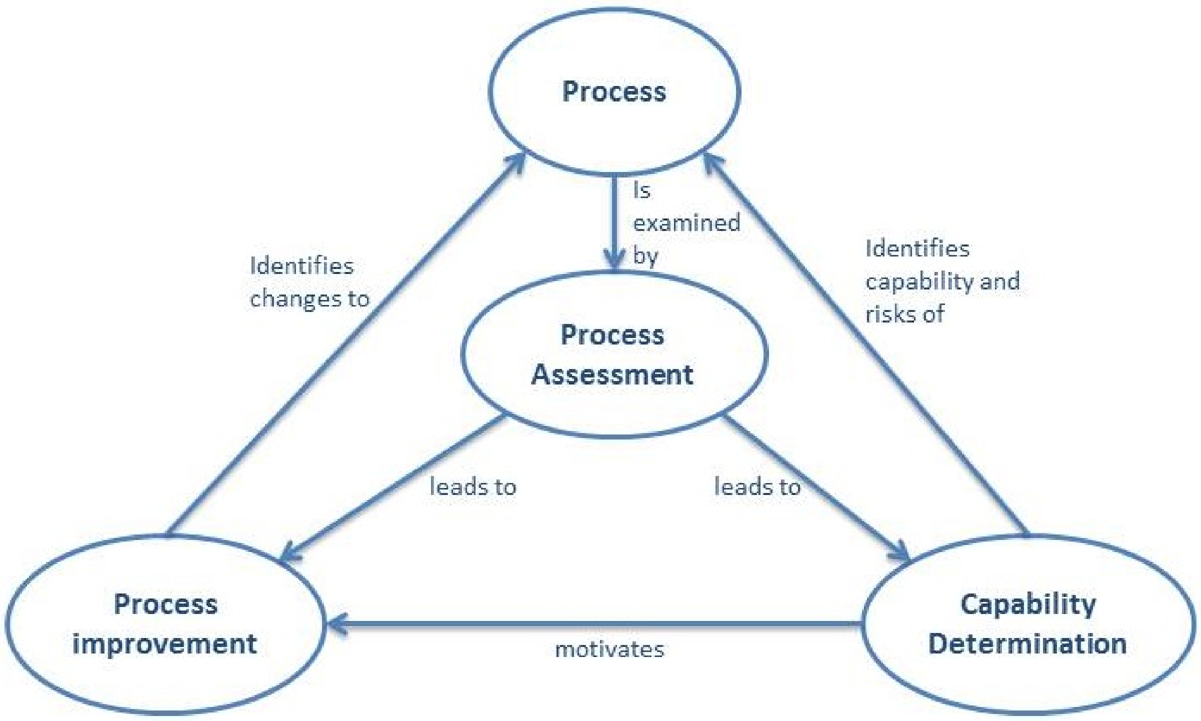


Figure 2.10: Process assessment on high level

- **Engineering** - The process that is directly associated with the system software and product and its documentation.
- **Project** - This covers the processes that affect the complete project and deals with the resources and its distribution through the process.
- **Support** - The processes that support other processes on a project.
- **Organization** - The processes that discuss the goals of an organization, and help it to achieve them.

As per software design, there is a specific set of requirements specification [SIG10]. The purpose of this process is to design in such a way that after implementation, it can be checked against the initial software requirements. When this process is used correctly, the following advantages are presented [SIG10]:

1. A software architectural design is defined that identifies the components of the software and meets the defined software requirements
2. The software requirements are allocated to the elements of the software
3. Internal and external interfaces of each software component are defined
4. The dynamic behaviour and resource consumption objectives of the software components are defined

5. A detailed design is developed that describes software units that can be implemented and tested
6. Consistency and bilateral traceability are established between software requirements and software architectural design
7. Consistency and bilateral traceability are established between software architectural design and software detailed design

The above mentioned advantages have been aimed for throughout this thesis.

2.5 Tools and Technologies

This section discusses various tools and technologies are discussed that were studied and surveyed which might be useful for the implementation of the concept. Each tool and technology is understood from the point of present research work and how can it be used. The explanation to each usage is covered under their respective topic.

2.5.1 Tools

1. **Enterprise Architect** - As mentioned before, one of the crucial steps in system development is understanding the requirements clearly. In order to visualize the idea or concept, we need a tool that can help us model our ideas. One such tool is Enterprise Architect. EA is a visual modelling and design tool based on OMG UML. It has a range of open industry standards for designing and modelling software and business systems [Spa98]. Some of the features include:

    - Requirements Managements
    - Business Modelling and Analysis
    - Simulation
    - System Development
    - Test Management
    - Visual execution analysis
    - System engineering and so on

    Apart from the mentioned points - performance, speed, traceability, complexity management, document generation and effective project management are amongst its powerful features.

2. **Understand C** - Understand C (UC) is a source code analysis and metrics tool. Its design can maintain huge amount of newly created source code. It provides cross-platform, multi-language, maintenance-oriented interactive development environment. The source code analysis may include C, C++, C#, Objective C/C++, Ada, JAVA, Pascal, COBOL, JOVIAL, VHSIC Hardware Description Language (VHDL), Fortan, PL/M, Python, PHP, HyperText Markup Language (HTML), CSS, JavaScript, and XML [Und]. Some of its powerful features are listed as follows:
   - Architecture features that help us create hierarchical aggregations of source code units
   - Provides a tool called CodeCheck to make sure code conforms to published coding standards or our own custom standards
   - For all languages, checks are provided to verify that various entity types confirm to naming conventions and to conrm that code meets metric requirements that were set for complexity, function length, and the nesting depth
   - Understand C can be extended for static checks using Perl
3. **CDK** - Component Development Kit (CDK) is an internal tool developed at Vector Informatik GmbH. It is used for parsing, static testing and analysing code. This has various options of testing and representing reports in form of HTML. This will be used during thesis to integrate the scripts and run with the database.

2.5.2 Technologies

In this section, various technologies are described in brief. Each one has its own purpose of understanding and will be used in different phases of the thesis.

1. **UML** - As discussed in section 2.2.2, UML is a general purpose modeling language which is used in software development process and it is used to describe the software/product in graphical representation. Details of the system can be described in terms of objects and its properties. It also allows you to clearly define the relationship between each of the objects. Thus, making it clear to understand for further implementation.
2. **C#** - C Sharp (Csh) is an object oriented programming language developed by Microsoft within its .Net initiative. The features of Csh are as follows [Csh]:
   - Portability
   - Typing
   - Meta Programming
   - Methods and functions
   - Namespace

- Functional programming

3. **Perl** - Practical Extraction and Reporting Language (Perl) is a high level, general purpose, interpreted, dynamic programming language [www15b]. It stands for 'Practical extraction and report language' [www15a]. Its powerful text extracting features are a result of combination features from various other languages like C, Shell scripting, awk. It is overall procedural in nature. This scripting language will be helpful during thesis for extracting data from certain databases and use them further for processing in different forms.
4. **XML** - XML is language for encoding data in such a way that it is human readable and machine readable format. The usability of this language is very high due to its simplicity of defining rules as per the requirements. Since its usability is across internet, it can be represented in the form of HTML document by transforming the data using XSLT (which is discussed in next section).
5. **XSLT** - Extensible Stylesheet Language Transformations (XSLT) is a transforming language that helps us represent the encoded data from XML format to HTML or text or just simple objects (readable and presentable) format. This can be further used to convert data into PDF as well. It can take more than one XML file as input to process the data. It can be specified with any computation and it will be performed by the computer, thus making it a Turing-complete language.
6. **HTML** - HTML is a mark-up language that is used to create web pages. There are defined tags that are used while writing a HTML file. But it is mainly used for representation, since the tags are not shown on the screen. It can be opened by any web browser. Data is interpreted and displayed on the page as we would like it. The page can be made interactive or just a read-only page depending on the requirement. In this thesis, it would be used for representation purpose.

For the representation of how some of the formats would be used, the following figure 2.11 is used as an example.

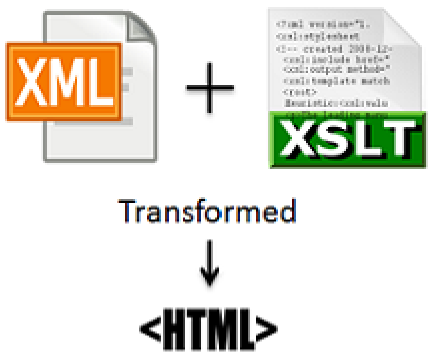


Figure 2.11: Depiction of usage of XML and XSLT technologies

3 Conceptualization

This chapter covers all the phases of conceptualization. In order to come up with a concept the following steps were taken:

- Existing Concept
- Survey
- New Concept
- Meta model formation - from different point of views
- Format analysis - of formats like UDB, VDC, ReqIf
- Aspects under consideration
- Intermediate format conceptualized

3.1 Concept

This section discusses about the concept completely. The presently used concept, problems detected and finally discusses the new concept. The concept was brought together by the survey, results and understanding the parts involved of the present scenario. The consolidated form of the present concept is seen in the end of this section.

3.1.1 Existing concept

In the figure 3.1 an overview of the present concept is seen. The two users are architect(s) and the component developer(s). The interaction between them and the data, along with the exchange of the data is shown clearly. Also, the formats are specified.

As seen in 3.1, the objects in blue colour code, are the data from various sources. The product architecture design, component architecture design and the source code itself are available.
All the objects noticed in yellow colour are the various formats that are used. In existing concept: .vcd format, .xmi format and .xml formats are used. Clearly, there is no one common file that can be used to exchange data. That is on of the main concerns that is dealt with during this thesis. One more important point to be noticed is the round-trip engineering concept is not covered within this existing concept. That has been taken care of within the research work.

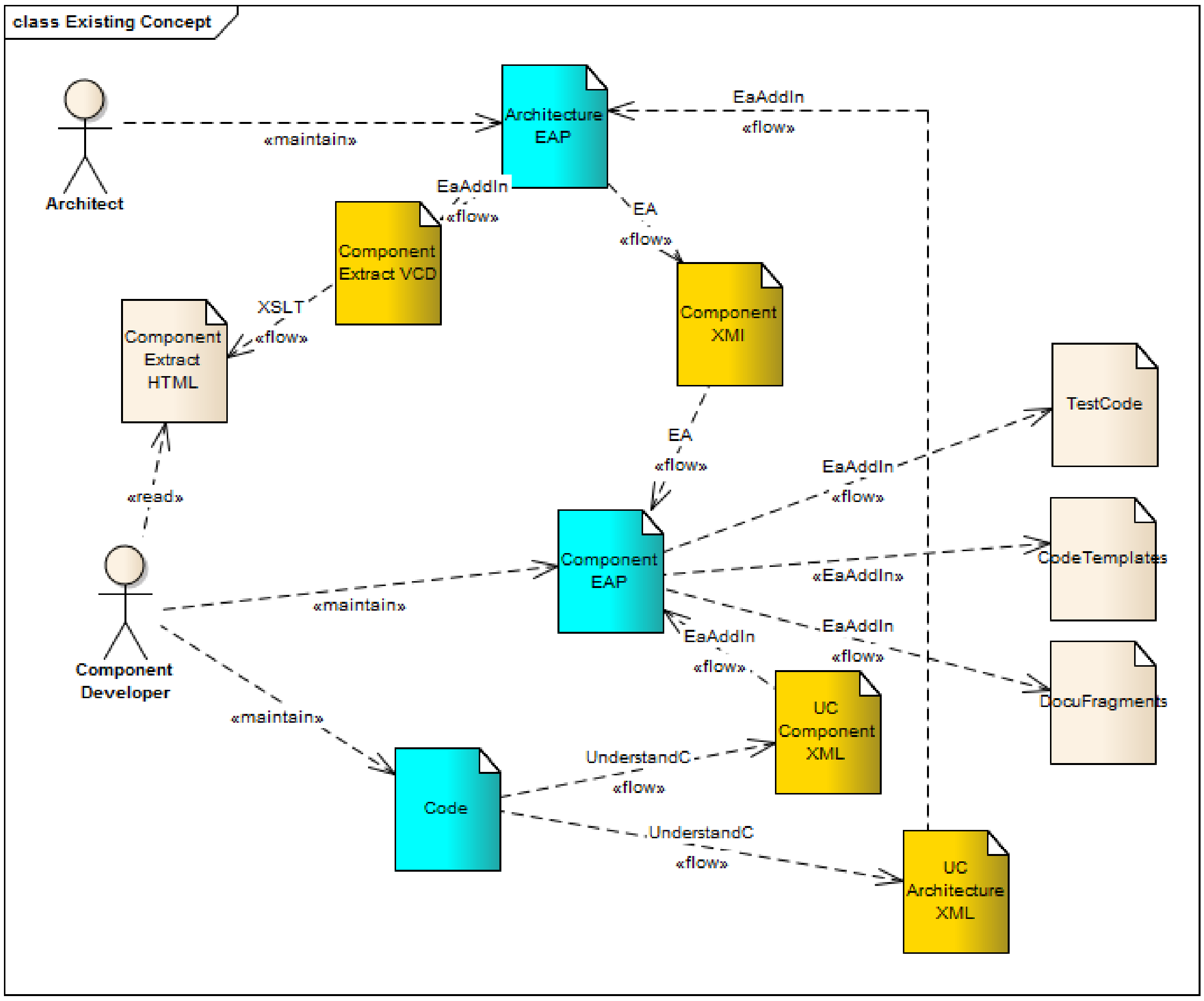


Figure 3.1: Existing concept - an overview

3.1.2 Survey

The first part discusses the reason of the survey and the questions asked. After which analysis of results of the survey is discussed. Finally how results of the survey helped in detecting the problems is shown.

These questions are directed mainly to gather input with respect to three areas.

1. Data - What data comes from where and what is actually required.
2. Format - What are the present formats and what could be improved based on what is needed.
3. Process - How is the exchange happening and if that is efficient to take care of updating data or verification process.

The questions that were asked were mainly concentrated in this spectrum. What data is more interesting data that is required by the developers from the architecture point of view? How was the data presently presented and what could be the changes they are looking for? In case of changes at component level, how is it informed to the architecture level? Followed by how is the developed component verified against the architecture? Their perspective on the present system can be improved and finally was the environment of EA comfortable for them.

- **Questions targeted -**

  The band of the questions asked varied from architecture requirements data format to verification of code against requirements after the implementation. This survey was collected based on the interview of colleagues from component development teams. Since they use the product architecture design as a reference, it was better to ask people directly involved with the present process. The list of questions were as follows:

  - What do you find interesting from Architecture point of view?
  - What information is needed? Why is it needed ?
  - What information is not needed?
  - How would you like the architecture design to be presented?
  - How would you want to use the information? (formal linkage/direct usage?)
  - For what do you use the information?
  - How do you communicate from component development to architecture in case of changes?
  - What would you provide in case of changes? (Documentation/changes/*.c file)?
  - How do you verify architecture design with component?
  - How would you like to do the verifications?
  - Any improvement aspects concerning work with architecture? (Req/XML/?)
  - Are you comfortable with enterprise architect environment?

- **Results of the survey and problems detected -**

  In this section, the consolidated results of the survey that was conducted are presented. Responses of each person was noted and added to the table. These responses were then combined together for further analysis. The final consolidated results include the majority common answers along with certain unique responses which can be further discussed in thesis. The merged results can be seen in the table 3.1. As seen in 3.1, it was found that the information that is needed by the developers was very specific. As compared to present system, they expressed that excess of information is not required for them. They also told that it would be better if the product architecture design would be more helpful and easy to understand if it was in a more readable format, with the information

Merged results of survey	
Questions	Overall Opinion
Interesting from Architecture point of view	Interaction between modules, Diagrams (class,sequence)
Information needed	Interaction between components, modules, API's
Information not needed	Too detailed (eg Parameter list, features)
Architecture design to be presented in which format	EA is okay but oversized. HTML or any form of readable format is better.
Using information	Mainly reference and understanding purpose
Communication in case of changes	Personal communication or feature manager (Basically direct)
Future prospect for communication	Either documentation form, or automated
Verification against architecture design	Not done and maybe sometimes manually reviewed
Future prospect for verification	Most opinions refer to .c against .ea verification
Improvements possible	Communication improvement (includes notification in case of update, documentation to raise interest) and important point is 'Runtime behaviour and interrupt locking' must be considered
Comfort level of EA environment	Yes but complex and may not be in favour of majority of other team members

Table 3.1: Consolidated results of the survey

that is only relevant for them. In case of changes from the component development level, presently, the information is given only via direct communication or e-mail. If there would be a more automatic tooling for this, it would be much more efficient and up to date. When it came to verification, it was realized that most of the verification is done manually.

The fact that need of communication had to be improved was also established.

Taking all the results into consideration, problems were clearly detected. They can be consolidated as follows:

- Too detailed
- EA is oversized environment
- Verification done manually and sometimes not at all
- Runtime behaviour

3.1.3 New concept

In this section, an new concept is formalised. This is based on the survey conducted and the idea of covering round-trip engineering theme. This concept gives a wide view of the possible solution. The overview of new concept can be seen as follows in 3.2:

As seen in 3.2, exchange of data and interaction between the users and data, is much simpler than what can be seen in 3.1.
Keeping the colour code same, the blue colour objects in the figure, represent the data sources. The new thing here is, all the data sources are interacting based on one common file - that is the new file format that was designed during this thesis. It's name is VSF file format. It stands for "Vector Specification File". This is seen as the yellow colour object in the figure.
The dotted lines that are interacting with the product architecture design, named as 'Architecture EAP', it can be noticed that the direction of data flow is both way. Hence, covering the theme of round-trip engineering concept.

Further, once the file format is conceptualized and detailed, it can be used for other smaller tooling purposes. The intermediate file can be used to generate C template from the architecture input. This can be also given to the developers which contains the definition of certain data that is mandatory for them to use, for example the functions. Next, the file can be used to generate user document. This document would contain all the details of product design and maybe also the data from source code, that might be useful. Another useful tooling could be generation of test scripts from this file. These test scripts could be used for static checks of the component developed. And finally, test cases can also be generated which gives a baseline of what the component is expected to do, what is available and what is the expected results. This helps in verification against specific requirements.

The above described concept not only covers the round-trip engineering model, but also explores the possibilities of tool-extensions. But, before moving on to finding ways of implementation, it is important to define meta-models from both architecture and source code. Without having a clear understanding of the different data having different sources, it is not possible to conceptualize the file format.

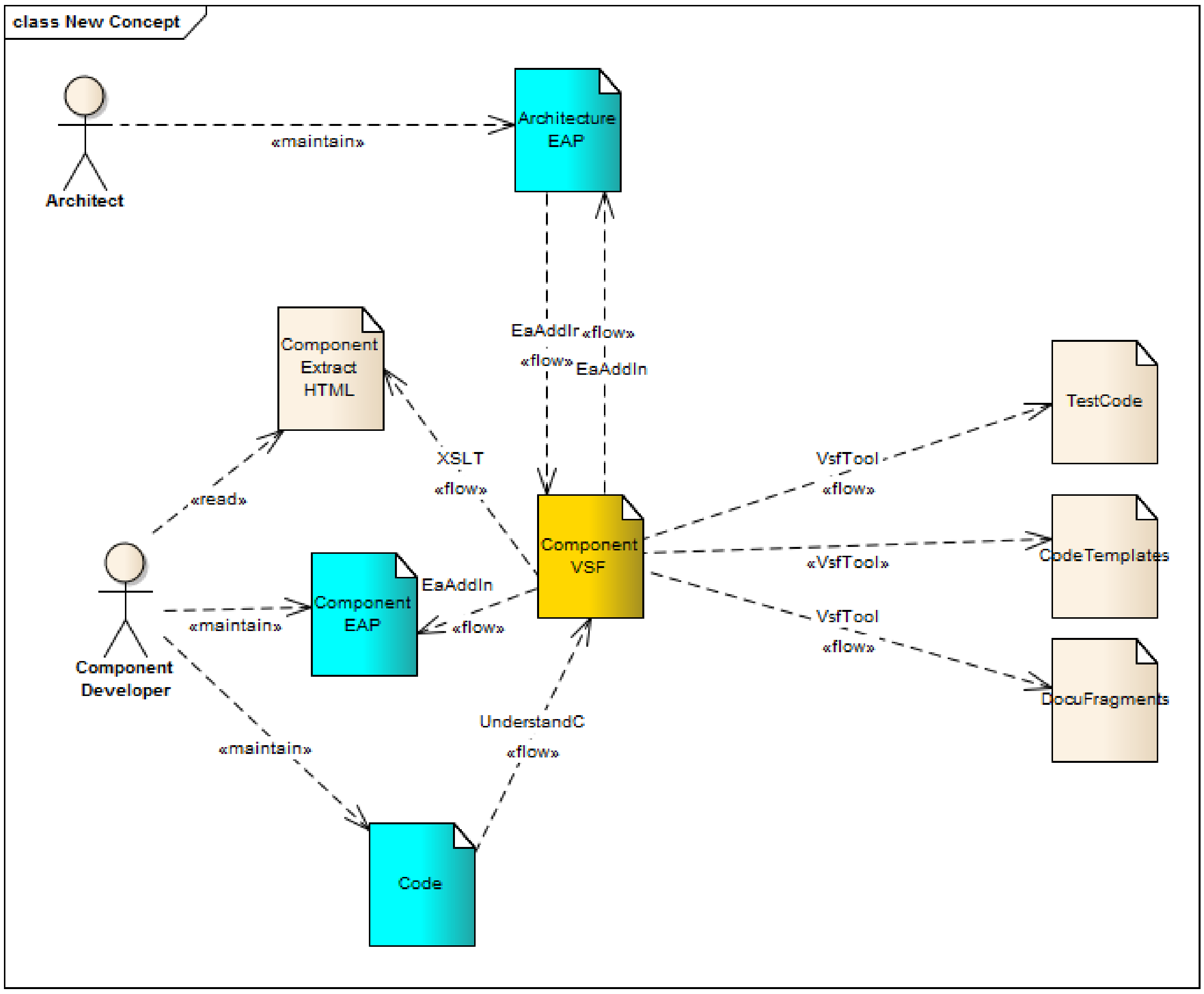


Figure 3.2: New concept - an overview

3.2 Meta model formation

In this section, meta-model of data from different sources is constructed. An extensive study on the data that sources contain was done and hence, based on that, the meta-model was made. The meta-models were made using EA. Each artifact is carefully taken into consideration along with its properties and most importantly, the relation between them. This was a very crucial step as far as conceptualization was concerned. Until the artifacts are clear and relations between them is understood, it was not possible to proceed further to analyse different formats and finally come up with a new file format. An overview of the complete model, where all the artifacts are involved is shown in 3.3.

In figure 3.3, all the artifacts that are involved in the meta-model that is under discussion can be seen. Various artifacts under examination are as follows:

- Source file

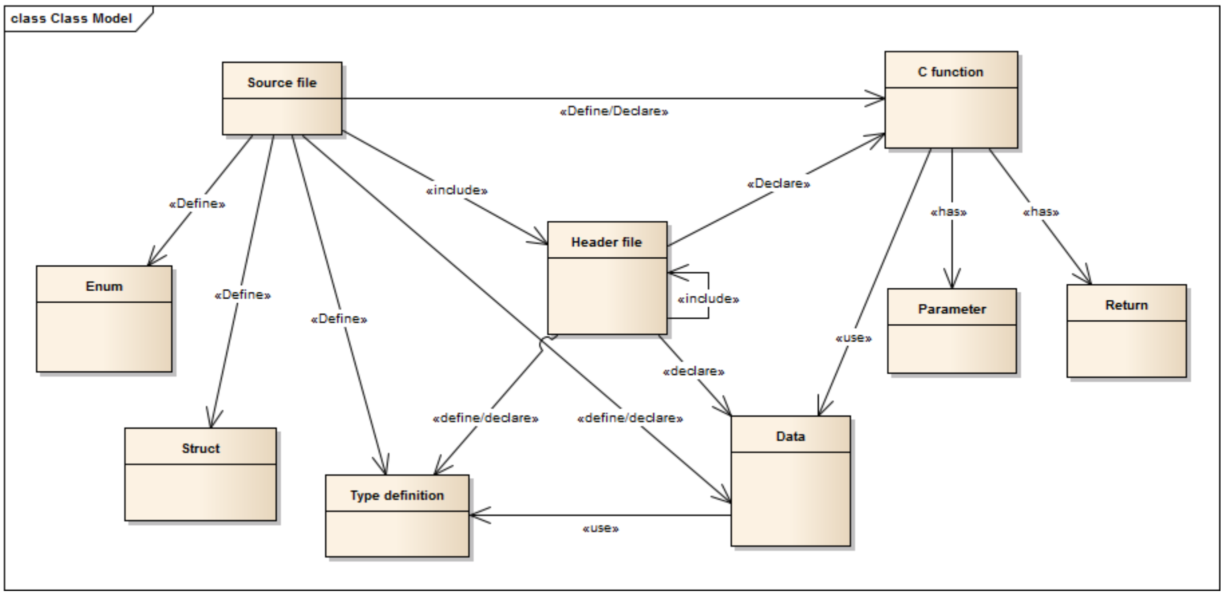


Figure 3.3: Abstract view of complete meta-model

- Header file
- C function
- Parameter
- Return
- Data object
- Type definition
- Structure
- Enumerator

Each of these artifacts are related with one another as already seen. Also, each of them have their own specific attributes. All the properties are noted and discussed will be elaborated in coming sections. But before that, meta model that comes from architecture and from source code is understood. And finally, data is analysed extensively.

3.2.1 Meta model - from architecture

This sub section specifically deals with understanding of data-artifacts that come from the enterprise architect or to be precise product architecture design. The relations between these objects was determined and also the properties of each object.

The meta-model is first shown in figure 3.4 and the explanation of each object follows.

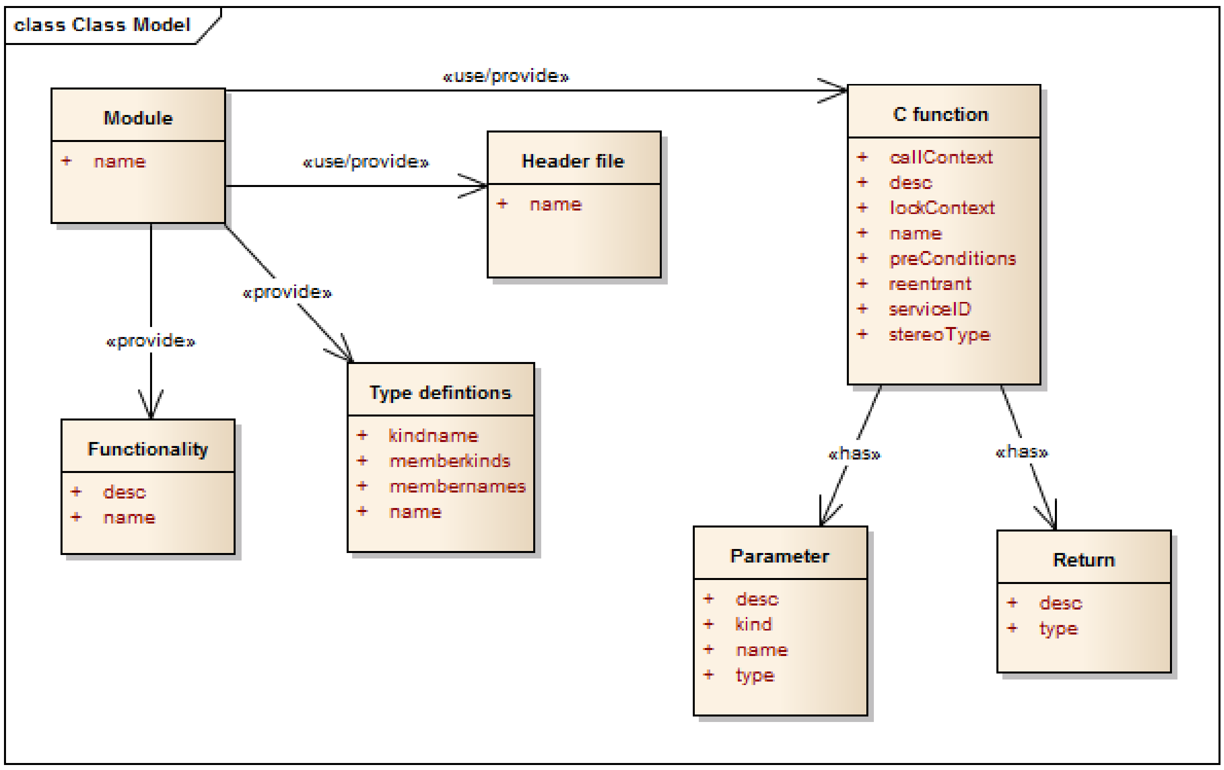


Figure 3.4: Meta model from Enterprise Architect

In the meta model 3.4 various objects/data artifacts which have the source - enterprise architect is seen. Although this is the product architecture design, we consider the artifacts/objects within each component of the product and consolidated into the following types. The overview of each artifact, along with its properties and the relations is summed up in the table 3.11:

The explanation to some of the properties of artifacts/objects is as follows:

- name - defines the name of the object.
- desc - or description as the name suggests, gives the description of the functionality of the object.
- lockContext - gives a value, which shows if the function can be called within a section of a lock interrupt or not.
- preConditions - state if there are any particular conditions that are to be taken care of before the function executes.
- reentrant - value states if a function can be re-entered while the function is executing.
- serviceId - is a identifier used for a function which is used while error logging.

Artifacts from architecture		
Artifacts	**Properties**	**Relations**
Source file	name	define, declare, include
Header file	name	includeby, include, declare, define
C function	name, call context, desc, lockContext, preConditions, reentrant, serviceId, stereoType	definein, declareby, has
Parameter	name, kind, type, desc	(none)
Return	name, desc	(none)
Type definitions	name, kindName, memberNames, memberKinds	definein, declareby

Table 3.2: Consolidated overview of artifacts from architecture

- stereoType - states the kind of the function, it can be service or call back.

The above data was gathered after analysis of the data artifacts that are present from enterprise architect.

3.2.2 Meta model - from source code

This sub section deals with understanding of data-artifacts that come from the source code files that is understand C in this case. The relations between these objects was determined and also the properties of each object. The meta-model is first shown in figure 3.5 and the explanation of each object follows.

In the meta model 3.5 various objects/data artifacts which have the source - UC is seen. The overview of each artifact, along with its properties and the relations is summed up in the table 3.3:

The properties that are seen in the table 3.3 have been already explained in the section 3.2.1. The extra objects that are noticed here are the structure and enumeration objects. The explanation to some of the properties that have not been covered earlier are as follows:

- Data - are the data objects, which are defined by source files, declared by header files and used by functions. Data objects use type definition objects.
- isConst - is the value that states if the data object is constant or not.
- memberName - name of the member of a structure.

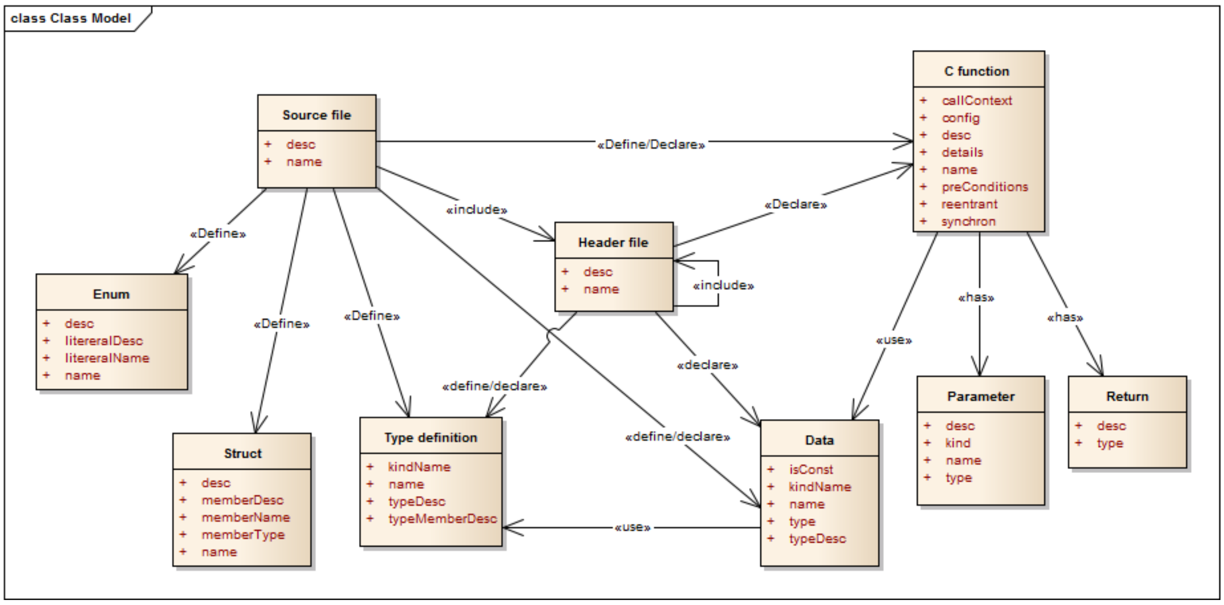


Figure 3.5: Meta model from UnderstandC

- memberType - type of the member of a structure.
- memberDesc - description of the member of a structure.
- literalName - name of the literal of an enumerator.
- literalDesc - description of the literal of an enumerator.

3.2.3 Analysis of data

This section now discusses the data that has been collected from both the sources. Here, to make it better for understanding, a table was created so that data analysis is explicitly completed. This table will also be clearly answering the question about who is the 'master of data'. Master of data is considered as the source, that defines the data initially. If the data is further refined, it is not considered as master or data. The table 3.4 shows all three sources of data, that is the product architecture, component architecture and the code.

Analysis of data				
Data	Product architecture(PA)	Component architecture(CA)	Code(CO)	Master of Data
Abstract architecture	Yes	No	No	PA
Used modules and using modules	Yes	No	No	PA
Functionality description of module	Yes	Yes	No	Both PA and CA

Continuation of Table 3.4				
Data	Product architecture(PA)	Component architecture(CA)	Code(CO)	Master of Data
State diagram	Yes	Yes	No	Both PA and CA
Control Flow and Butterfly diagram	No	No	Yes	CO
Modules	Yes	No	No	PA
Interaction between modules	Yes	Not fully	No	PA
Source file details	No	Not fully	Yes	CO
Name Desc Header file details	No No Not fully	Yes Yes Yes	Yes Yes Yes	PA
Name Desc Type definition	Yes No Not fully	Yes Yes No	Yes Yes Yes	CO
Typedef/Struct/Enum	Not fully	No	Yes	CO
Name Kind (Typedef/Struct/Enum) Type TypeDesc Members	No	Yes	Yes	CO
MemberName MembeType MembeDesc Variables (C objects)	No	No	Yes	CO
Name Kind Name Type Desc API (Routine/Function)	Yes	Yes	Yes	PA
Name Stereotype Description () Context Reentrant Synchronous Config Lock context Service ID C Parameters	Yes Yes Yes Yes Yes Yes No Yes Yes Yes	Yes Yes Yes Yes Yes Yes Yes Yes Yes No	Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes	PA
Name Type	Yes Yes	No No	Yes Yes	

Continuation of Table 3.4				
Data	Product architecture(PA)	Component architecture(CA)	Code(CO)	Master of Data
Kind (in/out) Description Return	Yes Yes Yes	No No Yes	Yes Yes Yes	PA
Type Description	Yes Yes	Yes Yes	Yes Yes	

Table 3.4: Analysis of data

The table 3.4 clearly show the data artifact into consideration along with the root location of that data. There are some extra data artifacts like state diagram, abstract architecture and some more diagrams which have not been taken into consideration when meta-modeling was done. Reason being, although it is a part of root location, it does not contain direct properties or relations that can be shown using meta-model. However, some of these artifacts will be considered in coming stages while finalizing the file format. Now with this information clearly in hand, we can move further to analyse the different kinds of format that presently describe these data artifacts and what are their advantages and disadvantages. That is another very important aspect that has to be considered and only after the examination of all these factors, a new file format can be devised/specified for the required purpose.

3.3 Format analysis

In this section, analysis of different formats that are presently available is done. They are: *UnderstandC*, *VCD* and *ReqIf*. There are three major aspects of these formats, naming:

1. Work flow
2. Structure
3. Context

Each of the formats stated here have their own advantages, uses, structure, scalability and disadvantages. By the end of the section, comparison of all these formats are done and amalgamated into one table based on requirements of the format. That will give a more evident apprehension for structuring a new file format.

3.3.1 Evaluating UnderstandC format

Understand C Database (UDB) is converted to an XML format using Perl scripting. This was used earlier to get better way of viewing all the objects it contains in the source code files and

Artifacts from source code		
Artifacts	**Properties**	**Relations**
Source file	name	define, declare, include
Header file	name, desc	includeby, include, declare, define
C function	name, call context, desc, lockContext, preConditions, reentrant, serviceId, stereoType	definein, declareby, use, has
Parameter	name, kind, type, desc	(none)
Return	name, desc	(none)
Data	name, kindName, type, typeDesc, isConst	use, useby, declareby, definein
Type definitions	name, kindName, memberNames, memberKinds	definein, declareby
Struct	name, desc, memberName, memberType, memberDesc	definein
Enum	name, desc, literalName, literalDesc	definein

Table 3.3: Consolidated overview of artifacts from source code

since it was XML, it could be easily converted to HTML format to have even better look at it, for understanding purposes. A small part of that XML file is considered, to get an idea of how does it access the objects and how is it structured.

```
<ent kind="Function" kindlong="C␣Unresolved␣Function" name="
   ComM_MainFunction_0" id="719" VScope="global" type="void">
  <comment></comment>
  <declarein kind="File" kindlong="C␣Header␣File" name="
     ComM_Lcfg.h" id="700" line="104" column="29"/>
</ent>
```

To understand the XML format of UC, a meta model is shown in 3.6.

As seen in 3.6, the ent/entity node has attributes such as id, kind, kindlong, name, type and scope. Some of these attributes are not required within the node itself. Also, relation node contains id, kind, kindlong and name. Again, kind and kindlong, both are not necessary information. Any one or combined information would be better. There are 3 kinds of relations involved,

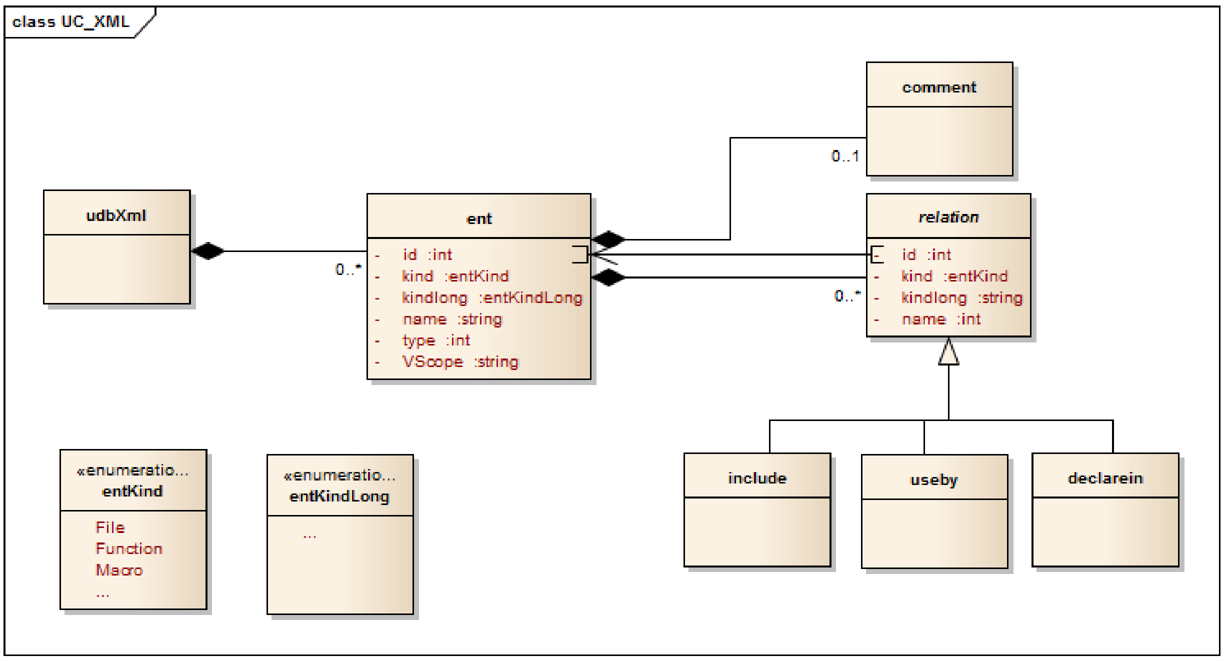


Figure 3.6: XML format of UC

that is include, useby and declarein - shown as examples. And some examples of kind is also mentioned, like file, function, macro, etc.

The work flow of this format can be seen in 3.7.

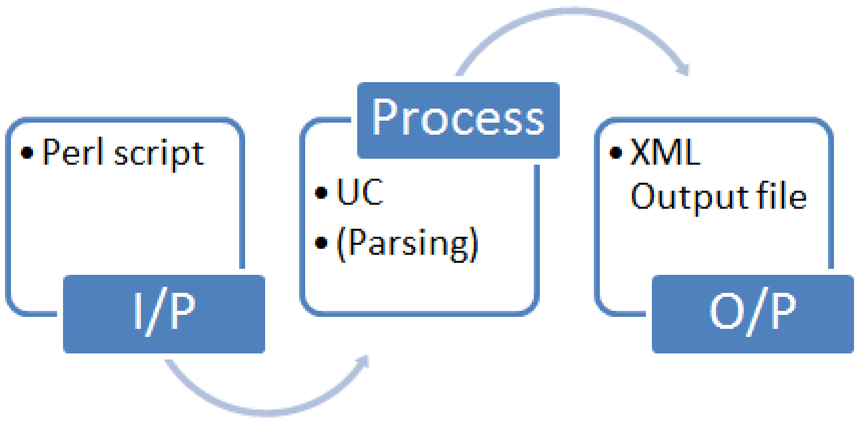


Figure 3.7: Work flow of UnderstandC

Now with this format, comparison is made of the data-model discussed in the previous sections. When it comes to handling such large amounts of data, and each data object having more than two attributes at the least, there arises a need to evaluate if this kind of listing is efficient to be

used. Next thing considered is the encapsulation of data when more than one component needs to be realised in the same file. Do keeping objects in parallel working well in that case as well or not is also examined.
Knowing the fact that there are more than one component in any product development, following this format only gives the wide overview of all the existing objects. Even when it comes to relations, having only the id and direction of the object related is not sufficient. It makes knowing all the relations more complicated, because each time there is a need to search for that id in the entire file. Expanded version of this example can be found in appendix section A.1.1.

So summing up the results, there are both advantages and disadvantages in this file format. They can be summed as follows -

Advantages:

- Suited as import/export format.
- The generic method of addressing objects is a good concept. Since any kind of object can be defined using that, and with the attribute kind, it gives a wider usage.
- The concept of id is also good since, each object has it's unique identifier and makes it easier to access or refer later. But we will discuss the other possibilities of other cases in coming sections.

Disadvantages:

- All the attributes are contained within the same tag, which creates a lot of confusion when a certain object has more than, say 2 attributes.
- The relations are addressed based on the kind name directly. Since it is not generic, tag has to be explicitly defined.
- The attributes 'kind' and 'kindlong' are redundant. This can be avoided.
- The line and column are not so significant and take up space for not a real good reason.

3.3.2 Evaluating VCD format

Vector (VCD) is format that was developed at Vector Informatik for internal purpose. This format deals with the data from EA. Converts the product architecture design into VCD format which is XML in structure. This format is used to convert data into HTML for reference purpose. All the data that is available in EA, is converted to VCD format.
In the following lines is an example of part of the file is given and analyses follows.

```
<Method GUID="{0C68D9B3-09A3-478a-8E3E-495051F7F527}" Name="
   LinSM_Init" Kind="ServiceFunction" Return="void">
 <Notes><![CDATA[This function initializes the LinSM.<br/>]]></
    Notes>
  <Tags>
   <Tag Name="CallContext">
    <Value>Any</Value>
   </Tag>
<!--.........many more tags............-->
   <ReturnType>void</ReturnType>
   <Parameter Name="ConfigPtr" Kind="in" Position="0" Type="
      const␣LinSM_ConfigType*" Type_Plain="LinSM_ConfigType">
    <Notes><![CDATA[Pointer to the LinSM post-build
       configuration data.]]></Notes>
   </Parameter>
   <UsedByModule GUID="{BE85B771-ECFF-4a77-961B-7327B1154E8E}"
      Name="EcuM______TBDel" />
</Method>
<!--...................................-->
<Dependency Type="Realisation" Stereotype="realize" Name=""
   Direction="out" MyMultiplicity="" TargetMultiplicity=""
   LinkedGUID="{5E542F8C-ADCD-49f6-89E7-85195020A637}"
   LinkedName="LinSm" LinkedType="Interface" LinkedStereotype="
   EmbeddedInterface">
 <Notes>BswM_LinSM_CurrentSchedule BswM_LinSM_CurrentState</
    Notes>
</Dependency>
```

In the above example the first thing that is noticed is, there is hierarchy in the structure. When it comes to objects, the nodes are named based on the kind of object itself. Example , node named 'method'. This object is of kind function/API/routine/method. The attributes are within the same node. When it comes to the properties of node 'method', it has sub nodes with name - 'tags and tag'. Each 'tag' has a name and value. Similarly, for each parameter in the method, it is under the concerned method and node name is 'parameter' with similar form as 'method'.

Each node is explicitly referred by it's name. Also, node name 'dependency' with attributes 'type' and 'stereotype', along with 'name', 'direction', 'multiplicity', 'targetmultiplicity', 'linkedname', 'linkedtype' and 'interface'. All these properties, are attached with each 'dependency' node. This makes it more complicated for a developer to understand. The information about multiplicity, etc is not really needed at that level. Not all the information given, would be useful. Thus, in the format that is going to be designed, we must take care that only the information that is needed by the developer (as based on results of the survey) must be given first preference. Expanded version of this example can be found in appendix section A.1.2.

Based on the example, and analysis, a list of advantages and disadvantages can be listed as follows -
Advantages:

- The structure is hierarchical.
- Suited as import/export format.
- Value of the property can be easily accessed.

Disadvantages:

- Too many attributes within the tag.
- Tag name specifically named.
- Nodes could occur multiple times
- Dependency not so clear to understand.
- A lot of information that is not required by the developer.

3.3.3 Evaluating ReqIf format

In this section, analysis is done of the format used by ReqIf. It is a standardized meta-model in XML schema. Since it is used to exchange information between the Original equipment manufacturer (OEM)'s and the suppliers, it is essential to understand their schema as well. There could be certain concepts that could be reformed and used for the intermediate file format that has to be created.
In the following lines is an example of part of the file is given and analyses follows.

```
<DATATYPE-DEFINITION-ENUMERATION IDENTIFIER = "Status" LAST-
   CHANGE = "2011-06-13T00:00:00+01:00">
 <SPECIFIED-VALUES>
  <ENUM-VALUE DESC = "Requirement has been proposed."
     IDENTIFIER = "Proposed" LAST-CHANGE = "2011-06-13T00
     :00:00+01:00">
   <PROPERTIES>
    <EMBEDDED-VALUE KEY = "1" OTHER-CONTENT = "foo" />
   </PROPERTIES>
  </ENUM-VALUE>
  <!--..................................-->
 </SPECIFIED-VALUES>
</DATATYPE-DEFINITION-ENUMERATION>
```

In the example above, a generic approach is seen while defining the objects. Something that is also easily noticed is the structure of the file is hierarchical.
Something that is not seen in other formats is a separate hierarchy for the definition of all relations. It defines the source, destination and the id of the relation. Expanded version of this example can be found in appendix section A.1.3.

Based on the example, and analysis, list the advantages and disadvantages of the format can be listed as follows -
Advantages:

- The structure is hierarchical.
- Suited as import/export format.
- ReqIf has a section to store tool specific settings.

Disadvantages:

- Extreme generic structure.
- Less useful as internal meta-model for tooling.
- Not many tools have ReqIf as internal meta-model.
- The specification of the structure (SpecType) and its values (SpecObject) reside often in the same XML. If this is split, the specification could be reused for multiple value files. But this is not indented this way and therefore at times creates problems reusing the specification and ends up in specification duplication (think about having XML scheme delivered within every XML).
- Many tools have misused this making ReqIf exchange across tools impossible.

3.3.4 Evaluating AUTOSAR format

In this section, analysis of the AUTOSAR meta-model is done. This example is taken not to understand with respect to contents but the structure of the meta-model.
A small part of the example is shown as follows.

```
<SHORT-NAME>Msn</SHORT-NAME>
 <DESC>
  <L-2 L="EN">Configuration of the Msn (Generic Component)
     module.</L-2>
 </DESC>
 <CATEGORY>VENDOR_SPECIFIC_MODULE_DEFINITION</CATEGORY>
<!--.................................-->
 <MODIFICATIONS>
  <MODIFICATION>
   <CHANGE>
```

```
        <L-2 L="EN">SW Version Number set to 3.00.03</L-2>
   </CHANGE>
  </MODIFICATION>
 </MODIFICATIONS>
```

In the example above, one major aspect that stands out is the structure of the file. It is clearly hierarchical in nature. Next important thing that is noticed is that every object or it's property has specific node names. Each node has its Unique identification number (UID). So either a node is referenced by its UID or by its name which can be tagged along with partial parent name.
This specific example does not deal with objects like files or functions like in the examples above as noticed. Expanded version of this example can be found in appendix section A.1.4.

Based on the example, and analysis, a list the advantages and disadvantages of the format is as follows-
Advantages:

- The structure is hierarchical.
- Used with the latest version of AUTOSAR.
- Has concepts of unique ID associated with every node used.

Disadvantages:

- Extremely specific node names.
- Will not comply if only ID is unique due to ID generation from different sources.
- Lot of extra information when it comes to giving it as a reference for developers.

3.3.5 Comparison of formats

In this section, the concept analysis of aspects is narrowed down. Here, aspects that are discussed are:

1. **Objects**
2. **Relations**
3. **Properties**
4. **Hierarchy**

Each aspect is individually comprehended as follows:

1. **Objects:**

   In the table 3.5, it is noticed that objects can have 2 variants. In the first variant, each node is individually named. But in the second approach, it is generic and differentiated based on the kind attribute of the object. This makes it a better choice for the new file format.

Variant	Example

A) Individual XML nodes	`<file name="File.c" id="1"/>`
	`<function name="Sample_Init" id="2"/>`

B) General XML nodes with kind attribute	`<object kind="File" name="File.c" id="1"/>`
	`<object kind="Function" name="SampleInit" id="2"/>`

Table 3.5: Object variants

XML Path Language (XPath) is a language for addressing parts of an XML document, designed to be used by both XSLT and XML Pointer language (XPointer) [JC15]. This is important different kinds of XML nodes are used. And XSLT is used for accessing the nodes in the XML file.
Next, XPath access samples are shown for the objects:

	Variant A	**Variant B**
All objects of certain kind	file	object[@kind="file"]
All objects	file | function | . . . (in some cases * is possible)	object
Object by name (id,. . .)	*[@name='Sample_Init']	object[@name="Sample_Init"]

Table 3.6: Sample XPATH access

Clearly, it is easier to access the object if it is a generic node. Only the kind has to be specified. Makes it more efficient to make changes and design XSLT.

2. **Relations**:

   Table 3.7 shows the variants of object relations. As seen in examples, it is always better to have generic nodes. This helps in scaling the model as per requirement. In this case, variant D has better approach for defining relations.

Variant	Example
A)	`<object kind= File id= 1 >`
	`  <include id= 2 >`
	`<object/>`
	`<object kind= File id= 2 />`
	`  <includeBy id= 1 >`
	`<object/>`
B) Relation as own XML node	`<object kind= File id= 1 >`
	`  <relation id= rel1 >`
	`<object/>`
	`<object kind= File id= 2 />`
	`  <relation id= rel1 >`
	`<object/>`
	`<relation id= 1 kind= include sourceId= 1 targetId= 2 >`
C) (like A, but with direction)	`<object kind= File id= 1 >`
	`  <include dir= out id= 2 >`
	`<object/>`
	`<object kind= File id= 2 />`
	`  <include dir= in id= 1 >`
	`<object/>`
D)	`<object kind= File id= 1 >`
	`  <rel kind= include id= 2 >`
	`<object/>`
	`<object kind= File id= 2 />`
	`  <rel kind= includeby id= 1 >`
	`<object/>`

Table 3.7: Relation variants

3. **Properties**:

   Table 3.8 shows the variants of object properties. As seen in the examples, there are different possibilities. But, when it comes to accessing the values of the properties, it is easier to access with the generic node name. It can be accessed based on name attribute of the property node, thus making variant C a better choice for designing new file format.

Variant	Example

A) as XML attributes	`<object kind=File id=1 scope=global editable=true/>`
	`<object kind=Function id=2 callcontext=any reentrant=false/>`

B) as specific XML nodes	`<object kind=File id=1 >`
	`  <scope>global</scope>`
	`  <editable>true</editable>`
	`</object>`
	`<object kind=Function id=2 >`
	`  <callcontext>any</callcontext>`
	`  <reentrant>false</reentrant>`
	`</object>`

C) as general XML nodes with kind	`<object kind=File id=1 >`
	`  <prop name=scope>global</prop>`
	`  <prop name=editable>true</prop>`
	`</object>`
	`<object kind=Function id=2 >`
	`  <prop name=callcontext>any</prop>`
	`  <prop name=reentrant>false</prop>`
	`</object>`

Table 3.8: Properties variants

4. **Hierarchy**:

   Aspects: How is hierarchy modelled? By "containment"-relations or by direct hierarchy? Table 3.9 shows the variants of hierarchy. As per examples seen in the table, having a flat model makes it difficult to understand the inter-relation of objects. Thus, having a hierarchy to certain extent is better for understanding and also with respect to the data model of new file format.

Variant	Example

A) Flat	`<object kind= File id= 1 >`
	`  <rel kind= contain relObjId= 2 />`
	`</object>`
	`<object kind= Function id= 2 >`
	`  <rel kind= containedBy relObjId= 1 />`
	`</object>`

B) Hierarchy	`<object kind= File id= 1 >`
	`  <object kind= Function id= 2 >`
	`  </object>`
	`</object>`

Table 3.9: Hierarchy variants

To consolidate all the data that was just analysed at abstract and detailed level, the comparison table 3.10 sums it all up for all the various formats.

File format	Object	Hierarchy	Properties	Relations
VCD	B	B	B,A	D
UCXML	B	A	A,B	A
ARXML	A	B	B	
ReqIf				
VSF	B	B	C(,A)	D

Table 3.10: Comparison of variants in various formats

In the table 3.11, the abstract view of all different formats with respect to the different artifacts and structure is seen.

Now, the conceptualization of the intermediate format is explained in the following section.

File format	Object	Relation	Hierarchy
UC	Generic definition, attributes within same tag	Identified defined using "kind" of relation directly. Ex. <use> with its properties	No
VCD	Generic definition, attributes with specific tags	Defined using <dependency> tags with its properties	Yes
ReqIf	Generic approach while defining	Separate hierarchy for relations with source, target and id	Yes
AUTOSAR	Generic approach while defining with unique Id's	-	Yes

Table 3.11: Analysed Data - Consolidated

3.4 Intermediate format conceptualization

In this section, the conceptualization of the intermediate format will be explained. This would include the naming conventions of the objects in the first sub section. That would be followed by the hierarchy of the format and finally the definition of relations. Each of the sub sections have to considered from both architecture and source code point of view. It is also found that the objects that come from both do not match completely. Some data artifacts are only applicable from architecture point of you some only from source code.
Overall, some important properties of intermediate format are:

- Hierarchical in nature (Variant B of table 3.9)
- Generic approach for the nodes (Variant B of table 3.5)
- All information with respect to a single object directly available, along with the extra information
- Exclusive artifacts like parameters and relations, directly placed under respective object for making it easier to edit (add/delete)
- Covers all the advantages of all the formats analysed and overcomes the disadvantages for them

Figure 3.8 shows the complete file model of Vector specification file (VSF) format.

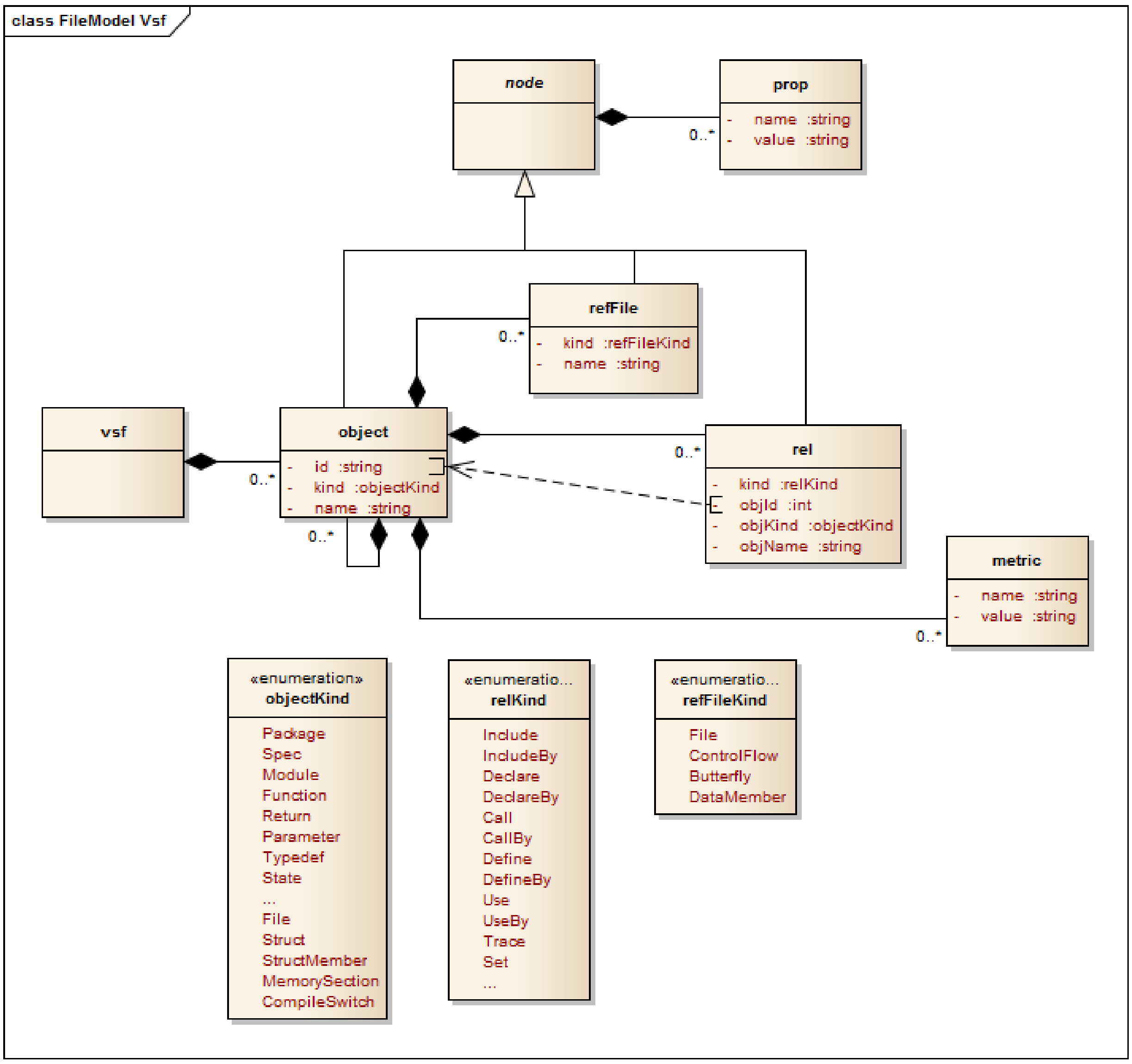


Figure 3.8: File model of VSF format

All the objects that are covered in the model are shown in 3.8. Along with the object names, properties of each are also seen. Most important points to be considered while understanding the file format are:

- Each VSF file contains multiple objects
- Each object has properties, relations and reference and metrics
- There are many different kinds of objects which can be seen in objectKind enumeration
- All the relation kinds covered in the model can be seen in relKind enumeration
- All the reference file kinds can be seen in refFileKind enumeration

- The objId under the relation node, refers to the id of individual node. It can be seen with the dotted lines between objId and id

3.4.1 Naming conventions

In this section, the naming conventions used and the purpose of using them is understood. This is the first aspect into consideration.

1. **Object** -
   All the artifacts considered have the name **'object'** and can be generalized.
   Each object has these attributes:

   - kind - kind of the object
   - name - name of the object
   - id - unique identification is generated for each object

   An example of object node in the VSF file can been here:

```
<object kind="Function" name="Function1" id="1">
 <!--properties-->
 <!--relations-->
 <!--refFiles-->
</object>
```

   Under this node, it can have properties with values and relations nodes. Different kinds can be differentiated by their property 'kind'. The possible kinds of objects can be as follows:

Objects	
Kind	Description
Spec	specification kind with name attribute. This is the outermost node, which contains the component and all it's sub-parts.
Component	component kind with name attribute. This is a sub-node of every specification and contains all the packages in a component.
Package	package kind with name attribute. There are several packages under a component when data comes from architecture. When it comes from source code, we have one main package under the component.
Diagram	diagram kind with name attribute. This mainly roots from the architecture when we have state machines and so on.

Continuation of Table 3.12	
Kind	Description
State	state kind with name attribute. This also comes mainly from architecture which contains relations from that state to another state.
Function	function kind with name attribute. When data comes from architecture, function can be of kind callback or service. But when this object originates from source code, it is not possible to decipher it's kind.
Return	return kind with type as its property. This gives the information of the return kind of a function and is always under the function node.
Parameter	parameter kind with name attribute. This object is always function specific and contains the various properties of the parameter under it.
File	file kind with name attribute. File can have various properties and relations within. Also, it can be header or code file. That can be known based on the extension.
Typedef	typedef or type definition kind with name attribute. This object specifies the type as its property. If a file or function defines a type definition, it is shown by using relations.
Struct	struct or structure kind with name attribute. It contains all properties, members, members details and all the relations with files and functions is placed within it.
Enum	enum or enumerator kind with name attribute. Similar to structure, it contains all properties members, members details and all the relations with files and functions is placed within it.
Data	data kind with name. This object describes all the variables that have been defined. It has its properties and also all the relations with files and functions.
DesignFeature	designfeature with name attribute, also comes from component abstract design level. It contains the description and properties related to the object.

Table 3.12: Kinds of objects

2. **Property** -
   Under each object, there are properties. They have the name **'prop'** with name as its attribute and value set within the node.
   The following example shows the property node with value:

```
<object kind="Parameter" name="ConfigPtr">
 <prop name="Type">const LinSM_ConfigType*</prop>
 <!--more properties-->
```

3. **Relation** -
   Relation node have the name **'rel'**. Each relation has these attributes:
   - kind - kind of the relation
   - objKind - the kind of the object that is related
   - objName - the name of the object that is related
   - objId - the unique identification of the object that is related

   All the relations can be generalized based on the node name 'rel'. This can be differentiated by their property 'kind'. The possible kinds of relations can be seen in table 3.13. The suffixes like 'by', and 'in' are used when the same relation is expressed in opposite direction between the objects.

Kind	**Description**
include/includeby	used when an object includes another object
define/definein	used when an object provides definition for another object
use/useby	used when an object uses another object
declare/declarein	used when an object declares another object
set/setby	used when an objects sets a value for another object
call/callby	used when an object is called by another object
modify/modifyby	used when an object is modified by another object
follow/followby	used when an object is followed by another object
contain/containedin	used when an object is contained within another object

Table 3.13: Kinds of relations

Not all relations are available in all data sources. Some of the relations are only present in architecture while remaining come from the source code.
An example of its definition can be seen here:

```
<object kind="File" name="BswM_LinSM.h" id="974">
 <rel kind="Include" objId="976" objKind="File" objName="
    LinSM.h"></rel>
```

4. **RefFile** -
   Reference file nodes can also be present in the file. Each of these nodes has these attributes:
   - kind - reference file that is referenced
   - name - name of the reference file that is referenced

   It also contains property of the name path, and it's value gives the path of the file referenced. Kind of the reference can be as follows:

Kind	**Description**
File	This is available when data comes from architecture. It provides path to the file it references.
Butterfly	This comes from source code. It refers to the butterfly kind of diagrams.
Data members	This comes from source code. It refers to the data members kind of diagrams.
Control flow	This comes from source code. It refers to the control flow kind of diagrams.

Table 3.14: Kinds of reference files

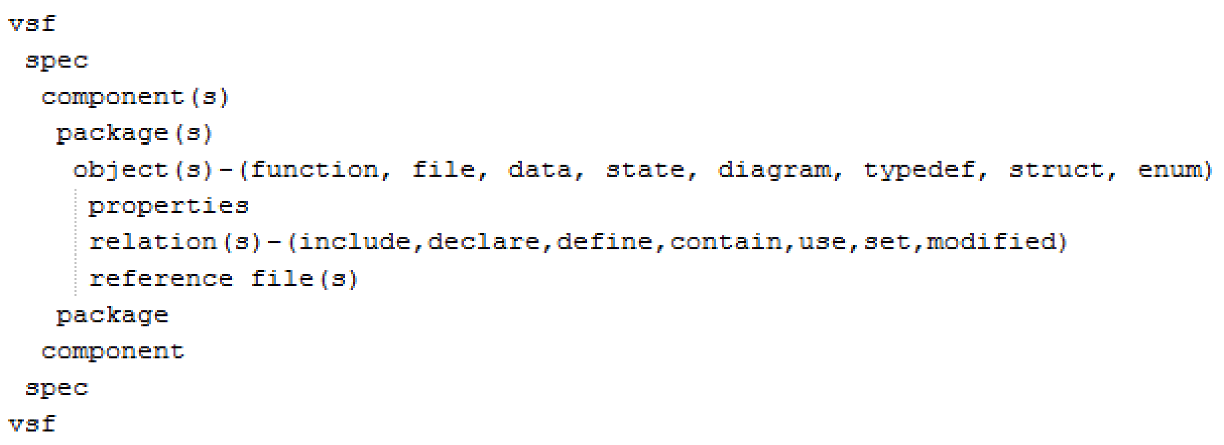
An example of its definition can be seen here:

```
<object kind="File" name="BswM_LinSM.h" id="974">
 <refFile kind="Butterfly" name="Butterfly_BswM_LinSM.h">
  <prop name="Path">RefFile\BswM_LinSM.h_Butterfly.png</
     prop>
 </refFile>
```

As seen above, various kinds have been generalized under the name 'object','prop' and 'refFile'. This makes it easier to scale this model later. In case of any more objects and properties of different kinds or names in case of properties, it can be easily added and included in the meta-model.

3.4.2 Hierarchy

In this section, the finalized structure of the format is seen in terms of it's hierarchy. In the analysis before, it was noticed that different formats have different structures and based on the pros and cons of each, the structure has been finalized. It can be seen as follows with simple example in the form of a diagram in 3.9.



```
vsf
 spec
  component(s)
   package(s)
    object(s)-(function, file, data, state, diagram, typedef, struct, enum)
     properties
     relation(s)-(include,declare,define,contain,use,set,modified)
     reference file(s)
   package
  component
 spec
vsf
```

Figure 3.9: Hierarchy of format - An example

The hierarchy of the format can be clearly noticed in the figure 3.9 . All the objects under package node are in parallel. There can be more than 1 package within a component and multiple components within a specification. In the figure,the structure below each object can be seen separately.

To give it more specific name, this format was named **'VSF'** or **'Vector Specification Format'** and has the extension '.vsf'.

The following example shows the hierarchical structure of the file:

```
<vsf name="test1">
 <object kind="Spec" name="MSR4">
  <prop name="CreateDate">2015-07-13 15:26:23</prop>
  <object kind="Component" name="LinSM">
   <object kind="Diagram" name="LinSM" id="1">
    <prop name="Type">Logical</prop>
    <rel kind="contain" objKind="Class_Module" objName="LinSM"
       objId="2" />
    <refFile kind="File" name="LinSM">
     <prop name="Path">RefFiles\Sample_Path__Name.png</prop>
    </refFile>
   </object>
   <!--.........-->
  </object>
 </object>
</vsf>
```

An entire example of VSF file can be found in appendix A.1.5.

4 Implementation

This chapter describes the phases of implementation. It begins with identifying all use cases that have to considered here. After which conversion of data is discussed. And finally, the comparing and merging data mechanism is realized.

4.1 Use Cases

Before discussing the different use-cases, it is essential to understand what exactly does use-case mean and why is it needed?
From software and systems engineering point of view, there is a technique to realize the interaction of software or a system with its users. When the user interacts with it, he/she uses the data. What data is used, how is it useful for the user and how can that data be realized are the questions that arise. In order to figure the questions out, this technique of classifying use-cases based on interaction, the contents of the software or system and identifying the end result - gives possible use-cases. In the use case diagram 4.1 the use-case is seen in form of a interactive diagram between architecture and component development levels. The use case diagram 4.1 will be split into modules and discussed in coming sections. These use cases have been derived after the analysis of results from the survey that was conducted. The use cases are classified in 3 main sections:

1. **Information from architecture** -

   The data that is present at architecture level comes from the requirements of the customer. In this case, product architecture design is designed by the architects and this data is then given to the component developers as their input of requirements. At this moment, what data is given as input is realized, how is the data passed or given as input in terms of format of the file. The following use case seen in 4.2 describes the 'information from architecture' evidently. In 4.2 the interaction between the architect(s) and the data is seen and also the formats involved. The architect(s) designs the product architecture design. The data that is relevant for the component developers is exported. It is exported in form of a VSF file and also converted to HTML format.

2. **Review within component development** -

   This use case is related to the component developer. What is done, what data is used, what is developed and finally the developed component is analysed. All these actions are a part of this use case. The use case 4.3 that is shown, gives more clarity. In 4.3 the actions

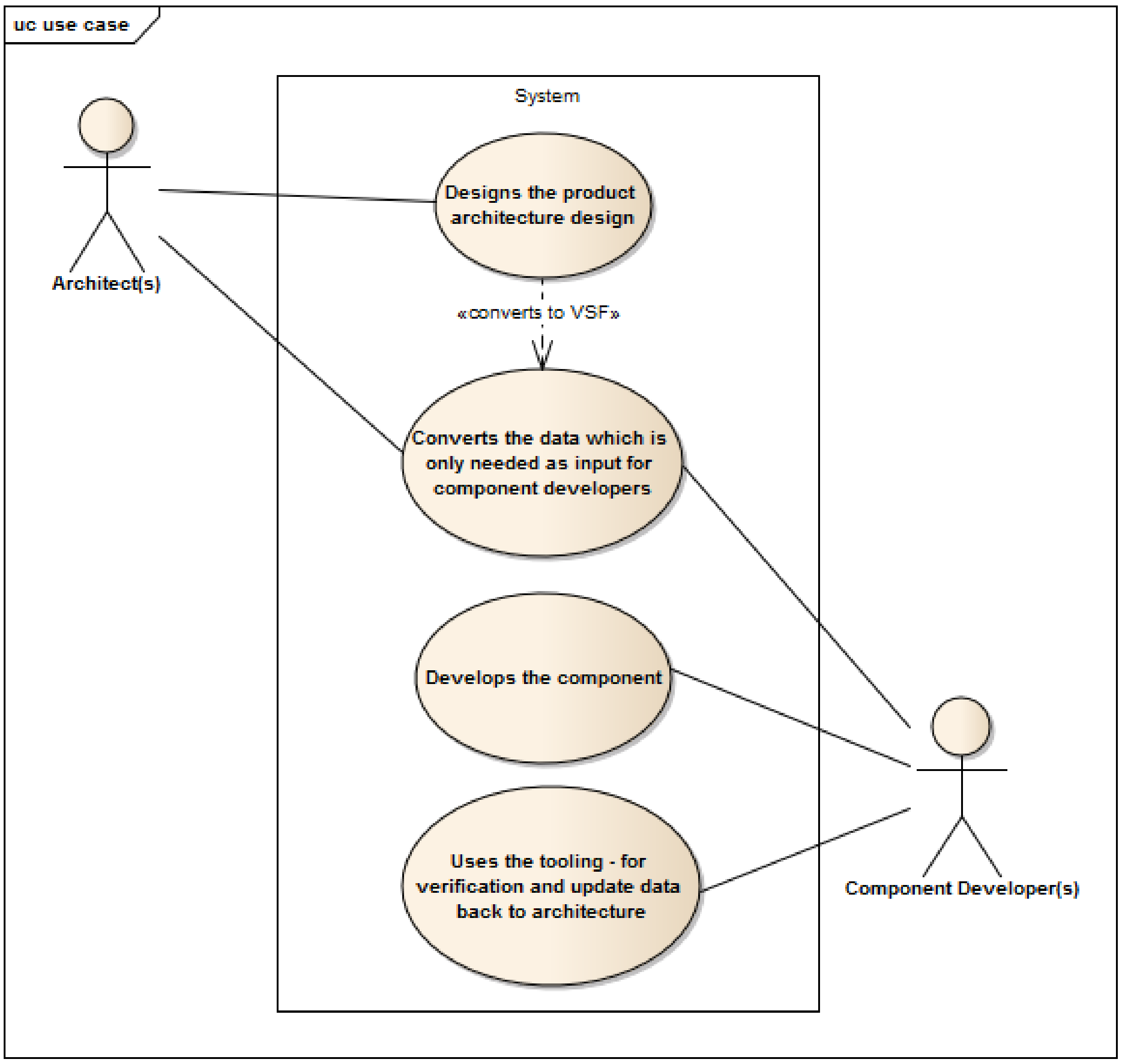

uc use case
System
Architect(s)
Designs the product architecture design
«converts to VSF»
Converts the data which is only needed as input for component developers
Develops the component
Uses the tooling - for verification and update data back to architecture
Component Developer(s)

Figure 4.1: Usecase diagram between architecture and component development level

involved with respect to the developers is realised. This can also be considered as the second half of the round-trip concerning data.

3. **Interaction with architecture -**

   This use case is the final step and shows the tooling involved within this thesis. After the component has been developed, it has to interact with the architecture. This is done for 2 purposes: First, it can be used to verify against the architecture and secondly, it can be used to update the architecture in case of changes. The use case 4.4 that is shown, explains the use case comprehensibly.

   In 4.4 the tooling is contained within the system, since it can be considered as an additional

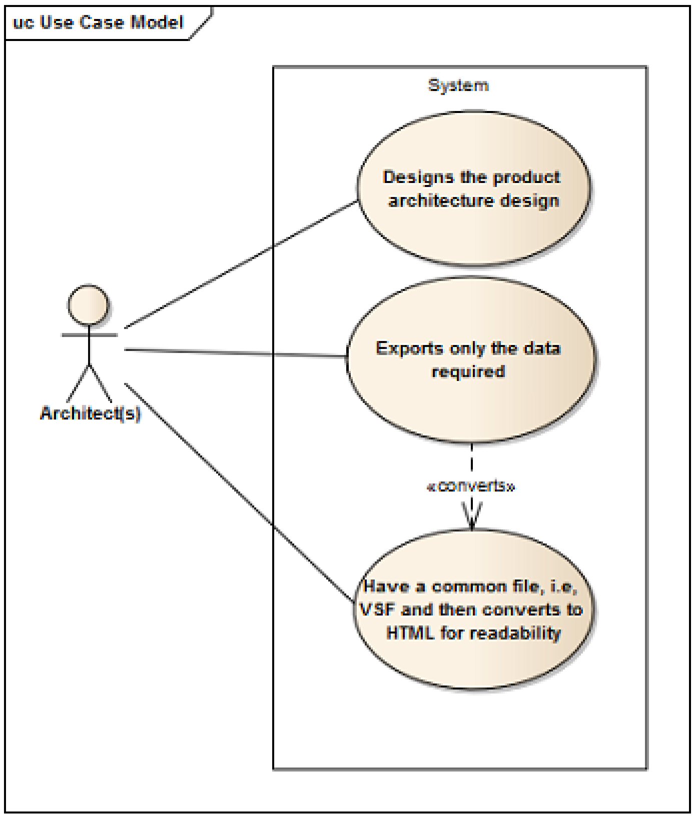

uc Use Case Model
System
Designs the product architecture design
Exports only the data required
«converts»
Have a common file, i.e, VSF and then converts to HTML for readability
Architect(s)

Figure 4.2: Use Case 1 - Information from Architecture

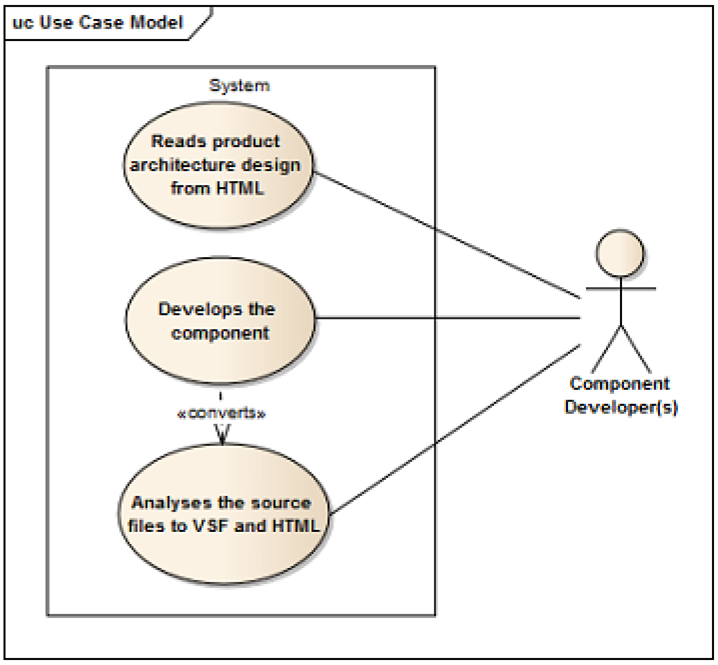

uc Use Case Model
System
Reads product architecture design from HTML
Develops the component
«converts»
Analyses the source files to VSF and HTML
Component Developer(s)

Figure 4.3: Use Case 2 - Review within component development

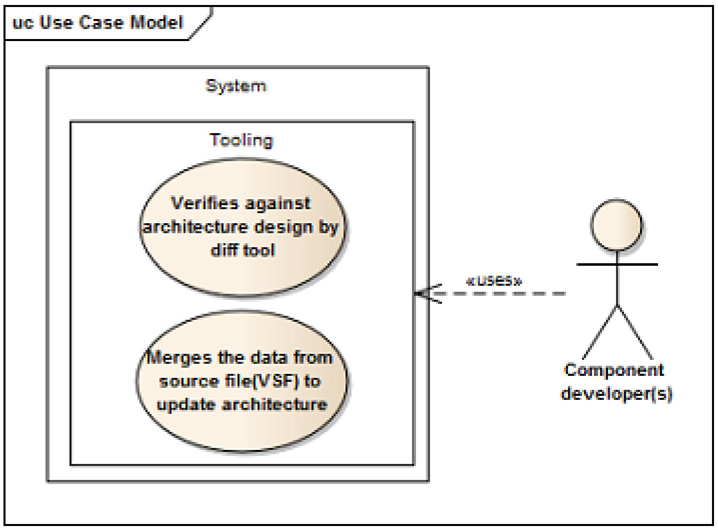

uc Use Case Model
System
Tooling
Verifies against architecture design by diff tool
Merges the data from source file(VSF) to update architecture
«uses»
Component developer(s)

Figure 4.4: Use Case 3 - Interaction with architecture

module to the existing system. It purposes of verification and updating data is clearly seen.

4.2 Conversion of data

This includes converting data from EA to the VSF and converting data from UDB to VSF. The subsections of each of these sections discuss the details of the data considered from each source, challenges that were faced and the abstract description of the final format from each of these sources. These refer to the data that is discussed in the data models of EA (3.4) and UDB (3.5).

In EA, the data is stored in form of classes, packages, interfaces, etc. in the database. The objects are accessed individually within the environment.

In UDB, data is stored in form of entities and references. Each entity can be accessed based on its kind. And after the kind is matched, the other attributes can be accessed easily in UDB. The relationship between these entities are called references. Each grouping of the reference kinds are in both directions. For e.g., declare and declarein.

Now that it is clear how data is organized within the database of each source, the subsections further explain the conversion of data from each source.

4.2.1 Converting EA to VSF

In this section the data that comes from architecture and what should be used while converting or exporting data to VSF file is mainly discussed. This section is divided into 3 main parts, and they are:

1. General details
2. Implementation details
3. Abstract description of result

1. **General details -**

   It was discussed in the previous section (3.2.1) about what data is actually present in architecture design and what is actually to be used as part of the data model. In the following figure 4.5, the objects that are exported from EA and which is used to convert into the VSF file are seen.

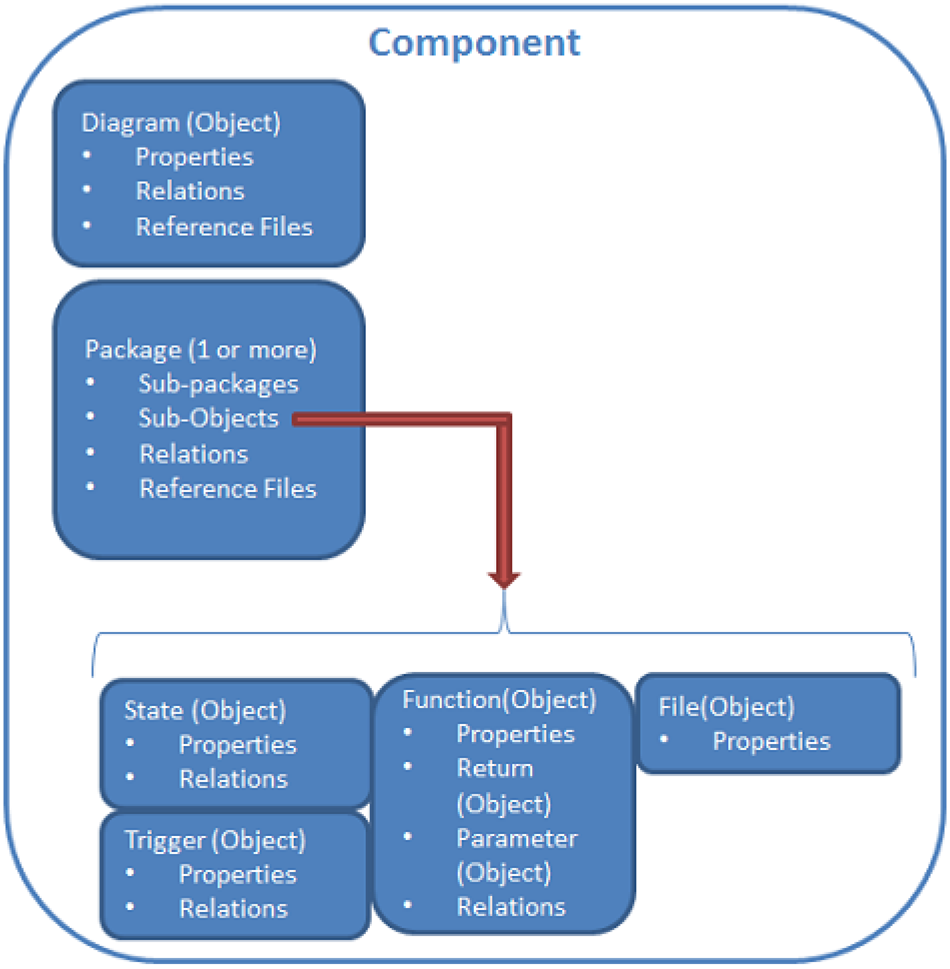


Figure 4.5: Data artifacts exported from EA

As seen in figure 4.5, all the data artifacts that are shown, along with its attributes and related objects, are based on the information that was collected during the survey. It was found that a lot of extra information was given from architecture product design to the component development level. Mainly very specific details related to the component was needed. The information about the interaction between modules, the diagrams like the

state machine, class diagrams and sequence diagrams was required. This gave a clear understanding of the requirements from that particular component.
Each diagram has its properties, modules relation and reference files. These diagrams can be present before the details of specific packages as well. Based on different requirements that is provided by the customer, product architecture design is divided into several packages. Each of this package may contain sub-packages and various kinds of objects. Some of the kinds of objects are:

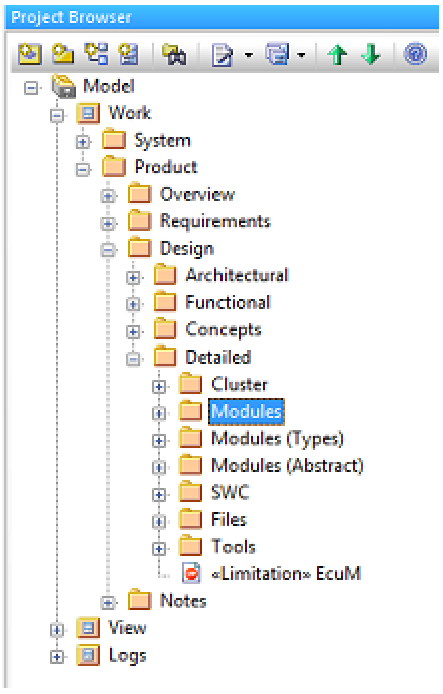
- State (diagram)
- Trigger
- Function
- File

This sums up the data artifacts that are needed for export.

2. **Implementation details -**

The implementation of this export was done using C# technology. When it comes to extracting data from the EA, Csh is used to fulfil the function since it can access the artifacts directly from the database of EA and helps to meet the requirements.

To begin with, an add-in was created in EA, which gave an option of directly converting the component that was selected into VSF. The screen shots of that add-in can be seen in figure 4.6 and 4.7.



Project Browser

- Model
  - Work
    - System
    - Product
      - Overview
      - Requirements
      - Design
        - Architectural
        - Functional
        - Concepts
        - Detailed
          - Cluster
          - Modules
          - Modules (Types)
          - Modules (Abstract)
          - SWC
          - Files
          - Tools
          - «Limitation» EcuM
      - Notes
  - View
  - Logs

Figure 4.6: Step 1 - Selecting component to be converted

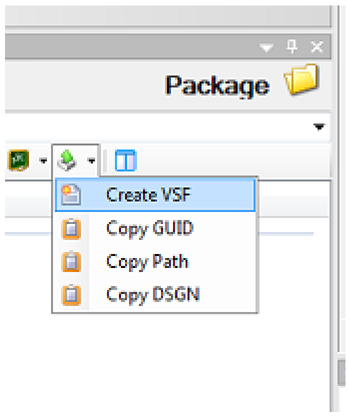


Figure 4.7: Step 2 - Selection of option from add-in

In the back end, when this option is selected, the processing begins. A new file is created with the component name that is selected and extension .vsf. After this for main export implementation, nodes are created in the newly created file. It begins with the main VSF node and properties like version, date, the process type and so on. After which, component node is created. In order to get the packages under that component, the component is searched completely for all the packages.

Once the package is found, all data is accessed within that package. As discussed before, only the artifacts that are needed are specifically searched based on kind. As and when object within our list of artifacts is found, a new node is created. All the properties related to that object are accessed and stored as prop nodes. This process is done recursively, until all the data within that component is searched and only the data that we need is added to the VSF file.

Most important are the relations node that are created. Since the naming and terms are different when it comes to EA, while searching for the relations or in this case the connectors, it is converted to the finalized names as in the meta-model. To sum it all, the table 4.1 shows the connector types and corresponsing relations.

Once these are calculated, they are then written as relations.
Finally, in order to process the diagrams. In this case, the state diagrams are processed in state table. After which, the diagrams are stored with path and appropriate name and extension. This path is saved as a property of the diagram. When the conversion is taking place, simultaneously the diagrams are drawn and saved in a folder, same as the VSF file.

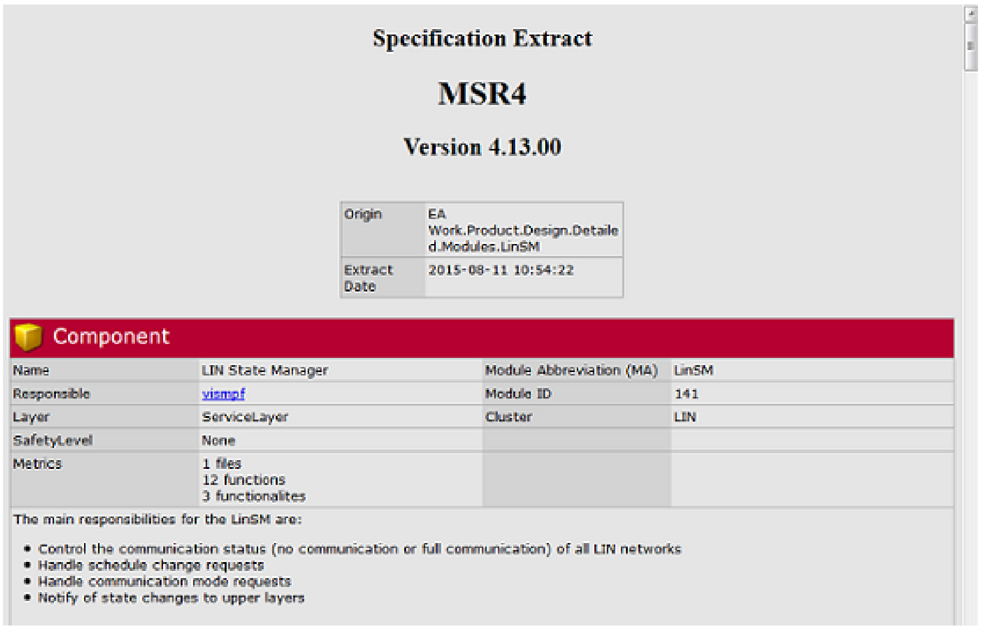
3. **Abstract description of result -**

   After the conversion or export of product architecture design into a VSF file is completed, it is used to convert it into HTML using XSLT technology. Since the VSF file is in XML technology, it can be easily converted to HTML for readability and understanding purposes. The screen shots ahead in figures 4.8, 4.9 and 4.10, give an abstract overview of

Connector types and corresponding relations	
Connector Type in EA	**Relation in VSF**
Sequence	
Generalization	Generalize/Generalizeby
Aggregation	Contain/ContainedIn
Dependency/include	Include/Includeby
StateFlow	Follow/Followby

Table 4.1: Connector types and corresponding relations

how the product architecture design is presented to be used further as an input (along with VSF file) for the component development.



Specification Extract

MSR4

Version 4.13.00

Origin	EA Work.Product.Design.Detailed.Modules.LinSM
Extract Date	2015-08-11 10:54:22

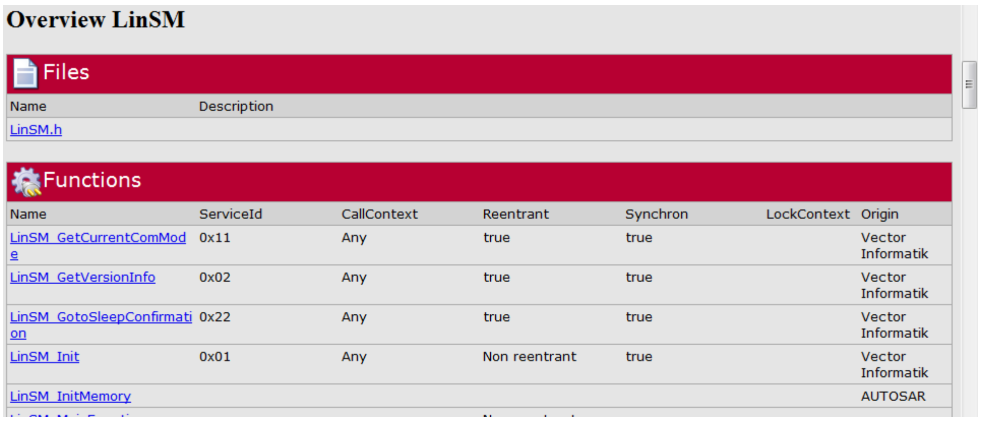
Component

Name	LIN State Manager	Module Abbreviation (MA)	LinSM
Responsible	vismpf	Module ID	141
Layer	ServiceLayer	Cluster	LIN
SafetyLevel	None		
Metrics	1 files 12 functions 3 functionalites		

The main responsibilities for the LinSM are:

- Control the communication status (no communication or full communication) of all LIN networks
- Handle schedule change requests
- Handle communication mode requests
- Notify of state changes to upper layers

Figure 4.8: Basic information given



Overview LinSM

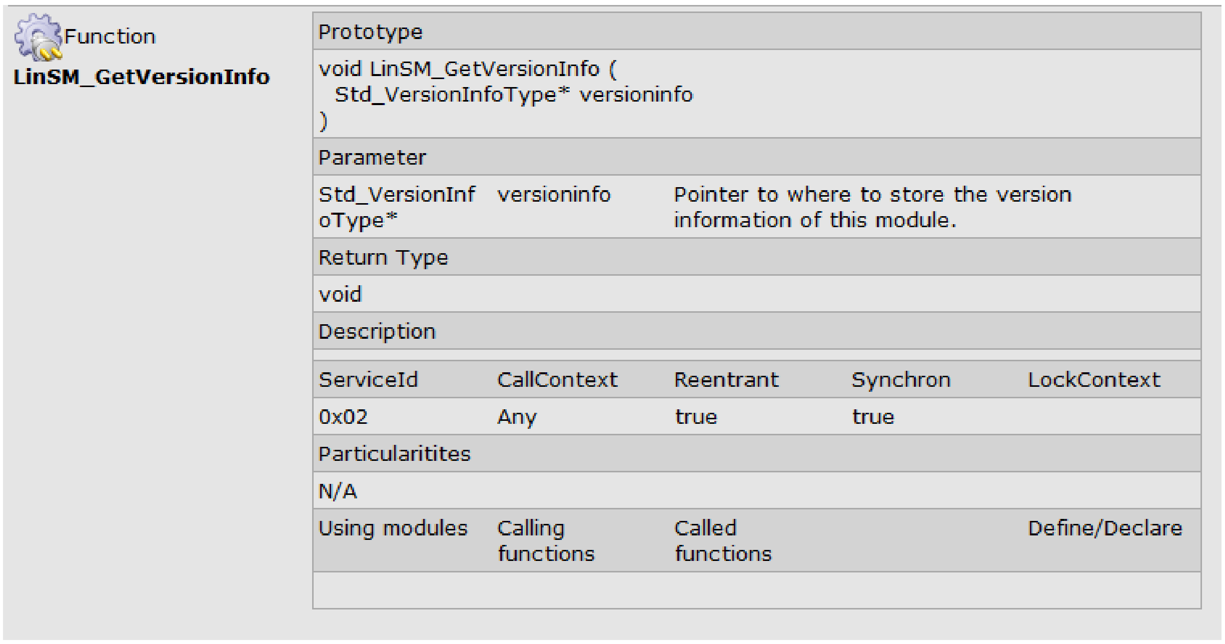
Files

Name	Description
LinSM.h	

Functions

Name	ServiceId	CallContext	Reentrant	Synchron	LockContext	Origin
LinSM_GetCurrentComMode	0x11	Any	true	true		Vector Informatik
LinSM_GetVersionInfo	0x02	Any	true	true		Vector Informatik
LinSM_GotoSleepConfirmation	0x22	Any	true	true		Vector Informatik
LinSM_Init	0x01	Any	Non reentrant	true		Vector Informatik
LinSM_InitMemory						AUTOSAR

Figure 4.9: Overview of the component



Function
LinSM_GetVersionInfo

Prototype				
void LinSM_GetVersionInfo (Std_VersionInfoType* versioninfo)				
Parameter				
Std_VersionInfoType*	versioninfo	Pointer to where to store the version information of this module.		
Return Type				
void				
Description				
ServiceId	CallContext	Reentrant	Synchron	LockContext
0x02	Any	true	true	
Particularitites				
N/A				
Using modules	Calling functions	Called functions		Define/Declare

Figure 4.10: Example of function details

As seen in the figures 4.8, 4.9 and 4.10, all the data that is required for further development is present. The component developers mentioned that the data was needed only for reference and understanding the requirements. That purpose has been taken care of using this.

This completed the first half of the round-trip engineering for the data artifacts. But, that does not complete the process.

4.2.2 Converting UDB to VSF

In this section, all the data that comes from source code files is explicitly discussed. In this case data from UDB and which data is actually used while converting to VSF. This section is divided into 4 parts. They are:

1. General details
2. Implementation details
3. Challenges faced
4. Abstract description of result

The basic work flow of data can be understood by the figure 4.11

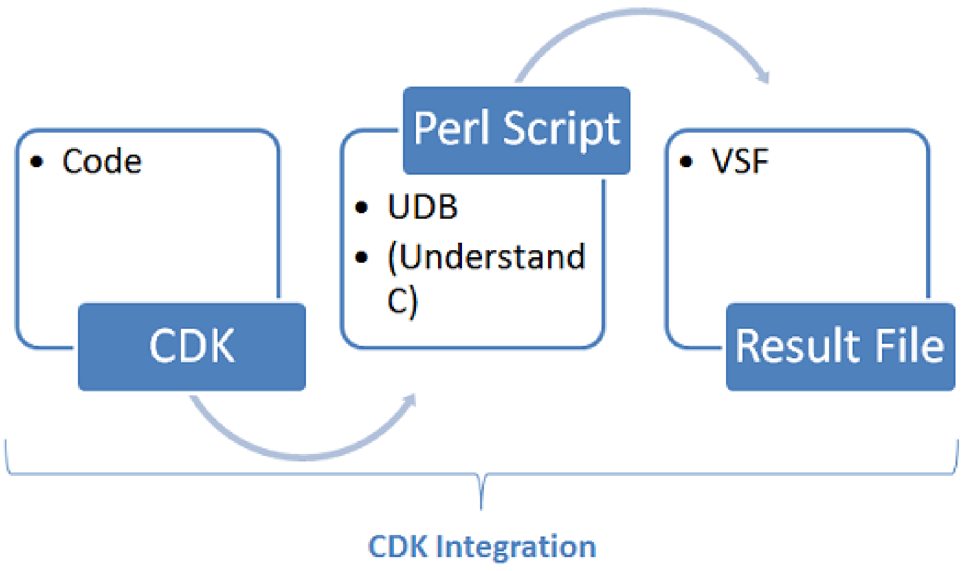


Figure 4.11: Work flow model for conversion from UDB to VSF

1. **General details -**

   In this section, the data artifacts that have to be exported from the source files for VSF are discussed. When data comes from the lower level of the V-model! (**V-model!**), that is from the component development level, there is a lot more data than what was given from architecture level. It is important to set rules and analyse which data artifacts are required in this case. Since, the final goal is to verify the data and merge it with data that comes from architecture, it is very essential to maintain the same level of hierarchy and naming conventions. This was also followed during the export in previous section.
   To get a clear picture of what is to be used and what not, the figure 4.12 is a good reference.

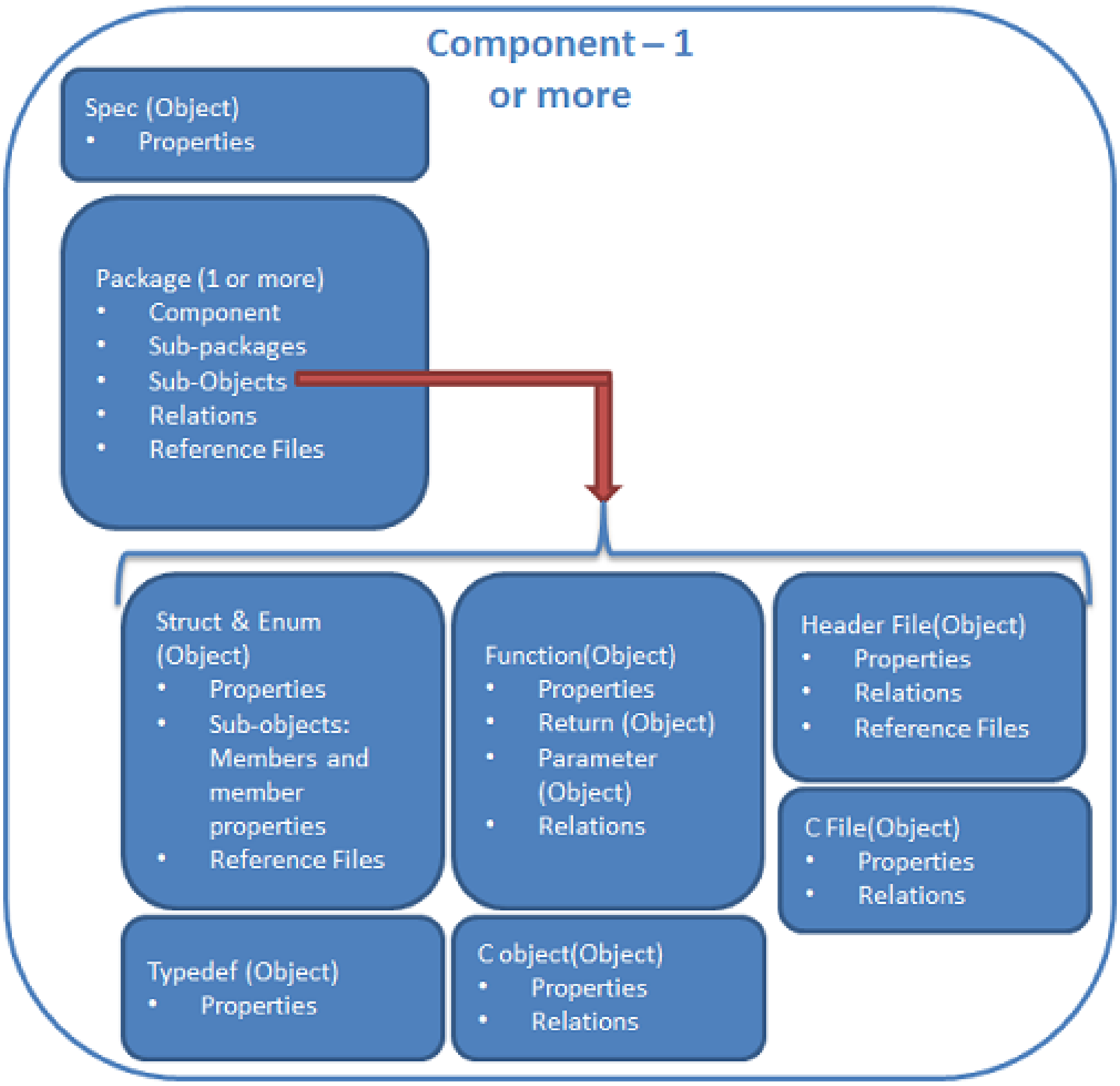


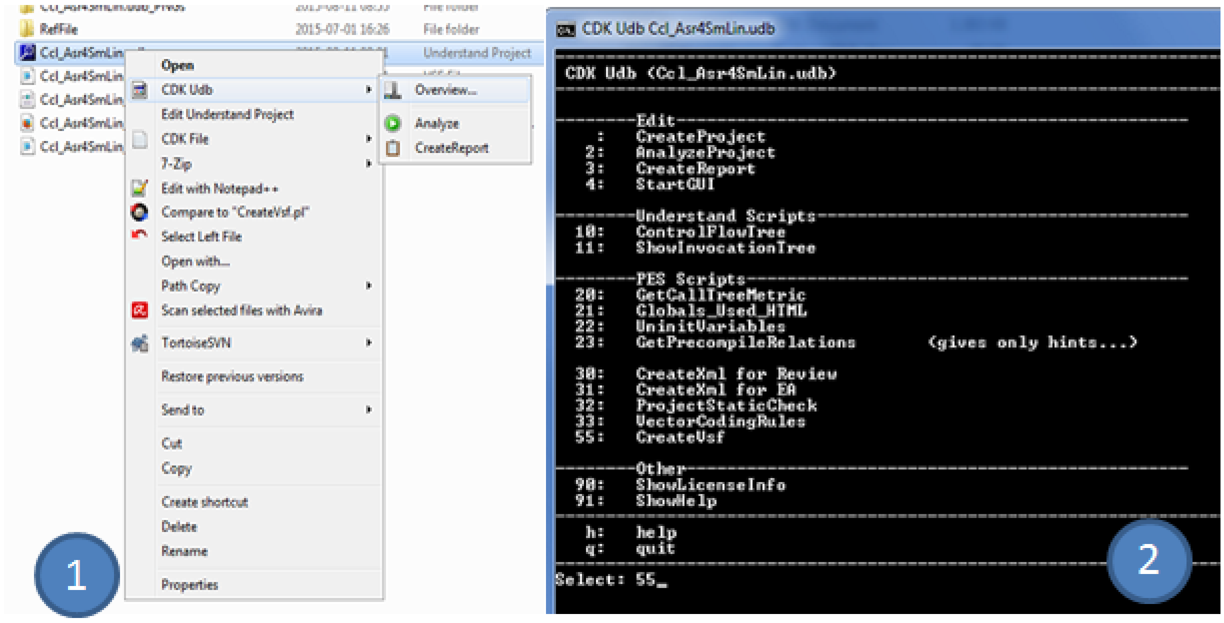
Figure 4.12: Data artifacts exported from source code

As noticed in figure 4.12, there is relatively more data than what was given before as input. The additional artifacts such as State and Trigger are not available from source code level. C objects are explicitly a part of the model, along with type definitions, structures and enumerations. Each artifact has its attributes and relation with one another specified. And each of these artifacts are contained in each component. When data comes from source code, it is possible to have more than one component, since the interaction or connection with other components can be easily represented.

2. **Implementation details** -

   This was done using 'Perl' scripting. Reason perl was opted was because the database that is used during implementation is UC. The compatibility of using and extracting data from UDB is most efficient with this technology. Hence, a script was written to extract data from UDB and write it on a VSF file.

   To begin with, a UDB project is selected to be analysed. The implementation has been added as a part of the working environment CDK at Vector. A glimpse of how it begins can be seen in the figure 4.13. In screen shot 1, the overview option in CDK can be selected. In screen shot 2, the options from CDK open in a command line interface. The option to convert into VSF can be selected.



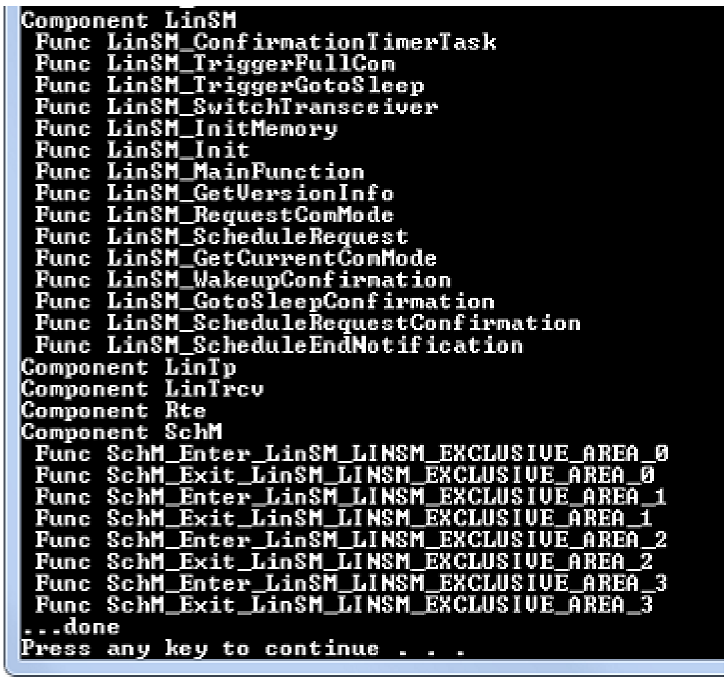
```
--------------------------------------------------------------------------------
 CDK Udb (Ccl_Asr4SmLin.udb)
--------------------------------------------------------------------------------

----------Edit---------------------------------------------------------------
    :    CreateProject
   2:    AnalyzeProject
   3:    CreateReport
   4:    StartGUI

----------Understand Scripts-------------------------------------------------
  10:    ControlFlowTree
  11:    ShowInvocationTree

----------PES Scripts--------------------------------------------------------
  20:    GetCallTreeMetric
  21:    Globals_Used_HTML
  22:    UninitVariables
  23:    GetPrecompileRelations        (gives only hints...)

  30:    CreateXml for Review
  31:    CreateXml for EA
  32:    ProjectStaticCheck
  33:    VectorCodingRules
  55:    CreateVsf

----------Other--------------------------------------------------------------
  90:    ShowLicenseInfo
  91:    ShowHelp
--------------------------------------------------------------------------------
   h:    help
   q:    quit
--------------------------------------------------------------------------------
Select: 55_
```

Figure 4.13: Screen shot 1 2 - Implementation of UDB to VSF

During the processing, all the components, files and functions under consideration in the database can be seen. The output was taken mainly for reading and following the implementation purpose. It is seen in figure 4.14.

During the processing, at the back end, the perl script is being implemented. The structure of the script is now explained.

The script begins with opening the database. If the database is not found, it reports an error. Then, it searches for a VSF file with same name as the UDB project. If any such file is not found, it creates a new file with .vsf extension, that is named after the project. Basic information such as the author, date, version and so on are written on the file. The entire script is divided into functions. This makes it easier to call them any number of times, and



```
Component LinSM
 Func LinSM_ConfirmationTimerTask
 Func LinSM_TriggerFullCom
 Func LinSM_TriggerGotoSleep
 Func LinSM_SwitchTransceiver
 Func LinSM_InitMemory
 Func LinSM_Init
 Func LinSM_MainFunction
 Func LinSM_GetVersionInfo
 Func LinSM_RequestComMode
 Func LinSM_ScheduleRequest
 Func LinSM_GetCurrentComMode
 Func LinSM_WakeupConfirmation
 Func LinSM_GotoSleepConfirmation
 Func LinSM_ScheduleRequestConfirmation
 Func LinSM_ScheduleEndNotification
Component LinTp
Component LinTrcv
Component Rte
Component SchM
 Func SchM_Enter_LinSM_LINSM_EXCLUSIVE_AREA_0
 Func SchM_Exit_LinSM_LINSM_EXCLUSIVE_AREA_0
 Func SchM_Enter_LinSM_LINSM_EXCLUSIVE_AREA_1
 Func SchM_Exit_LinSM_LINSM_EXCLUSIVE_AREA_1
 Func SchM_Enter_LinSM_LINSM_EXCLUSIVE_AREA_2
 Func SchM_Exit_LinSM_LINSM_EXCLUSIVE_AREA_2
 Func SchM_Enter_LinSM_LINSM_EXCLUSIVE_AREA_3
 Func SchM_Exit_LinSM_LINSM_EXCLUSIVE_AREA_3
...done
Press any key to continue . . .
```

Figure 4.14: Screen shot 3 - During the processing

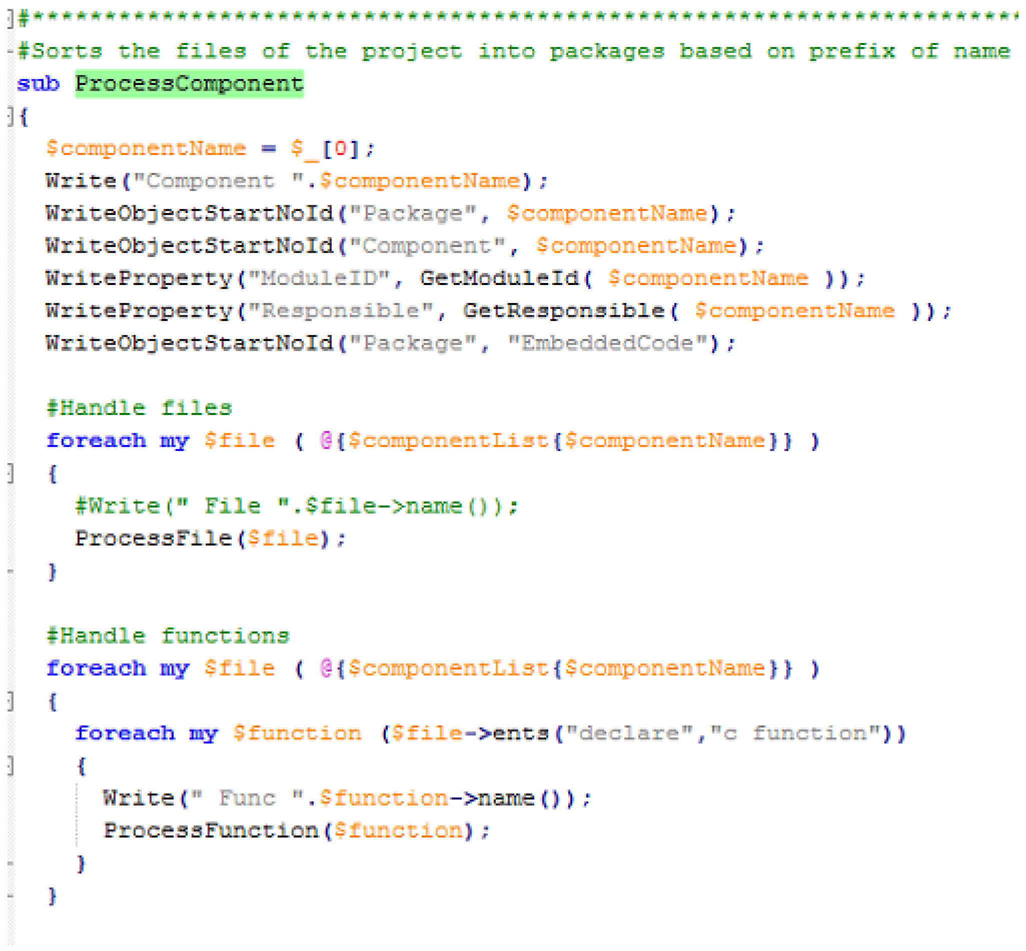
also for scalability purposes. In case of any addition/deletion, it makes it really handy to work with functions.

UDB provides with options to directly access certain objects such as functions, files, type definitions, structures, enumerators and C objects. Mainly, it can give information that is directly available within the code such as names, types, kinds, etc. So that is taken care of by accessing the entities of database directly. Also, unique identification codes are generated for each artifact. This is used for making the internal reference smooth.

It starts with separating all these entities based on which component it is contained under. For that, the component name has to be realized. Various components are grouped by their names and all the entities related only to each of them are written on the file accordingly. Screen shot 4.15 give a little preview of part of the code where only the files and functions are being handled within a component. Similarly, other entities such as type definitions, structures, etc. are handled.that is responsible for getting component name and processing of each component.

Screen shot 4.15 shows that the components are first separated into a list. It is identified based on the component name. Each file is analysed within that component and processed. This is done by calling the 'ProcessFile' function, with file entity sent as an argument. Similarly, the process repeats for function and all other entities within that component.

All this was elementary and directly accessible, the problems arose when the elements that were part of the doxygen comment had to be accessed and written in the file as various properties. This is discussed at length in the next section.



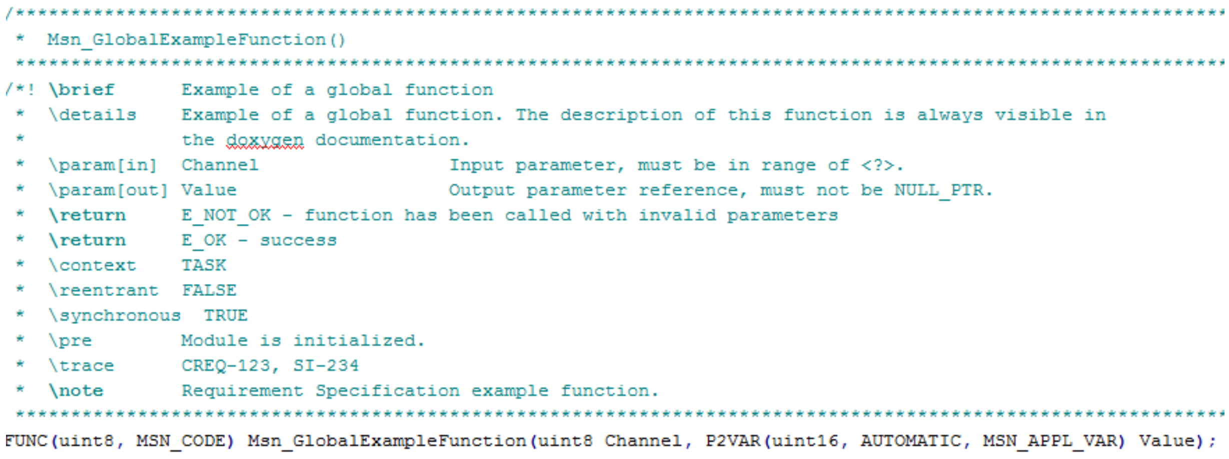
```
#*******************************************************************
#Sorts the files of the project into packages based on prefix of name
sub ProcessComponent
{
  $componentName = $_[0];
  Write("Component ".$componentName);
  WriteObjectStartNoId("Package", $componentName);
  WriteObjectStartNoId("Component", $componentName);
  WriteProperty("ModuleID", GetModuleId( $componentName ));
  WriteProperty("Responsible", GetResponsible( $componentName ));
  WriteObjectStartNoId("Package", "EmbeddedCode");

  #Handle files
  foreach my $file ( @{$componentList{$componentName}} )
  {
    #Write(" File ".$file->name());
    ProcessFile($file);
  }

  #Handle functions
  foreach my $file ( @{$componentList{$componentName}} )
  {
    foreach my $function ($file->ents("declare","c function"))
    {
      Write(" Func ".$function->name());
      ProcessFunction($function);
    }
  }
```

Figure 4.15: Code Part 1 - Processing a component

3. **Challenges faced -**

   This section explores the challenges that were faced and how they were handled. As identified earlier, it is easy to access data which are directly a part of the code. But data from the comments section are also needed. UC does give us options to access the comments, but the doxygen comment is divided into many parts. The figure 4.16 presents a small example of this. Some of the individual problems are discussed as follows:

   - In the example given in 4.16 the extraction of values of the properties is needed. Each one has a specific way of being written, that is, each one is written with ’
     ’ in front of it. So first, all the comments that are found above or before a definition are accessed and then split into multiple lines.
   - Once the comment is split, a match is done against every starting tag. But still there are many issues. For tags which are present only once, it is easy to extract their value. But, in order to get values of tags which occur multiple times, it is important to store them in the right order. Let us consider the example of multiple parameters in this case. When the parameter nodes are written in the VSF file, properties such as



```
/*****************************************************************************************************
 *  Msn_GlobalExampleFunction()
 *****************************************************************************************************
/*! \brief      Example of a global function
 *  \details    Example of a global function. The description of this function is always visible in
 *              the doxygen documentation.
 *  \param[in]  Channel                Input parameter, must be in range of <?>.
 *  \param[out] Value                  Output parameter reference, must not be NULL_PTR.
 *  \return     E_NOT_OK - function has been called with invalid parameters
 *  \return     E_OK - success
 *  \context    TASK
 *  \reentrant  FALSE
 *  \synchronous  TRUE
 *  \pre        Module is initialized.
 *  \trace      CREQ-123, SI-234
 *  \note       Requirement Specification example function.
 *****************************************************************************************************
FUNC(uint8, MSN_CODE) Msn_GlobalExampleFunction(uint8 Channel, P2VAR(uint16, AUTOMATIC, MSN_APPL_VAR) Value);
```

Figure 4.16: Code Part 2 - Example of doxygen comment

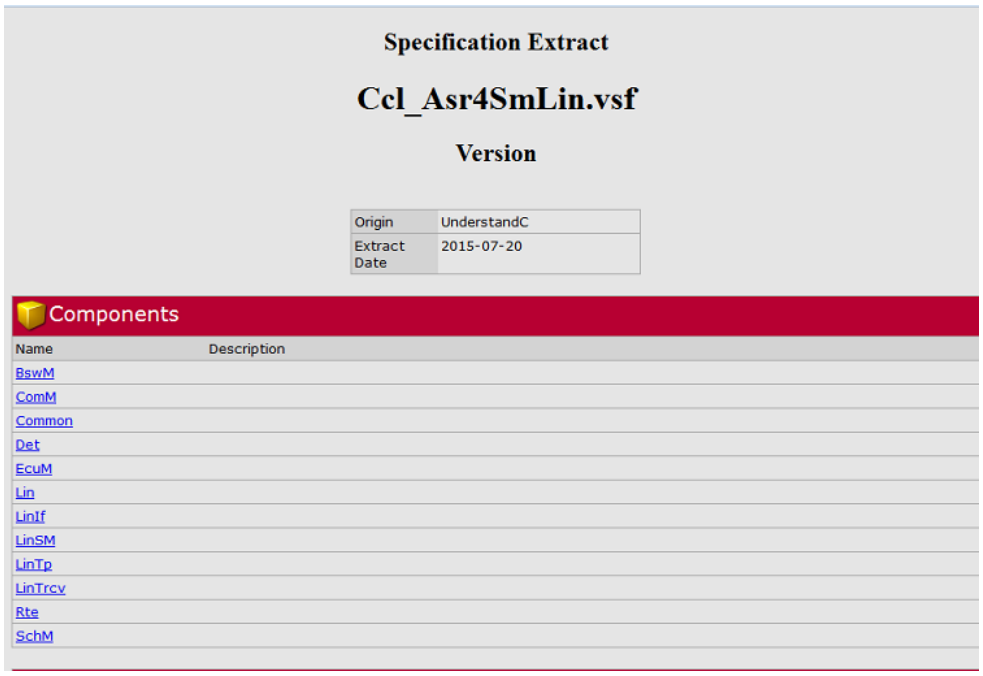
its description is also written. In order for that to happen, another split is required on the already split doxygen comment.For example, this helps in retrieving the correct value based on name.

- This can be done by using hash mapping technique. In this scenario, it is used to store the values of parameter along with its properties in the right order, based on a unique key for each entry. Once this technique is clear, same concept for all the tags that occur multiple times is used.
- Next problem was renaming the function scope. Or rather, finding the scope and assigning the value based on 'kind name'. If the entity is of the kind static or private, the scope is assigned as 'local'. And if it is not specified, then it is assigned as 'global'. This is done for all the functions and variables.
- Sometimes, the values of 2 tags in order is needed. This needs to be assigned to one tag in the file and write it in result file. This can be referred to extracting values of 'brief' and 'details', which combine to give the value of 'desc' or description.
- After solving all the issues, the alignment of nodes was also very important. In order for better readability, the structure of VSF file is important. It makes it simpler to look for objects since the hierarchy is defined.

4. **Abstract description of result -**

Once all the data is written in the VSF file, like used in previous sections, the result file is converted to HTML format. The same XSLT script is used for the conversion. As noticed, there is lot more data when it comes from source code, so the screen shots 4.17 , 4.18 and 4.19 display some parts as example for the final result.

Screen shot 4.17 shows that the look and feel of the HTML remains the same, with origin as UC. It lists all the components in the project. When any of the component is clicked,



Specification Extract

Ccl_Asr4SmLin.vsf

Version

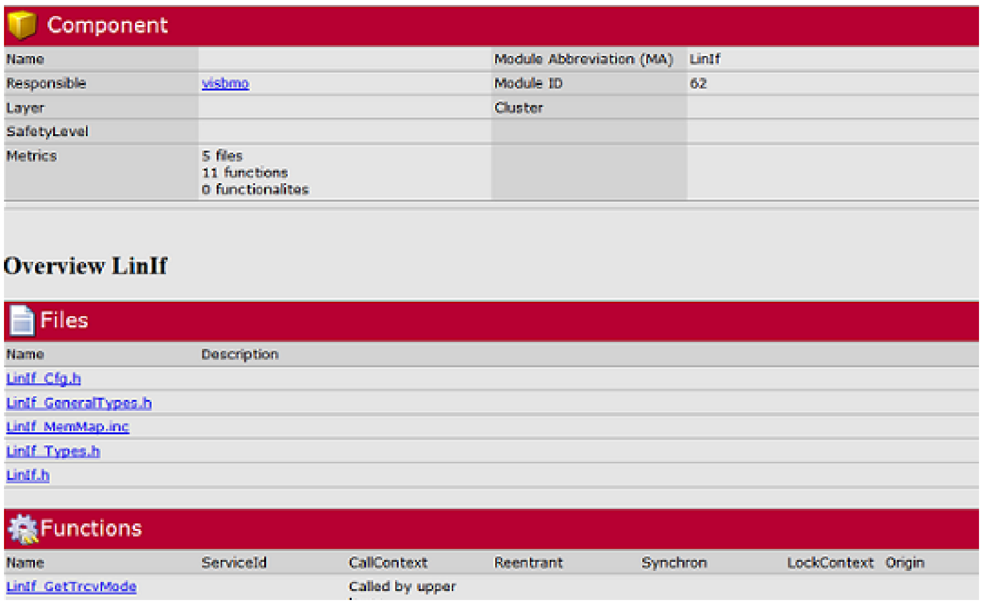
Origin	UnderstandC
Extract Date	2015-07-20

Components

Name	Description
BswM	
ComM	
Common	
Det	
EcuM	
Lin	
LinIf	
LinSM	
LinTp	
LinTrcv	
Rte	
SchM	

Figure 4.17: Basic overview of output file

the control is taken to that component. There is only one main component that is being analysed, so if any other component is clicked, it gives only the related information with the component under analysis. That is seen in screen shot 4.18. In screen shot 4.19 one function as an example is seen. It contains a lot more information about it since, it comes from the developed component itself.



Component

Name		Module Abbreviation (MA)	LinIf
Responsible	vishmo	Module ID	62
Layer		Cluster	
SafetyLevel			
Metrics	5 files 11 functions 0 functionalites		

Overview LinIf

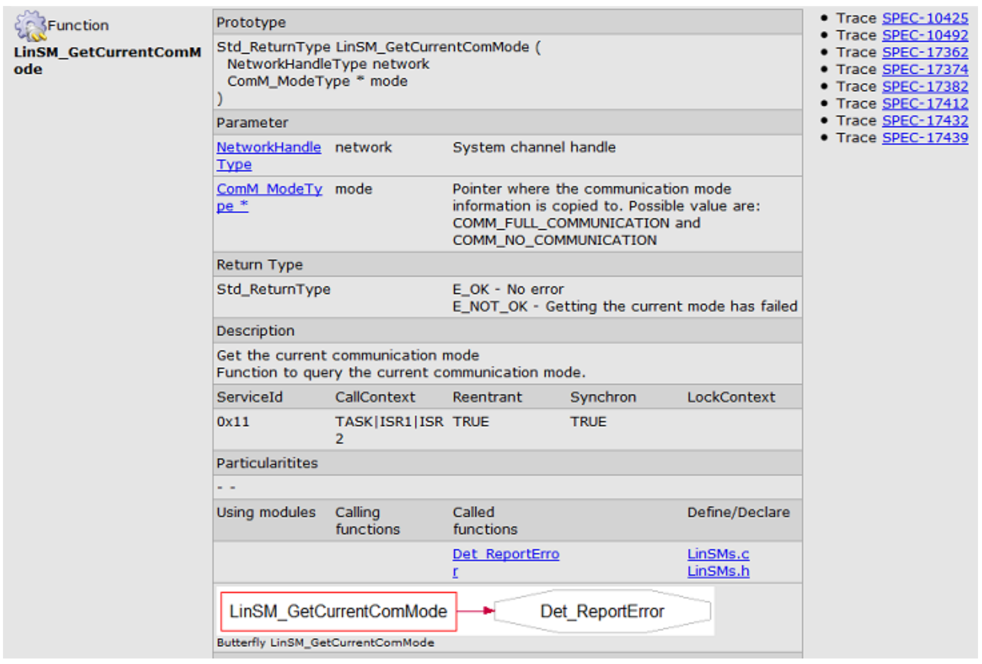
Files

Name	Description
LinIf_Cfg.h	
LinIf_GeneralTypes.h	
LinIf_MemMap.inc	
LinIf_Types.h	
LinIf.h	

Functions

Name	ServiceId	CallContext	Reentrant	Synchron	LockContext	Origin
LinIf_GetTrcvMode		Called by upper				

Figure 4.18: Overview of selected component



Function
LinSM_GetCurrentComMode

Prototype				
Std_ReturnType LinSM_GetCurrentComMode (NetworkHandleType network ComM_ModeType * mode)				
Parameter				
NetworkHandleType	network	System channel handle		
ComM_ModeType *	mode	Pointer where the communication mode information is copied to. Possible value are: COMM_FULL_COMMUNICATION and COMM_NO_COMMUNICATION		
Return Type				
Std_ReturnType		E_OK - No error E_NOT_OK - Getting the current mode has failed		
Description				
Get the current communication mode Function to query the current communication mode.				
ServiceId	CallContext	Reentrant	Synchron	LockContext
0x11	TASK|ISR1|ISR2	TRUE	TRUE	
Particularitites				
- -				
Using modules	Calling functions	Called functions		Define/Declare
		Det_ReportError		LinSMs.c LinSMs.h

Butterfly LinSM_GetCurrentComMode

- Trace SPEC-10425
- Trace SPEC-10492
- Trace SPEC-17362
- Trace SPEC-17374
- Trace SPEC-17382
- Trace SPEC-17412
- Trace SPEC-17432
- Trace SPEC-17439

Figure 4.19: Example of a selected function

This completes the second half of the round-trip engineering. The combination and tooling of using these created files is explained in the next section.

4.3 Comparing and merging data

In this section, the different mechanism for the data is discussed. This includes comparison of data and merging of data. The section is sub divided into 3 parts:

- Diff tool
- Merge tool
- Implementation

4.3.1 Diff tool

In this section, we discuss at length about the concept of the ’diff tool’ (name given for purpose of ease) or in formal terms the verification tool.
Since we have the same platform, i.e, same file format, a tool was implemented to take these files as input and have a comparison. We know that file has been formulated based on the data, structure and process and hence, it is not another file comparison. This comparison takes place

based on the component name. So, these are the 4 arguments that are given as input for the implementation. Before going further, let us look at the activity diagram 4.20 which would help us the steps involved distinctly. And the explanation to each step follows after.

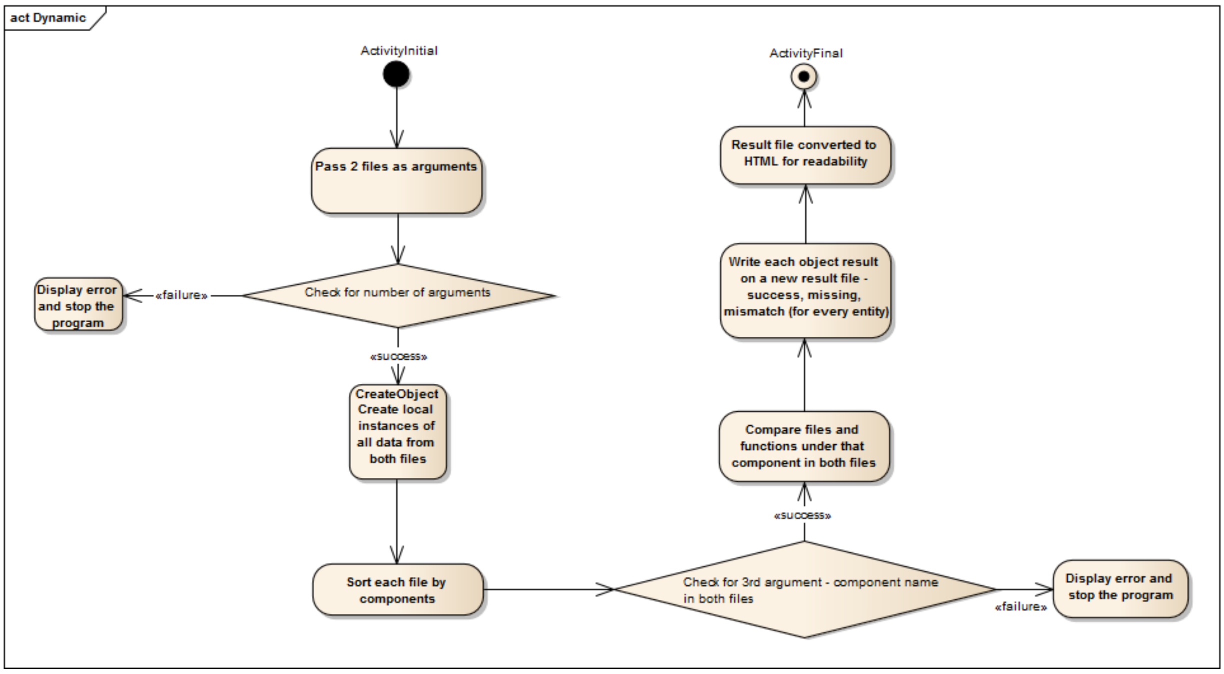

act Dynamic
ActivityInitial
Pass 2 files as arguments
Check for number of arguments
«failure»
Display error and stop the program
«success»
CreateObject Create local instances of all data from both files
Sort each file by components
Check for 3rd argument - component name in both files
«failure»
Display error and stop the program
«success»
Compare files and functions under that component in both files
Write each object result on a new result file - success, missing, mismatch (for every entity)
Result file converted to HTML for readability
ActivityFinal

Figure 4.20: Activity Diagram 1 - Process involved in Diff tool

Let us see the steps that are taken in diagram 4.20:

- It begins with passing of 4 arguments. The first 2 are the VSf files. One comes from architecture and other from component developed. Third argument is type of mechanism called. In this case, Diff1 or Diff2. The difference is explained in next section under implementation. Fourth argument is the component name that it has to verify.
- If all these conditions meet, the process continues. Else, there is an error given and process is stopped.
- Local instances of the file data are created.
- Each file data is sorted based on components
- The component name given as argument is searched in the file data.
- If not found, error is displayed and process is stopped. Else, it goes to next step.
- Each file and function under that component is compared.
- For each object, the result is written in the new result file. The result is mentioned along with the object that is success, missing or mismatch.

- Result file can be converted to HTML for readability purpose, since the result file is also a VSF file format.

The key ideas behind the tooling are as follows:

- Collect the data from both the files
- Group the data based on a key
- Each object had a unique comparison key. This key was created based on the table that is shown.
- There were 3 exceptions to the comparison key. One was for parameter, then the return object and finally the relations. These objects were handles separately.

The comparison key for all the objects was 'Kind' + 'Name'. This formed a unique combination for all the objects. But there were exceptions to this rule. They can be seen in table 4.2.

Comparison key for exceptions			
	Artifact	**Comparison key**	**Description**
(1)	Parameter	Base object comparison key + Name	Since same parameter can occur in various functions, it is necessary to check the comparison key of it's parent object. In this case the comparison key of the base function is also considered.
(2)	Return	Base object comparison key + Name + Type	Since return can occur in various functions, it is necessary to check the comparison key of it's parent object. In this case the comparison key of the base function is also considered.
(3)	Relations (Kind1)	Kind + ObjKind + ObjId	Combination of kind of the relation, kind of the related object along with the id. This is for the related objects outside the component.
(4)	Relations (Kind2)	Kind + ObjKind + ComparisonKey	Combination of kind of the relation, kind of the related object along with the comparison key of the related object. This is for the related objects within the component.

Table 4.2: Comparison keys for exceptions

4.3.2 Merge tool

In this section, the tool that is implemented for the purpose of merging data and updating it back to architecture is explained.
This tool has more than one purpose. Firstly, the data can be merged from both ends, that is architecture and component developed. After which, if there are multiple versions of component source code, then even that can be merged. Based on some changes, it is possible to set the base file as new version and the merged file will be up to date version of the file. The activity diagram 4.21 plainly gives us an overview of the process involved. The process is explained extensively after the diagram.

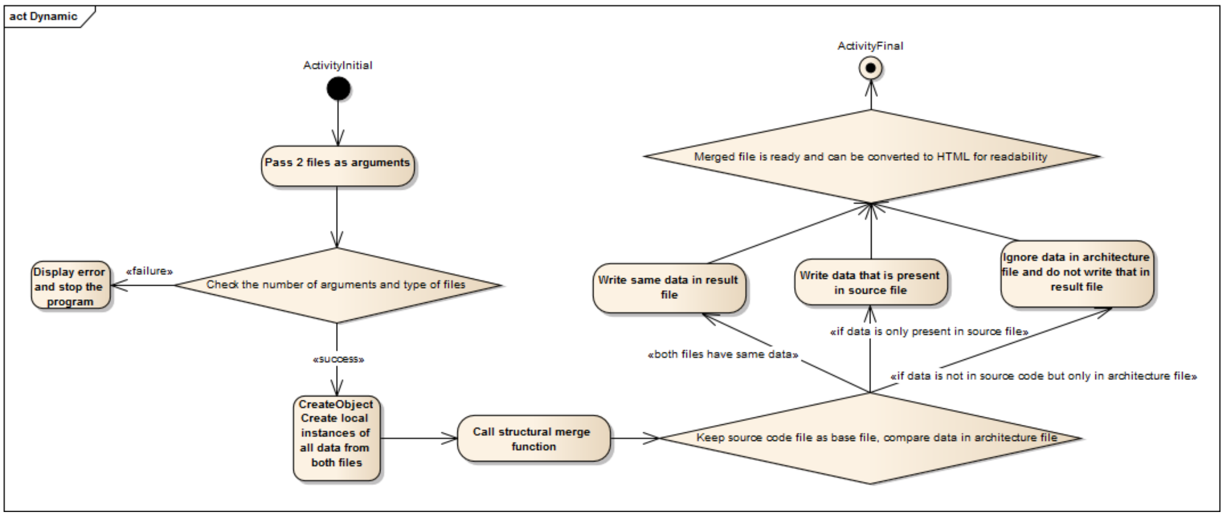

act Dynamic
ActivityInitial
Pass 2 files as arguments
Check the number of arguments and type of files
«failure»
Display error and stop the program
«success»
CreateObject Create local instances of all data from both files
Call structural merge function
Keep source code file as base file, compare data in architecture file
«both files have same data»
«if data is only present in source file»
«if data is not in source code but only in architecture file»
Write same data in result file
Write data that is present in source file
Ignore data in architecture file and do not write that in result file
Merged file is ready and can be converted to HTML for readability
ActivityFinal

Figure 4.21: Activity Diagram 2 - Process involved in Merge tool

To understand the tool better in 4.21, following were the steps involved:

- It begins with passing of 3 arguments. They are the **VSf!** (**VSf!**) files. These files can be from 2 different ends of development or also, from the same end. For updating into new version purposes. The third argument states the type of mechanism and in this case, SpecMerge1 or SpecMerge2. The difference between these 2 mechanisms is explained in the implementation section below, in detail.
- If all these conditions meet, the process continues. Else, there is an error given and process is stopped.
- Local instances of the file data are created.
- Structural merge function is called.
- In this case, source file is kept as base file and comparison begins.
- There are 3 possibilities.
    1. When both files have same data, write that data into result file

   2. If data is present only in source file, write that data into result file
   3. If data is not present in source file, but only in architecture file, then ignore.

- Merged result file is ready and can be converted to HTML

Merge tool also uses the concept of comparison keys, as discussed in section 4.3.1.

4.3.3 Implementation

In this section, the implementation aspects of the tooling is discussed. Now that the steps involved in each tooling are clear (in sections 4.3.1 4.3.2), more detailed aspect of each and final output is explained.
This section is divided into the following parts:

1. General details
2. Implementation details
3. Abstract description of result

1. **General details** -

   The implementation of this tooling was done using "Microsoft Visual C# 2010 Express" and using clearly Csh technology. For the ease of understanding, 2 simple figures 4.22 and fig:fig51-MergeTool have been created, which provides the idea of the program without any ambiguity. As seen in 4.22 and fig:fig51-MergeTool , the input and output is shown.

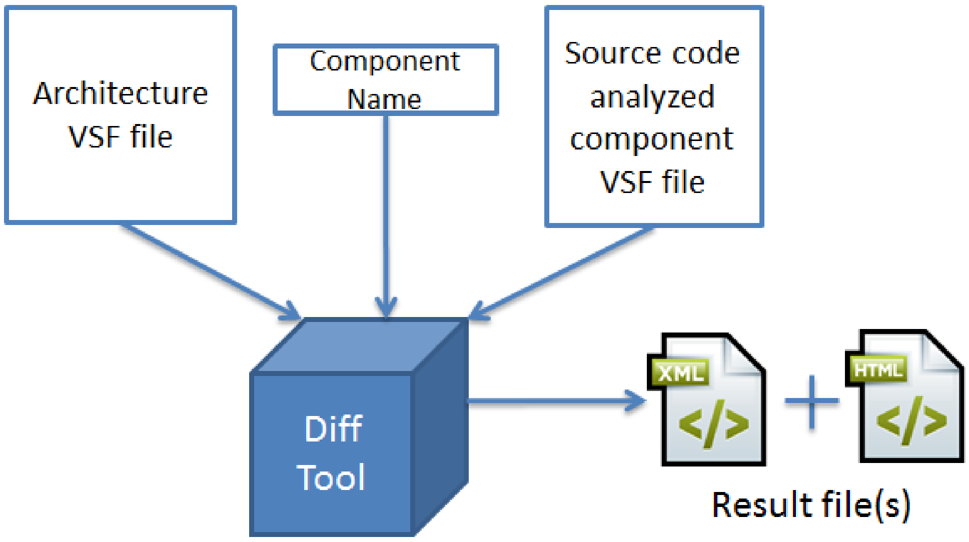


Figure 4.22: Diff tool details

   The box with tool written on it is the internal functionality and implementation part of it.

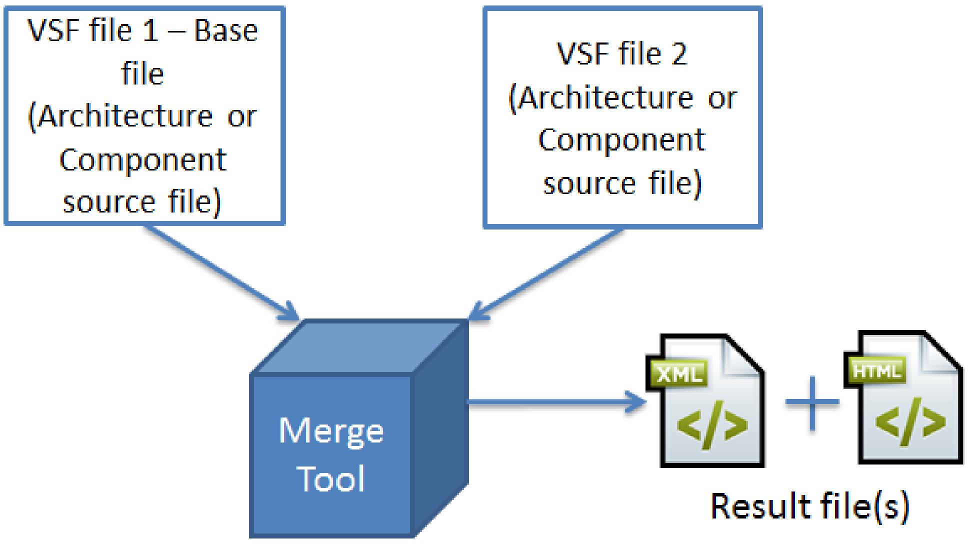

VSF file 1 – Base file (Architecture or Component source file)
VSF file 2 (Architecture or Component source file)
Merge Tool
XML
HTML
Result file(s)

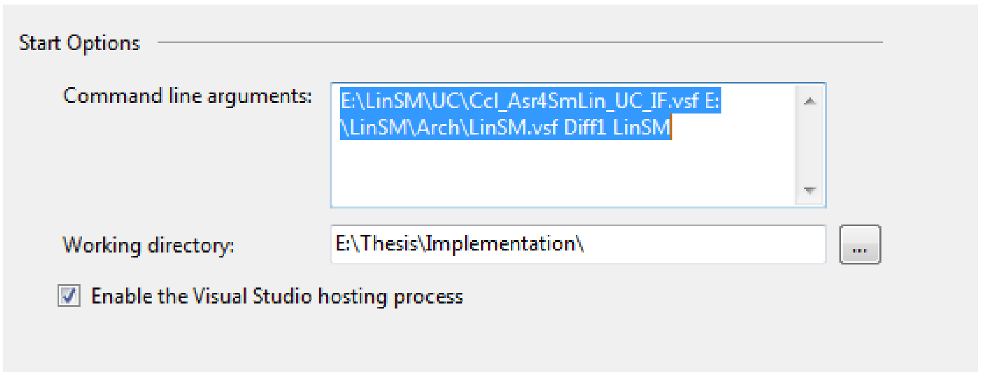
Figure 4.23: Merge tool details

2. **Implementation details -**

   Here, the actual implementation of the tooling in the form of screen shots is seen. Firstly, the mechanism of each of these tools are divided into 2 parts - Diff1, Diff2 and Merge1, Merge2.
   The main difference between these was the input files given. In Diff1 and Merge1, the input was product architecture design file and source code file. For Diff2 and Merge2, component architecture design file was given as input along with the source code file.

   When the objects was matched, there are different properties that are checked for in these 2 Diff tools. This can be better explained from the table 4.3.

   The implementation begins by passing the arguments. This can be seen in screen shot 4.24.


Start Options

Command line arguments: E:\LinSM\UC\Ccl_Asr4SmLin_UC_IF.vsf E:\LinSM\Arch\LinSM.vsf Diff1 LinSM

Working directory: E:\Thesis\Implementation\

☑ Enable the Visual Studio hosting process

Figure 4.24: Tool screen shot 1 - Passing the arguments

Different aspects checked for objects		
Object	**Diff1**	**Diff2**
Component Properties	Responsible	Responsible
File Properties	Desc	Desc
Function Properties	Desc, Reentrant, Synchron	Desc
Return Properties	Type	Desc
Parameter Properties	Type, Desc	Desc
Relation Properties	(none)	Kind, ObjKind, ObjName
File Relations	(none)	Include
Function Relations	Definein	Definein

Table 4.3: Different aspects checked for objects in Diff1 and Diff2

For a more detailed idea on how this works, each component, that is each data artifact was divided into individual classes and functions. This was mainly making the usage of artifacts throughout the program very easy. Each of the classes had their specific properties and remaining common properties was inherited based on one main class, that is class object. As seen before, each object has its name and kind. Those can be taken as common properties. But, in case of 'Property' itself, kind is not considered. Only the name property is inherited. This is discussed in previous section 4.3.1.

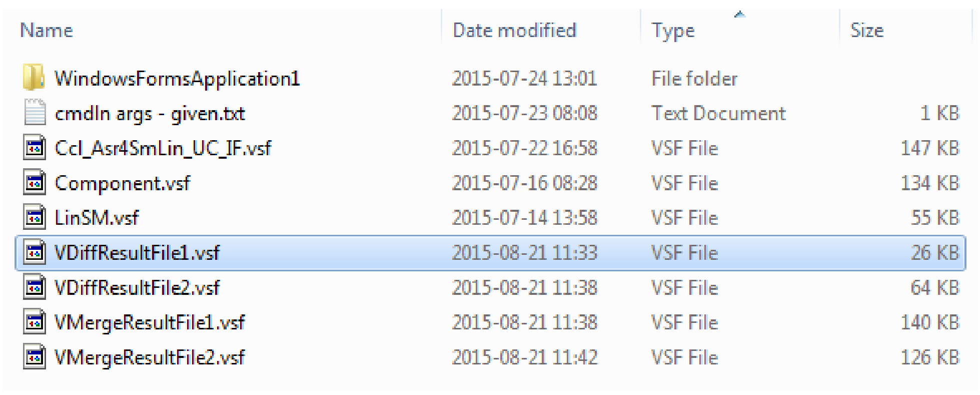
By having separate modules, it makes the program very easy to scale. In case of additional functionalities, only a function needs to be added to that specific class.

Once, the implementation is completed, and the results are ready, it can be seen in the same working directory.

3. **Abstract description of result -**

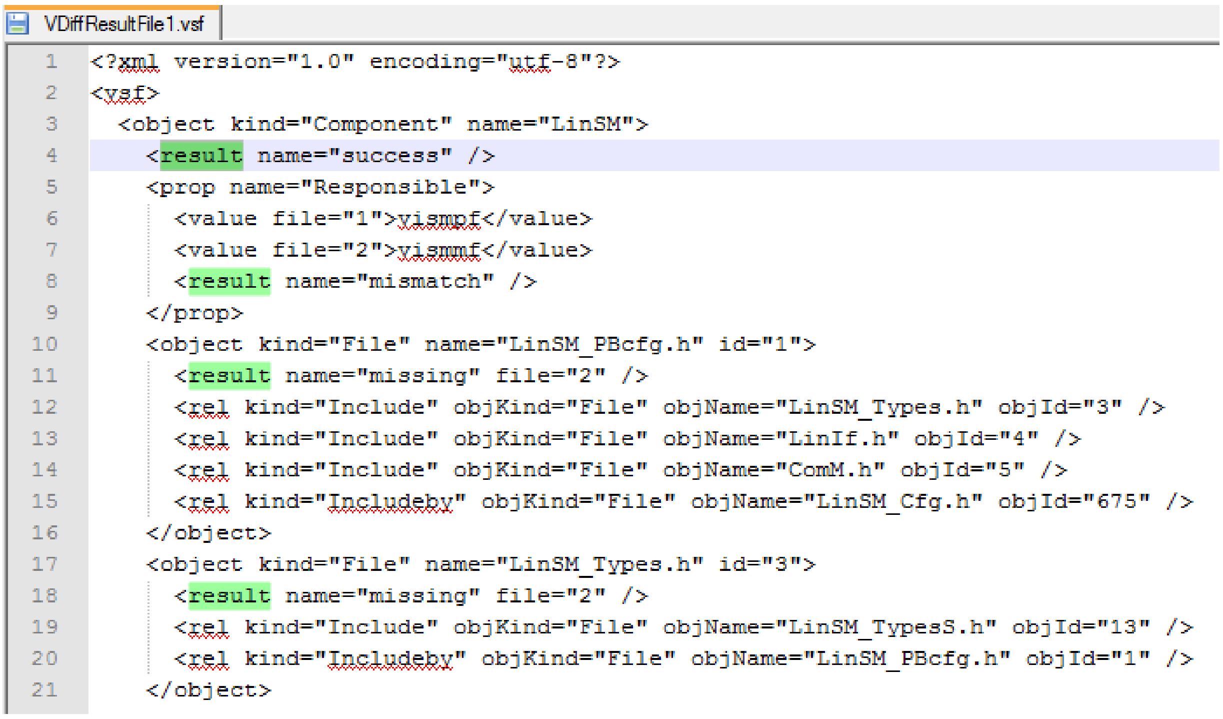
When the implementation is complete, a new file is created in the working folder, with the name 'VDiffResultFile.vsf'. This is shown in 4.25.

To get a look into how the result file looks, 4.26 shows that it is the same VSF file format. But, we can notice the extra 'result' node in the VSF file. With slight changes in the XSLT file, that helps in transformation into HTML, it is possible to convert the VSF file into a better readable format. In that, each result kind would have a different colour code for being able to easily get an overview.



Name	Date modified	Type	Size
WindowsFormsApplication1	2015-07-24 13:01	File folder	
cmdln args - given.txt	2015-07-23 08:08	Text Document	1 KB
Ccl_Asr4SmLin_UC_IF.vsf	2015-07-22 16:58	VSF File	147 KB
Component.vsf	2015-07-16 08:28	VSF File	134 KB
LinSM.vsf	2015-07-14 13:58	VSF File	55 KB
VDiffResultFile1.vsf	2015-08-21 11:33	VSF File	26 KB
VDiffResultFile2.vsf	2015-08-21 11:38	VSF File	64 KB
VMergeResultFile1.vsf	2015-08-21 11:38	VSF File	140 KB
VMergeResultFile2.vsf	2015-08-21 11:42	VSF File	126 KB

Figure 4.25: Tool screen shot 2 - Diff result file created in working folder

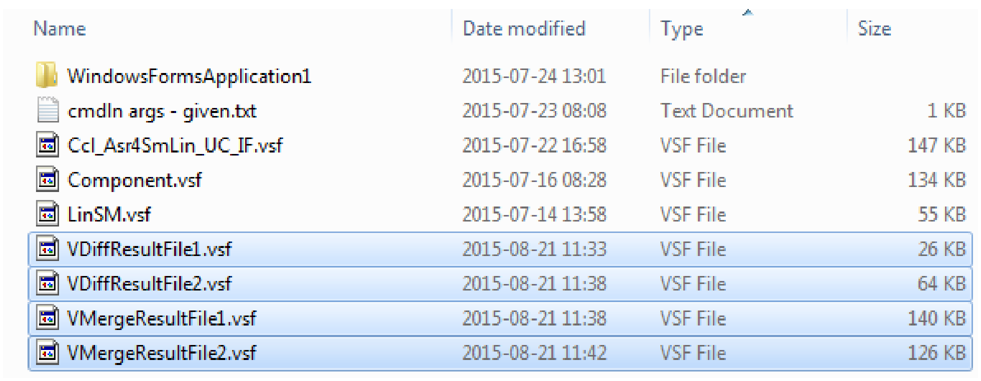


VDiffResultFile1.vsf

```
<?xml version="1.0" encoding="utf-8"?>
<vsf>
  <object kind="Component" name="LinSM">
    <result name="success" />
    <prop name="Responsible">
      <value file="1">vismpf</value>
      <value file="2">vismmf</value>
      <result name="mismatch" />
    </prop>
    <object kind="File" name="LinSM_PBcfg.h" id="1">
      <result name="missing" file="2" />
      <rel kind="Include" objKind="File" objName="LinSM_Types.h" objId="3" />
      <rel kind="Include" objKind="File" objName="LinIf.h" objId="4" />
      <rel kind="Include" objKind="File" objName="ComM.h" objId="5" />
      <rel kind="Includeby" objKind="File" objName="LinSM_Cfg.h" objId="675" />
    </object>
    <object kind="File" name="LinSM_Types.h" id="3">
      <result name="missing" file="2" />
      <rel kind="Include" objKind="File" objName="LinSM_TypesS.h" objId="13" />
      <rel kind="Includeby" objKind="File" objName="LinSM_PBcfg.h" objId="1" />
    </object>
```

Figure 4.26: Tool screen shot 3 - Part of the Diff result file

Similarly, merge tool is implemented. And as and when the mechanism is called, it's corresponding result file is created or updated. All 4 result files are seen in screen shot 4.27.



Name	Date modified	Type	Size
WindowsFormsApplication1	2015-07-24 13:01	File folder	
cmdln args - given.txt	2015-07-23 08:08	Text Document	1 KB
Ccl_Asr4SmLin_UC_IF.vsf	2015-07-22 16:58	VSF File	147 KB
Component.vsf	2015-07-16 08:28	VSF File	134 KB
LinSM.vsf	2015-07-14 13:58	VSF File	55 KB
VDiffResultFile1.vsf	2015-08-21 11:33	VSF File	26 KB
VDiffResultFile2.vsf	2015-08-21 11:38	VSF File	64 KB
VMergeResultFile1.vsf	2015-08-21 11:38	VSF File	140 KB
VMergeResultFile2.vsf	2015-08-21 11:42	VSF File	126 KB

Figure 4.27: Tool screen shot 4 - All 4 result files created (one after the other)

5 Testing

In software engineering, the process of checking the program that is implemented, matched the initial requirements, is called testing. Testing can be performed in various ways. During this thesis, the work included learning about various kinds of testing methods and testing levels. After the implementation was completed, it was tested in few of the mentioned ways and that is discussed further. In this chapter, the following topics are mainly discussed:

1. Testing methods - Possible ways of testing
2. Testing levels - Possible levels at which testing is done
3. Verification of implementation - All checks done after the scripts were ready

5.1 Testing methods

This section explains about what exactly are testing methods and the types that are considered for this thesis. This section is divided into following sub-sections:

- Static vs Dynamic testing
- Black box vs White box approach

Before brief explanation of each, it is important to know that there are various methods that can be used for testing. These methods are based on the information that are provided about the implementation, and also when are these tests performed.

5.1.1 Static vs Dynamic testing

As the name suggests, static testing concerns with testing of the program without running it. This includes going through the program in the form of reviews, walkthroughs or inspection of the code [Mel15]. During any of these static methods, as a developer, code is read through and verification against the requirements is tried as much as possible. In this thesis, it was performed by making the code into a modular format rather than a continuous format.
The program was divided into many smaller functions for each functionality. This made the program modular and easy to edit (add or delete functionality without affecting anything else).

Dynamic testing is performed when test cases are specifically written for a program.

When the program is executed, it gives test results based on test cases written for it. This was not covered during this thesis due to limitation of time and hence has been considered as a part of the future work of the thesis.

5.1.2 Black box vs White box approach

Before explaining how these approaches were used for this thesis, it is important to understand what is meant by each of these approaches. The abstract difference between the two is shown in figure reffig:fig52-Testing1.

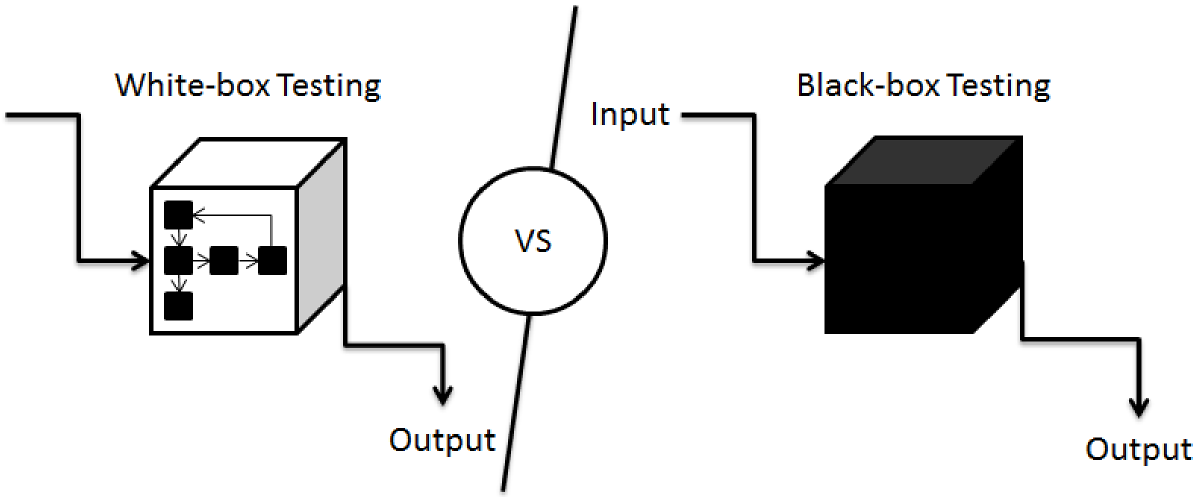


Figure 5.1: White box VS Black box testing approach

Black box approach -

This approach is used when a check has to be made, how the program is used and interacts with the user. User only knows about what the program does - what is input supposed to be and what is expected as output. When the program is given to the user, for testing purpose only basic information is given. The back end of the program is unknown. So, the review that is gathered from the user, about the usability and if it works with various inputs or not, is mainly this approach - called black box.
During this thesis, black box approach was done very briefly, by making some of the team members use the program. Feedback from them helped in improvisation on some usability details with respect to the program.

White box approach -

This approach is used by the developers, who know what is happening within the program. The usability based on the knowledge of the program is checked. The white box approach, was done by me since the working of the program in detail was known. The explanation of the tests done from white box approach has been comprehensively covered in section 5.3.

5.2 Testing levels

This section clarify the topic under testing levels. Although there are various levels of testing, in this section the following are discussed:

- Unit testing
- Integration testing

Both of these testing levels were used during this thesis but at a very primary level. Specific test cases and scripts are to be implemented as a part of future work (as already mentioned in previous section).

5.2.1 Unit testing

Unit testing as the name suggests, means testing each part of the program individually. This can be done by writing test cases for each and noting the results from each function. This can also be called module testing, since each part is tested separately. The dependencies can be faked or mocked during unit testing. And also, these tests run faster.

5.2.2 Integration testing

Integration testing is when the whole system is tested together. These tests can be slower than unit testing. Also, if any part of the program is not tested carefully, integration testing can fail at smallest of errors.

5.3 Verification of Implementation

This section explains the various tests performed on the implemented scripts. The results of each one were noted and then the data was modified for further tests. It is divided into 3 sub sections:

- Conversion of data
- Diff tool
- Merge tool

5.3.1 Conversion of data

Tests - The tests were done to check if the implemented scripts provided the results as expected. So, source code from real component was used for the tests. Data from EA and UDB for same component was used to check the result VSF files.

Intermediate Result - Result VSF file was created as expected. Next important thing was if the structure was found as designed for the file. VSF file was designed with a hierarchy. The objects that are placed under groups and some objects within objects, like the functions and parameters was checked. This was found to be same in both the result files.

Modification - Data was intentionally changed in the source code and the script was run again. This test was done to check if the modification is noticed and processed corresponding to the rules of the script.

Result - The script worked as intended and the bugs that were inserted in the source code or the modification that was done on the source code was handled as expected. For instance, if an entity is not defined in the right way, it was not identified to be a part of the result file.

5.3.2 Diff tool

Tests - The first test that was done was to check if the script was running as expected. Both the result VSF files were taken as input. Both the files was the output from the previous section 5.3.1. And since the result file was of the same component from both architecture and the source code, same component name was given as the argument for this test.

Intermediate Result - The script provided the output as expected and the comparison of data was done manually at this stage. This step had the following main checks:

- If the data existed in both files and was same in both files, the result is expected to output one value and describe it as ’success’.
- If the data existed in both files, but the values were different, the result is expected to output both values and describe it as ’mismatch’.
- If the data existed in any one of the file, the output is expected to be value of that one file and result as ’missing’, with the file name mentioned.

These results were successful.

Modification - Since the output for the 2 files was known, data from one of the file was intentionally modified. Some data was changed, some was removed and some were made to be same as in other file.

Result - The results even after modification were successful. The modified data was processed as expected and a new and different result file was created.

5.3.3 Merge tool

Tests - Tests began with checking if the script was implementing as expected. The input were same files that were collected in section 5.3.1 (Since they are collected from implementing with real data).

Intermediate Result - The script provided the output as expected and the comparison of data was done manually at this stage. This step had the following main checks:

- If the data had maintained its hierarchical nature. The objects were placed under right parent objects.
- If data existed in both files, and the data was same, it was directly placed in result file.
- If data existed in both files, and the data was different, base file was considered as the source code VSF file, and only that data was to be written in the result file.
- If the data only existed in any one of the files. If the data was from architecture VSF file, then it should not be considered for the results. If it is found only in source code VSF file, then it is considered and written in the result file.

These results were all successful.

Modification - Now that the result file was created and its successfully matched the rules, data was intentionally modified in both files slightly. For instance, some data was added, and some was deleted from base file - that is source code VSF file.

Result - The results even after modification were successful. The data that was deleted from base file was not found in new result file. That clearly shows that the script runs as per its defined rules.

6 Conclusion and future work

A research is incomplete if the differentiation between the scenarios of past work and present implementation of the theme is unable to be made. In order to recognize and realize the work done during this thesis, it is important to consolidate all the work and give a temporary closure. Also, no research can be ended without having an idea for scope of improvement and thus it is necessary to know what is the prospective of the work and topic.

All research and development done during this thesis has been encapsulated within the section of conclusion in this chapter. After which, a section on future work has been included to concretely give a compact overview of what can be further done with respect to this research.

6.1 Conclusion

The topic of this thesis deals with the concept of round-trip engineering, data models and understanding of AUTOSAR safety projects. The focus was to find more clarity on the process of development of a software, data that is exchanged and how does the mechanism work. The exchange of data that takes place between architecture and development levels have been studies extensively.

Since, V-model is being considered to have a start of concept, we know that the flow of data is only downwards. But, for making use of the data more efficient, this thesis research shows that flow of data can be reversed for a lot of positive changes. This research had 3 main aspects to look into and develop a concept, they are the data, the format and the process.

The main concept is build based on the problems that are found with the present process. The process and format of the data exchange was found. If the data that was used necessary was the next part of research. Conceptualization was focused on building the format for that data, which combines the advantages of other presently used formats. Once this was achieved, a mechanism was build that mainly focussed on 'exchangeable' property for the data. It was designed in a way that with the same base of the file, it could have different purposes. The technique involved verification of data from two different sources and merging of the data in such a way that it could in turn be used for updating in the reverse manner.

The results of the research have been proven to be positive in terms of usability, usefulness and covering the problems that existed before this. The fact that it is the same file that is being used, expanding the mechanism is even more interesting to use and see it directly. The base for this has already been designed well. With time constraints, the extra features including specific test-cases, test scripts, C code generator and also a document generator could not be implemented. The most important part of the result was that the concept that was developed was extremely generic. This has helped in a lot of ways and the results can be scaled to powerful levels in the time ahead.

6.2 Future Work

The concept developed does not confine itself to only be used for V-model software development process. This model can be used for any development process that involves interaction between various phases of development. And as far as the studies have shown, for software development, there always have to be data that is exchanged from one level to another.

Besides expanding to other processes, it has promising properties of covering the data exchange for entire V-model. This can be done from each level and data can be exchanged as per requirement to make the model usability even mote powerful.

At a more detailed level, present concept can be scaled to not just work with one component, but a cluster of components together. This will be useful to the other users that are presently not dealt with in this thesis, like all the users who exchange more detailed level data and also the people involved with testing of these components at each stage.

Overall, the solution developed, hints promising concepts of development presently for software development. But, as a concept it can be improvised and further developed with any process that involves data exchange.

Bibliography

[Aut] *AUTOSAR Tooling in Practice*. 2013 (cit. on p. 8).

[AUT15a] AUTOSAR.org. *AUTOSAR - Partners*. 2015. URL: `http://www.autosar.org/partners/current-partners/` (cit. on p. 6).

[AUT15b] AUTOSAR.org. *Technical Overview: AUTOSAR*. 2015. URL: `http://www.autosar.org/about/technical-overview/` (cit. on p. 6).

[AW05] Addison-Wesley. *Unified Modeling Language User Guide, The (2 ed.),p. 496*. 2005. ISBN: 0321267974 (cit. on p. 3).

[Boa15] International Requirements Engineering Board. *How the ReqIF Standard for Requirements Exchange Disrupts the Tool Market*. May 2015. URL: `http://re-magazine.ireb.org/issues/2014-3-gaining-height/open-up/` (cit. on pp. 13 sqq.).

[car15] carthrottle.com. *Disassembled MkII Golf*. 2015. URL: `http://www.carthrottle.com/post/check-out-your-picture-of-the-day/` (cit. on pp. 15 sq.).

[CDC15] CDC. *CDC UNIFIED PROCESS PRACTICES GUIDE*. 2015. URL: `http://www2.cdc.gov/cdcup/library/practices_guides/CDC_UP_Requirements_Management_Practices_Guide.pdf` (cit. on p. 12).

[Cle+10] Paul Clements et al. *Documenting Software Architectures: Views and Beyond, Second Edition*. 2010. ISBN: 0321552687 (cit. on pp. 9 sq.).

[Csh] *C Sharp (programming language)*. 2015. URL: `https://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29` (cit. on p. 21).

[EBC15] PE Eric Byres and John Cusimano. *Safety and Security: Two Sides of the Same Coin*. 2015. URL: `http://www.controlglobal.com/articles/2010/safetysecurity1004/` (cit. on p. 15).

[Ecl15] Eclipse.org. *Requirements Management for Eclipse*. May 2015. URL: `http://www.eclipse.org/rmf/` (cit. on p. 15).

[Gro] Object Management Group. *OMG Unified Modeling Language (OMG UML), Superstructure* (cit. on p. 10).

[Inc15] Suresoft Technologies Inc. *Automotive*. 2015. URL: http://www.suresofttech.com/solution/solution/ (cit. on p. 18).

[Ins15] National Instruments. *What is the ISO 26262 Functional Safety Standard?* 2015. URL: http://www.ni.com/white-paper/13647/en/#top (cit. on p. 16).

[Int15] Project Performance International. *What is the significance of different types of requirements such as states and modes, functional, performance, external interface, environmental, resource, physical, other qualities and design?* 2015. URL: http://www.ppi-int.com/systems-engineering/types-of-requirements.php (cit. on p. 12).

[Iso] *Road vehicles — Functional safety — Part 1: Vocabulary*. First edition, 2011-11-15. Reference number ISO 26262-1:2011(E). Nov. 2011 (cit. on p. 16).

[JC15] Steve DeRose James Clark. *XML Path Language*. 2015. URL: http://www.w3.org/TR/xpath/ (cit. on p. 42).

[Kum15] S Kumar. *What is V-model- advantages, disadvantages and when to use it?* May 2015. URL: http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/ (cit. on p. 2).

[Mal+15] Ivano Malavolta et al. *Architectural languages Today*. 2015. URL: http://www.di.univaq.it/malavolta/al/#people (cit. on p. 11).

[Mel15] MelbourneStar. *Software testing*. 2015. URL: https://en.wikipedia.org/wiki/Software_testing#Testing_methods (cit. on p. 79).

[MR04] McBride and Matthew R. *The software architect: essence, intuition, and guiding principles*. 2004. ISBN: 1-58113-833-4 (cit. on p. 9).

[Pix15] Pixshark. *Requirement*. 2015. URL: http://pixshark.com/requirement.htm (cit. on p. 12).

[Pyh04] Marko Pyhäjärvi. *SPICE International Standard for Software Process Assessment*. 2004 (cit. on p. 18).

[Req] *Requirement Interchange Format(RIF)*. 2015 (cit. on p. 13).

[Rpl13] Rplano. *Introduction to Software Engineering*. en.wikibooks.org, 2013 (cit. on pp. 2, 11).

[SIG10] Automotive SIG. *Automotive SPICE® Process Assessment Model*. May 10, 2010 (cit. on p. 19).

[Spa98] Geoffrey Sparks. *Enterprise Architect User Guide*. Sparx Systems Pty Ltd, 1998-2014 (cit. on p. 20).

[suc15] successsaravana. *AUTOSAR - Introduction*. 2015. URL: `http://successsaravana.blogspot.de/2014/08/autosar-introduction.html` (cit. on p. 8).

[Sys15] Vanadium Systems. *AUTOSAR*. 2015. URL: `http://vanadiumsystems.com/autosar.html` (cit. on p. 5).

[Uml] *UML for Systems Engineering: Watching the Wheels IET,p.58*. 2004. ISBN: 0-86341-354-4 (cit. on p. 10).

[Und] *User Guide and Reference Manual*. 2013 (cit. on p. 21).

[www15a] www.die.net. *perl(1) - Linux man page*. 2015. URL: `http://linux.die.net/man/1/perl` (cit. on p. 22).

[www15b] www.perl.org. *About Perl*. 2015. URL: `https://www.perl.org/about.html` (cit. on p. 22).

A Appendix

A.1 Examples of various file formats

A.1.1 Example of UnderstandC file format

```xml
<udbXml ucVersion="770" scriptVersion="2013-08-14">
<ent kind="File" kindlong="C␣Header␣File" name="LinSM_PBcfg.h" id="1" VScope="global" type="">
  <comment></comment>
  <include kind="File" kindlong="C␣Header␣File" name="LinSM_Types.h" id="3"/>
  <include kind="File" kindlong="C␣Header␣File" name="LinIf.h" id="4"/>
  <include kind="File" kindlong="C␣Header␣File" name="ComM.h" id="5"/>
  <define kind="Macro" kindlong="C␣Macro" name="LINSM_PBCFG_H" id="2"/>
  <define kind="Macro" kindlong="C␣Macro" name="LINSM_PBCONFIG" id="6"/>
  <define kind="Macro" kindlong="C␣Macro" name="LINSM_LTCONFIGIDXOFPBCONFIG" id="9"/>
  <define kind="Macro" kindlong="C␣Macro" name="LINSM_PCCONFIGIDXOFPBCONFIG" id="10"/>
  <use kind="Macro" kindlong="C␣Macro" name="LINSM_PBCFG_H" id="2"/>
  <includeby kind="File" kindlong="C␣Header␣File" name="LinSM_Cfg.h" id="675" line="47" column="0"/>
</ent>
<!--Object kind function-->
<ent kind="Function" kindlong="C␣Unresolved␣Function" name="ComM_MainFunction_0" id="719" VScope="global" type="void">
  <comment></comment>
  <declarein kind="File" kindlong="C␣Header␣File" name="ComM_Lcfg.h" id="700" line="104" column="29"/>
</ent>
```

A.1.2 Example of VCD file format

```
<vcd BaseType="Package" BaseName="Work.Product.Design.Detailed.Modules.LinSm" CreationDate="2015-02-24␣14:35:51">
  <ModelData Name="MSR4" Release="AR4-R12" ShortName="MSR4" Status="UnderDesign" TemplateVersion="10000" Process="Process2x" Type="Product" Version="4.12.00" />
  <Package Name="LinSm" Stereotype="" GUID="{D10FA7BB-8951-43a0-BA2D-08A7DDBCD3FB}">
    <Description><![CDATA[]]></Description>
    <Element GUID="{A68D7649-3244-4d3e-9678-634395C9F918}" Name="LinSM" Type="Class" Stereotype="Module" DateCreate="2007-02-13" DateModify="2014-10-10" ModifyAge="137" VersionCreate="1.0" VersionModify="1.0" VersionPlan="" Review="NotRequired">
      <Notes><![CDATA[LIN State Manager]]></Notes>
      <CompositDiagram Name="LinSM" GUID="{7E1D719B-C9A4-4b74-983D-6CBEC165261B}" />
      <Tags>
        <Tag Name="Cluster">
          <Value>LIN</Value>
        </Tag>
        <Tag Name="ComplexityClass">
          <Value>Normal</Value>
        </Tag>
<!--.....................-->
        <Dependency Type="Realisation" Stereotype="realize" Name="" Direction="out" MyMultiplicity="" TargetMultiplicity="" LinkedGUID="{5E542F8C-ADCD-49f6-89E7-85195020A637}" LinkedName="LinSm" LinkedType="Interface" LinkedStereotype="EmbeddedInterface">
                <Notes />
        </Dependency>
        <Dependency Type="Dependency" Stereotype="use" Name="" Direction="out" MyMultiplicity="" TargetMultiplicity="" LinkedGUID="{3DE52450-C179-45f2-9887-8981259EABCA}" LinkedName="BswM" LinkedType="Interface" LinkedStereotype="EmbeddedInterface">
                <Notes>BswM_LinSM_CurrentSchedule
        BswM_LinSM_CurrentState</Notes>
        </Dependency>
<!--.....................-->
        <Method GUID="{0C68D9B3-09A3-478a-8E3E-495051F7F527}" Name="LinSM_Init" Kind="ServiceFunction" Return="
```

```
            void">
                <Notes><![CDATA[This function initializes the
                   LinSM.<br/>]]></Notes>
                        <Tags>
                                <Tag Name="CallContext">
                                 <Value>Any</Value>
                  </Tag>
                  <Tag Name="IrLockContext">
                                 <Value />
                  </Tag>
                  <!--.........many more tags............-->
                  <Tag Name="MemResourceClassification">
                                 <Value>Medium</Value>
                  </Tag>
                  <Tag Name="Synchron">
                                 <Value>true</Value>
                  </Tag>
                </Tags>
                <ArgString>const LinSM_ConfigType* ConfigPtr</
                   ArgString>
                <ReturnType>void</ReturnType>
                <Parameter Name="ConfigPtr" Kind="in" Position=
                   "0" Type="const␣LinSM_ConfigType*"
                   Type_Plain="LinSM_ConfigType">
                         <Notes><![CDATA[Pointer to the LinSM
                            post-build configuration data.]]></
                            Notes>
                </Parameter>
                <UsedByModule GUID="{BE85B771-ECFF-4a77-961B
                   -7327B1154E8E}" Name="EcuM______TBDel" />
        </Method>
```

A.1.3 Example of ReqIf file format

```xml
<?xml version="1.0" encoding="UTF-8"?>
<REQ-IF xmlns="http://www.omg.org/spec/ReqIF/20110401/reqif.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.omg.org/spec/ReqIF/20110401/reqif.xsd http://www.omg.org/spec/ReqIF/20110401/reqif.xsd"
xml:lang="en">
<THE-HEADER>
<REQ-IF-HEADER IDENTIFIER="EF5AEFF3-C695-488A-9955-EDAAEACFE359">
<COMMENT>Optional comment associated with the Exchange Document as a whole</COMMENT>
<CREATION-TIME>2011-06-13T10:24:18+01:00</CREATION-TIME>
<REPOSITORY-ID>A4123EB9-CC82-4B62-95E1-31CB3203E39C</REPOSITORY-ID>
<REQ-IF-TOOL-ID>SparxSystems Enterprise Architect 8.0</REQ-IF-TOOL-ID>
<REQ-IF-VERSION>1.0</REQ-IF-VERSION>
<SOURCE-TOOL-ID>microTool in-Step</SOURCE-TOOL-ID>
<TITLE>Example ReqIF file</TITLE>
</REQ-IF-HEADER>
</THE-HEADER>
<CORE-CONTENT>
<REQ-IF-CONTENT>
<DATATYPES>
<DATATYPE-DEFINITION-STRING IDENTIFIER = "Text" LAST-CHANGE = "2011-06-13T00:00:00+01:00" MAX-LENGTH = "1000">
</DATATYPE-DEFINITION-STRING>
<DATATYPE-DEFINITION-ENUMERATION IDENTIFIER = "Status" LAST-CHANGE = "2011-06-13T00:00:00+01:00">
<SPECIFIED-VALUES>
<ENUM-VALUE DESC = "Requirement has been proposed." IDENTIFIER = "Proposed" LAST-CHANGE = "2011-06-13T00:00:00+01:00">
<PROPERTIES>
<EMBEDDED-VALUE KEY = "1" OTHER-CONTENT = "foo" />
</PROPERTIES>
</ENUM-VALUE>
<ENUM-VALUE DESC = "Requirement has been accepted." IDENTIFIER = "Accepted" LAST-CHANGE = "2011-06-13T00:00:00+01:00">
<PROPERTIES>
<EMBEDDED-VALUE KEY = "2" OTHER-CONTENT = "foo" />
</PROPERTIES>
```

```
</ENUM-VALUE>
<ENUM-VALUE DESC = "Requirement has been rejected or de-scoped
   ." IDENTIFIER = "Rejected" LAST-CHANGE = "2011-06-13T00
   :00:00+01:00">
<PROPERTIES>
<EMBEDDED-VALUE KEY = "3" OTHER-CONTENT = "foo" />
</PROPERTIES>
</ENUM-VALUE>
<ENUM-VALUE DESC = "Requirement has been satisfied by an
   implementation of system components." IDENTIFIER = "
   Implemented" LAST-CHANGE = "2011-06-13T00:00:00+01:00">
<PROPERTIES>
<EMBEDDED-VALUE KEY = "4" OTHER-CONTENT = "foo" />
</PROPERTIES>
</ENUM-VALUE>
<ENUM-VALUE DESC = "Satisfaction of requirement has been
   approved by QA and customer/user." IDENTIFIER = "Approved"
   LAST-CHANGE = "2011-06-13T00:00:00+01:00">
<PROPERTIES>
<EMBEDDED-VALUE KEY = "5" OTHER-CONTENT = "foo" />
</PROPERTIES>
</ENUM-VALUE>
</SPECIFIED-VALUES>
</DATATYPE-DEFINITION-ENUMERATION>
```

A.1.4 Example of AUTOSAR file format

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!---->
<AUTOSAR xsi:schemaLocation="http://autosar.org/schema/r4.0 AUTOSAR_4-0-3.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://autosar.org/schema/r4.0">
 <AR-PACKAGES>
 <AR-PACKAGE UUID="1a558d5f-22b4-4518-b46b-79a97233bf1f">
  <SHORT-NAME>MICROSAR</SHORT-NAME>
  <ELEMENTS>
   <ECUC-MODULE-DEF UUID="e394ce6b-b9e6-407b-925f-8ffdc04bd789">
   <SHORT-NAME>Msn</SHORT-NAME>
    <DESC>
     <L-2 L="EN">Configuration of the Msn (Generic Component) module.</L-2>
    </DESC>
    <CATEGORY>VENDOR_SPECIFIC_MODULE_DEFINITION</CATEGORY>
        <ADMIN-DATA>
    <DOC-REVISIONS>
         <DOC-REVISION>
         <REVISION-LABEL>-</REVISION-LABEL>
         <ISSUED-BY>virchl</ISSUED-BY>
         <DATE>2014-08-19T06:01:53+02:00</DATE>
         <MODIFICATIONS>
          <MODIFICATION>
           <CHANGE>
            <L-2 L="EN">SW Version Number set to 3.00.03</L-2>
           </CHANGE>
          </MODIFICATION>
         </MODIFICATIONS>
         </DOC-REVISION>
         <DOC-REVISION>
         <REVISION-LABEL>3.00.04</REVISION-LABEL>
         <ISSUED-BY>virchl</ISSUED-BY>
          <DATE>2015-01-13T04:51:04+01:00</DATE>
          <MODIFICATIONS>
           <MODIFICATION>
            <CHANGE>
            <L-2 L="EN">Default value of MsnVersionInfoAp set to false</L-2>
            </CHANGE>
            <REASON>
            <L-2 L="EN">ESCAN00067535</L-2>
            </REASON>
           </MODIFICATION>
          </MODIFICATIONS>
         </DOC-REVISION>
        </DOC-REVISIONS>
        </ADMIN-DATA>
    <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
    <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
    <REFINED-MODULE-DEF-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/
       EcucDefs/Msn</REFINED-MODULE-DEF-REF>
    <SUPPORTED-CONFIG-VARIANTS>
     <SUPPORTED-CONFIG-VARIANT>VARIANT-PRE-COMPILE</SUPPORTED-
        CONFIG-VARIANT>
    </SUPPORTED-CONFIG-VARIANTS>
```

A.1.5 Example of conceptualized VSF file format

```
<!--Intermediate format example based on real data:-->
<component>
        <object id="21" kind="SequenceDiagram">
                <prop name="name">SeqDiagram</prop>
        </object>
        <object id="31" kind="HeaderFile">
                <prop name="name">FileName</prop>
                <prop name="context"></prop>
        </object>
        <object id="1" kind="Routine">
          <prop name="id">1</prop>
          <prop name="name">A</prop>
          <prop name="desc">TEXT</prop>
          <prop name="callcontext"></prop>
          <prop name="details"></prop>
          <prop name="lockcontext"></prop>
          <prop name="reentrant"></prop>
          <prop name="returntype"></prop>
          <prop name="returntypedesc"></prop>
          <prop name="serviceid"></prop>
          <prop name="stereotype"></prop>
          <prop name="synchron"></prop>
          <prop name="preconditions"></prop>

          <rel id="rel1" kind="use" dir="o" relobj="2">
          <rel kind="contain" relatedId="ParamA1" dir="" id="rel123">
          <rel kind="use" relatedId="345" dir="o" id="rel2">
          <object id="123" kind="Parameter">
                <prop name="name">ParamA</prop>
                <prop name="desc">TEXT</prop>
                <prop name="type">TEXT</prop>
                <prop name="kind">TEXT</prop>
          </object>
          <object id="345" kind="Variable">
                <prop name="name">VarA</prop>
                <prop name="type">TEXT</prop>
                <rel kind="define" relatedId="3" dir="o" id="rel3">
                <rel kind="use" relatedId="1" dir="i" id="rel2">
          </object>
        </object>
        <object id="2" kind="TypeDefinition">
                <prop name="type">TypeA</prop>
                <prop name="typedesc">TEXT</prop>
                <prop name="typemember">TEXT</prop>
                <prop name="typememberdesc">TEXT</prop>
                <rel kind="use" relatedId="1" dir="i" id="rel1"
                   >
        </object>
        <object id="3" kind="Cfile">
                <prop name="desc">TEXT</prop>
                <prop name="includeheader">TEXT</prop>
                <prop name="variable">TEXT</prop>
                <rel kind="use" relatedId="1" dir="i" id="rel1"
                   >
        </object>
```

A.2 Detailed Embedded Model

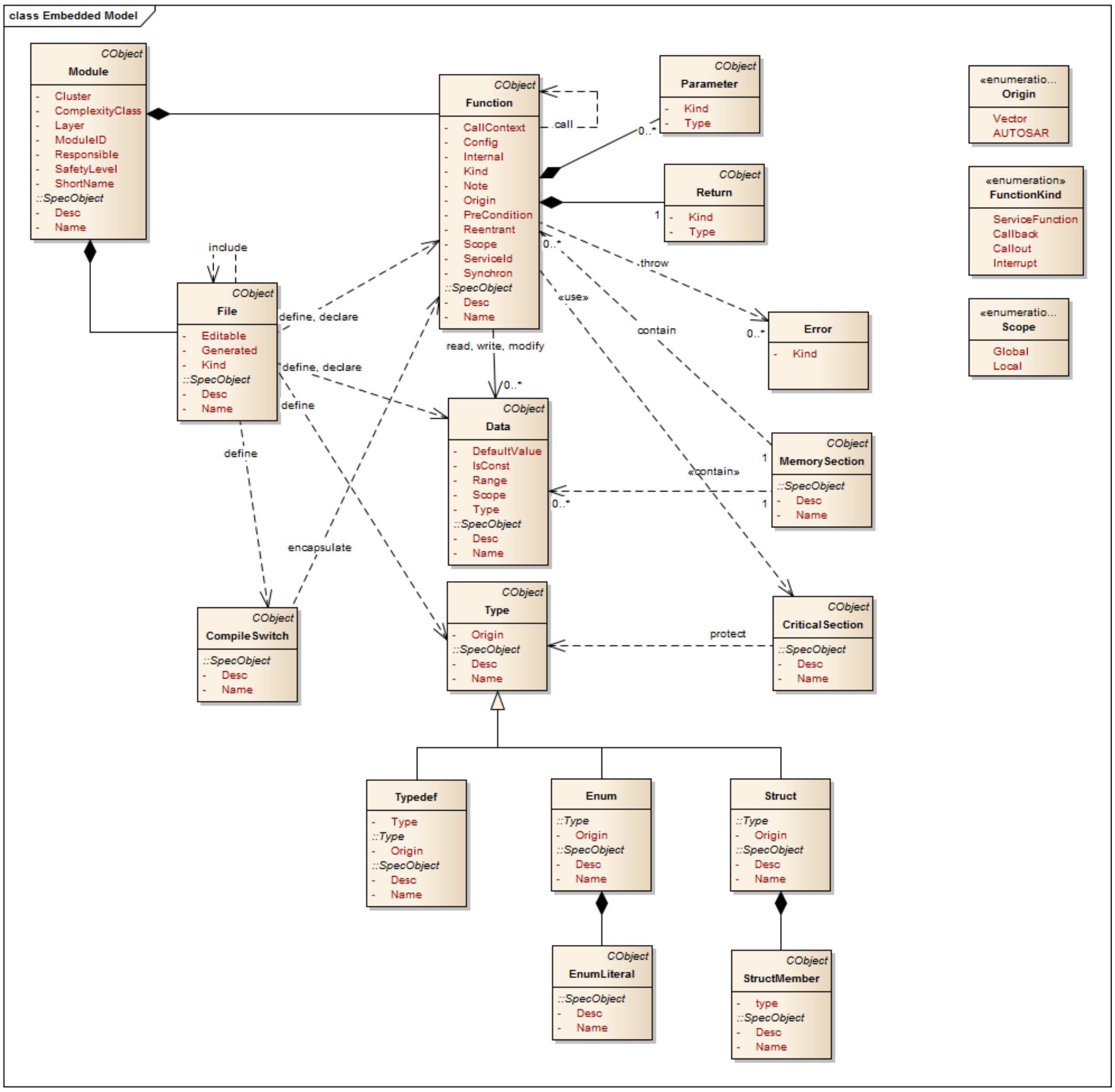


Figure A.1: Detailed Embedded Model