Info-Chunk Objects as New Behavior Representation for System-based Model of Product

Yatish Bathla

Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University, Bécsi út 96/b, H-1034 Budapest, Hungary
yatish.bathla@phd.uni-obuda.hu

Abstract: Requirement Functional Logical Physical (RFLP) structure has emerged as one of the prominent approaches for modeling the multidisciplinary products. Information Content (IC) provides effective interaction between the human and multidisciplinary product model. Though it controls the RFLP level by the Multilevel Abstraction based Self-Adaptive Definition (MAAD) structure, it needs to be further enhanced in terms of Human-Computer Interaction (HCI), multidisciplinary product behavior representation and structured processing of interrelated engineering objects to obtain coordinated decisions. Therefore, this paper introduces the Object-Oriented Principle (OOP) concepts in the IC for behaviors representation of the multidisciplinary product where Info-Chunk is considered as an object. Here, Behavior Info-Chunk (BiC) and Context Info-Chunk (CxiC) objects are proposed in the MAAD structure to model the behavior of the multidisciplinary product. Further, the concepts of Info-Chunk objects are extended to Intelligent Property (IP) that uses Initiative Behavior Context and Action (IBCA) structure to handle the RFLP structure. Based on the communication between the MAAD and RFLP structure, an API (Application Programming Interface) called "InfoChunkLib" is proposed. It can generate the graphs to represent the behaviors of a multidisciplinary product model. The API is handled by the information content to represent the behavior information and store the results in a database.

Keywords: Behaviors representation; Multidisciplinary product modeling; Info-Chunk based Information Content; RFLP structure; MAAD structure; IBCA structure

1 Introduction

Modeling of multidisciplinary products requires coordination of a significant amount of model information. The integrated definition is raised to the conceptual level of product design, which requires high-level abstraction. A four-leveled structure of the product model using Requirement Functional Logical Physical

(RFLP) structure [2] was introduced in the virtual environment. It is applied from system engineering and offers handling product and its model as a system. It accommodates product behavior definitions on its F and L levels. Product assembly is done in the specification tree (red square) of RFLP structure as shown in Fig. 1. Due to complex Human-Computer Interaction (HCI), Information Content (IC) was used to record and apply the content of information that is represented in the product model space [19]. In this content, an intent is defined by the human to control the definition of engineering objects [22]. IC [1] controls the RFLP level by the Multilevel Abstraction based Self-Adaptive Definition (MAAD) structure [2]. However, IC needs to be enhanced in terms of the practical feasibility of HCI, behavior representation and structured processing of interrelated engineering objects to obtain the coordinated decisions. To solve above-mentioned issues, this research work proposes the Info-Chunk objects and InfoChunkLib API (Application Programming Interface) is proposed in the IC.

Info-Chunk objects are based on the Object-Oriented Principle (OOP) concepts that are used in software programming. Previous research work deployed OOP concepts in the RFLP structure in the form of the Modelica language [6]. It is used for logical and physical modeling of a multidisciplinary product. Here, models and their components are defined by the object diagram. This research work uses OOP concepts in the Functional layer and Logical layer of the RFLP structure for behaviors representation of the multidisciplinary product modeling. Here, Info-Chunk entity is converted into the object first. Then, Behavior Info-Chunk (BiC) object and Context Info-Chunk (CxiC) object are introduced in the MAAD structure and the IBCA structure to store the behaviors of the multidisciplinary product. The proposed Info-Chunk objects are used to establish a link with the Layer Info-Chunk (LiC) objects of RFLP structure. InfoChunkLib API is coded based on the communication between the MAAD structure and RFLP structure. The Java language is used as a JavaFX application. It represents the behaviors of the components in the multidisciplinary product model. The generated output is shown using the graph between the components of engineering disciplines. IC imports the InfoChunkLib API and coded as a Web application. The multidisciplinary product model can be more efficiently handled through the IC instead of the Specification Tree. This paper begins with the conversion of Info-Chunk entity into the object. Further, behavior Info-Chunk (BiC) objects and context Info-Chunk (CxiC) objects are proposed in the MAAD structure and the IBCA structure. Then, behavior storing techniques for the multidisciplinary product are explained with the aforementioned concepts. Then, Info-Chunk objects based Information Content (IC) is emphasized. Here, rules for generating the BiC objects and CxiC objects in the MAAD structure are defined by using the pseudo-codes. Then, InfoChunkLib API is over viewed. Here, LiC objects of the RFLP structure, BiC and CxiC objects of the MAAD structure are demonstrated. Finally, InfoChunkLib API is imported in the IC to handles the multidisciplinary product model.

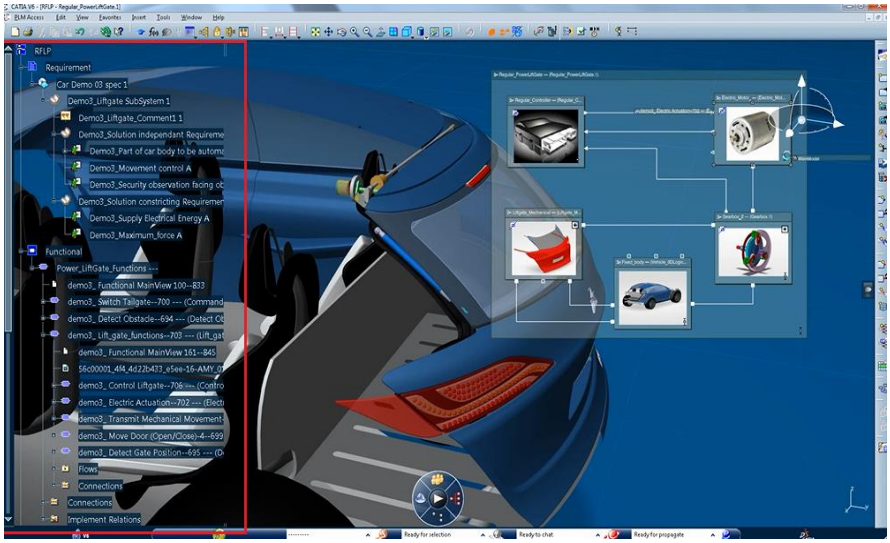

Figure 1
Specification tree of the CATIA V6 RFLP structure

2 Background

The Classical Product Model (CPM) [1] is limited to the physical level. The separated integrated mechanical engineering modeling increasingly demands multidisciplinary integration [5]. Modeling of a multidisciplinary product must have a means for the integration of discipline-specific models into a model with a unified structure. It makes the product model virtually executable. Higher abstraction is realized by using of RFLP structure product model [2]. It is compliant with the IEEE 1220 standard. Requirement against the product function to fulfill the requirement, product-wide logical connections, and representations of physically existing objects was organized in the highly contextual RFLP structure.

Human-Computer Interaction (HCI) during the multidisciplinary product modeling is a challenging task. Therefore, Information Content (IC) [1] assists in effective communication between engineers of different disciplines and information-oriented product modeling procedures. Community zones [17] are used in the IC to organize the product model entities and their relationship. Further, behaviors of the modeled entities are evaluated in the information content by the process plane [13]. IC requires MAAD structure to drive the levels of RFLP structure. This structure is used for self-adaptive modeling, where the *objectives and requests* level, *product behaviors* level, *contexts* level, *actions*

level, and feature objects are applied in order to connect engineers with RFLP implementations [4]. The MAAD modeling methods and model structures are introduced as a generalized means for the support of higher level abstraction-based generation of RFLP elements. The MAAD modeling was based on the knowledge representation, contextual change propagation, and extended feature definition capabilities for advanced modeling systems [4]. Further, active knowledge in a product model has become organized in the form of Intelligent Property (IP) of the company. Here, IP drives the RFLP level by the IBCA structure which represents active knowledge content [5].

To store the information of a multidisciplinary product, Info-Chunk entity is introduced in the logical level of the RFLP structure [9]. This entity is mapped with the information content to control the structure activities through the MAAD structure. Here, the Layer Info-Chunk (LiC) entity stores the information of the Logical layer and the Component Info-Chunk (CiC) entity stored the information of the logical component. Then, the Info-Chunk entity is defined in the Functional layer of the RFLP structure [19]. Here, the Layer Info-Chunk (LiC) entity stores the information of the main function of the Functional layer of the RFLP structure and Sub-function Info-Chunk (SFiC) entity stores the information of sub-function. Nowadays, OOP concepts are used in system engineering as object-oriented system engineering (OOSE) [3]. OOSE blends system engineering with software engineering.

3 Info-Chunk as an Object

In this research work, OOP concepts are used for multidisciplinary product modeling. Encapsulation, inheritance, and polymorphism are the three principles of OOP methodology. This work starts with the entities and their relationship. Info-Chunk [9] [19] is an entity defined in the RFLP structure. However, OOP concepts are not directly applicable to an entity. For InfoChunkLib API, Info-Chunk entity must be converted into the Info-Chunk object for communication between IC and RFLP structure. Based on the entity-object conversion process by Ou Y. [10] and Bernhard Thalheim [11]:

- The parameters of an Info-Chunk entity is equivalent to the attribute of an Info-Chunk object
- ER (Entity Relationship) between Info-Chunk is equivalent to the OR (Object Relationship). Here, the method of an Info-Chunk object is derived from the OR as per the requirement of a specific discipline

Then, behavior Info-Chunk (BiC) object and context Info-Chunk (CxiC) object are proposed in the MAAD structure and IBCA structure. According to the proposed concept of Info-Chunk objects:

- In the RFLP structure, logical layer Info-Chunk (LiCL) object consist of the attributes and methods of the Logical level and functional layer Info-Chunk (LiCF) objects consist of the attributes and methods of Functional level
- In the MAAD structure, behavior layer Info-chunk (BiC) object consist of the attributes and methods of Behaviors level and context layer Info-Chunk (CxiC) objects consist of attribute and method of Contexts level
- In the IBCA structure, behavior layer Info-chunk (BiC) objects consist of the attributes and methods of *Situation defining behaviors (SB)* level of Behavior substructures and context layer Info-Chunk (CxiC) objects consist of attribute and method of *Product definition Activity Contexts (AC)* level, *Adaptive Drive Contexts (DC)* level, *Product Feature Contexts (FC)* level of Contexts substructures

4 Behavior Storing Techniques using Info-Chunk Objects

Behavior is based on well-defined situations for sets of circumstances. It is represented in the Functional level and Logical level of the RFLP structure. BiC objects and CxiC objects represent dynamic behavior information. They are stored in the MAAD structure and IBCA structure to communicate with the LiC objects of the RFLP structure. Information Content operates the RFLP structure by the MAAD structure. Also, Intelligent Property (IP) operates the RFLP structure by IBCA structure. The behavior storing techniques are classified as the operation performed by the BiC objects and CxiC objects in the MAAD structure and IBCA structure.

4.1 Info-Chunk Objects-based MAAD Structure

The Behavior level of the MAAD structure drives the Functional level and Logical level of the RFLP structure. The relationship between the abstraction levels of the MAAD structure is described by using the Unified Modelling Language (UML) diagram as shown in Fig. 2. In the Object-Oriented Programming, a UML diagram is used to define the relationship and model the behavior of the product.

Here, Entities Relationship Modeling (ERM) of the MAAD structure is converted into object relationship modeling (ORM). As per the ORM concept, the relationship between objects is defined by the composition, aggregation, and association. Hence, there is a bi-directional association relationship between the *Objectives and Requests* level, *Behaviors* level, *Contexts* level, and *Actions* level.

Inside the Behaviors level, the behavior object has a composite relationship with the situation object, which further has an aggregation relationship with the circumstances object. Also, the behavior object has a bi-directional association with the Adaptive Drive object. In the MAAD structure, a behavior is represented at Behaviors and Contexts level. For behavior representation, communication between the RFLP structure and the MAAD structure is done by using the proposed BiC objects and CxiC objects.

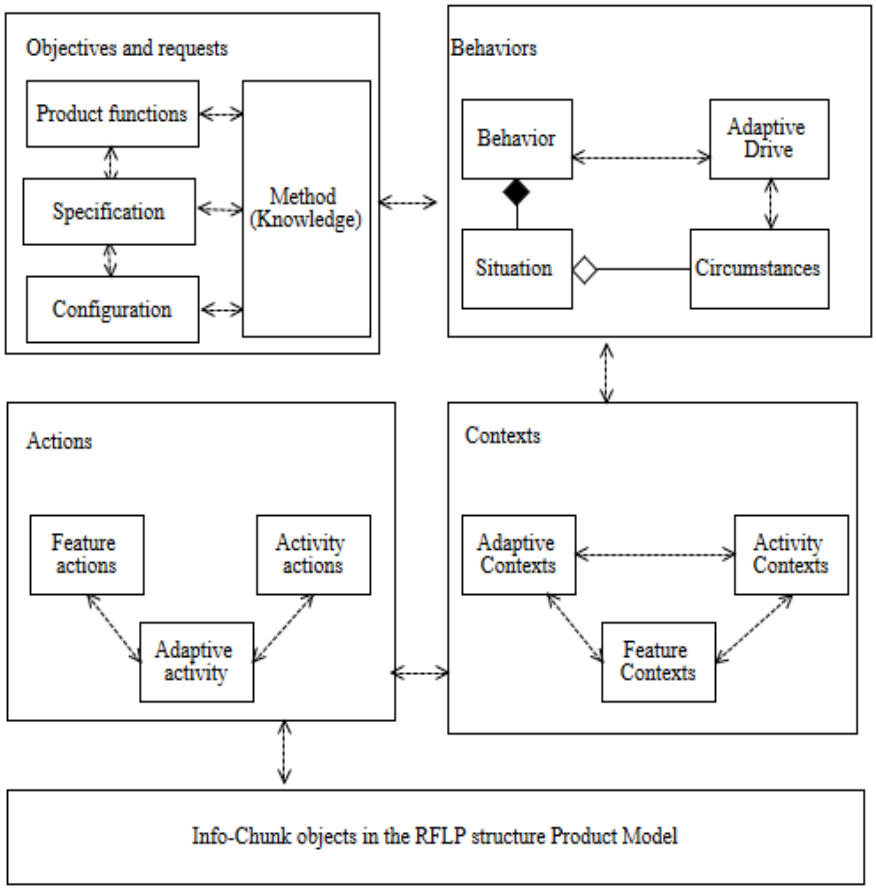

Figure 2
UML representation of MAAD structure with the Info-Chunk based RFLP Structure

The BiC objects communicate with the LiC objects as shown in Fig. 3, where the main contextual connections of the MAAD structure are organized as follow:

- The solid line is the inside contexts (C) of Behaviors levels for the MAAD structure. It is explained in the paper [18], where the contextual connection of model entities in the MAAD level is defined.

- The bold line is the driving contexts (D) of Behaviors levels for the MAAD structure. It drives the Functional level and Logical level of the RFLP structure. The dashed lines are the information retrieved by the BiC objects from the LiC objects of the Functional level and Logical level.

In the case of the Logical layer of RFLP structure, it retrieves the *situation* attribute of the LiCL object {$LiCL_1$, $LiCL_2$, .. $LiCL_o$} and corresponding *behavior* attribute of their CiC objects {CiC_1, CiC_2, .. CiC_n}. It is represented inside the oval shape in the diagram. The information retrieved by the driving contexts populates the BiC objects in the Behaviors level of MAAD structure. Here, n is the number of CiC objects in a LiCL object and o is the total number of LiCL objects in the logical layer. The information retrieved is the actual situation, circumstances for the situation and the adaptive drive to drive context definitions.

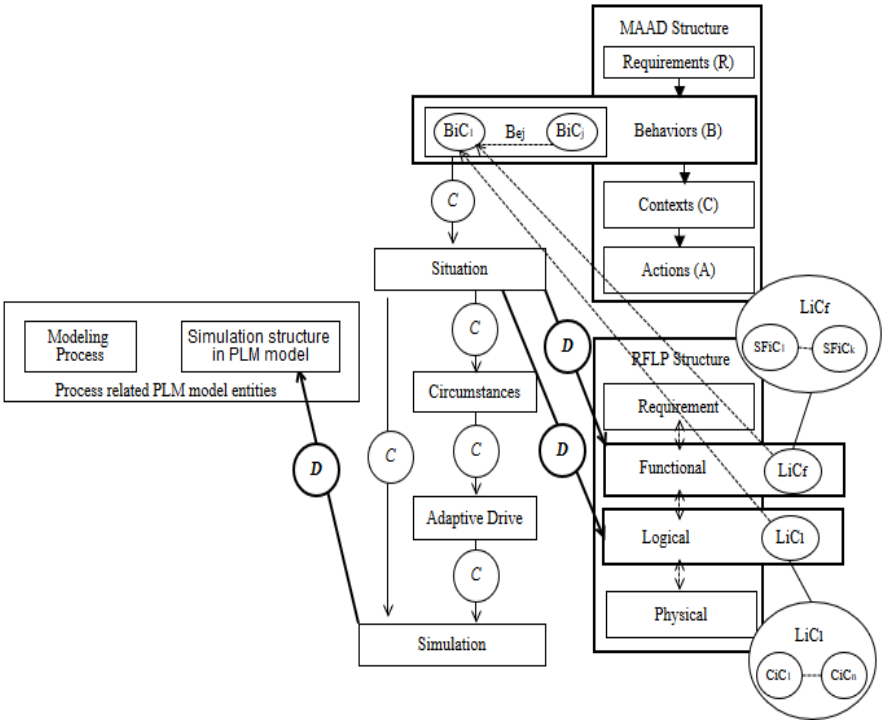

Figure 3
Communication between RFLP and MAAD structure at Behaviors level

In the case of Functional layer of RFLP structure, driving contexts (D) retrieves the *requirement class* attribute of the LiCF object {$LiCF_1$, $LiCF_2$, .. $LiCF_l$} and corresponding *elements description* attributes of the SFiC objects {$SFiC_1$, $SFiC_2$,.. $SFiC_k$}. It is represented inside the oval shape in the diagram. The information retrieved by the driving contexts populates the BiC objects in the Behaviors level

of MAAD structure. Here, k is the number of SFiC objects in a LiCF object and l is the number of LiCF objects in the functional layer of RFLP structure

The retrieved BiC objects are represented as {BiC_1, BiC_2,... BiC_j}. Here, j is the number of BiC objects in the Behaviors substructure. The CxiC objects communicate with the LiC objects is shown in Fig. 4, where the main contextual connections of the MAAD structure is organized as follows:

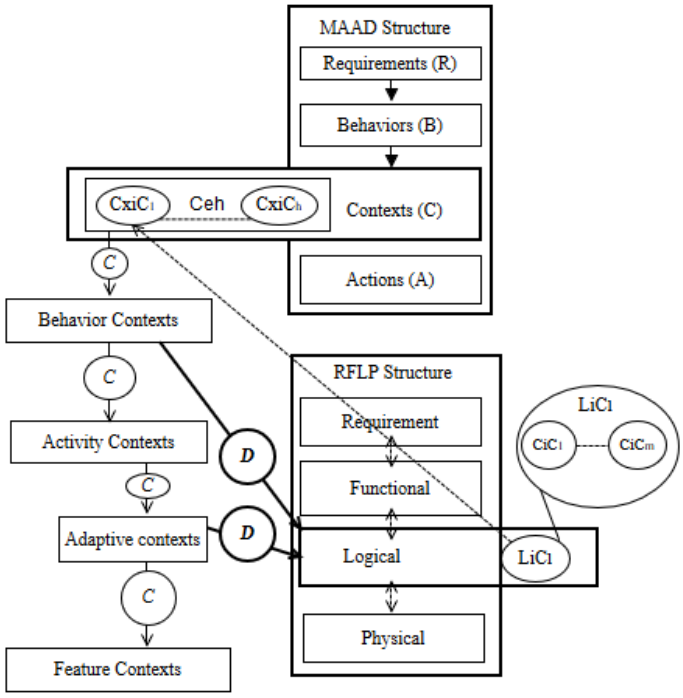

Figure 4
Communication between RFLP and MAAD structure at Contexts level

- The solid line is the inside contexts (C) of Contexts levels for the MAAD structure. It is explained in the paper [18], where the contextual connection of model entities in the MAAD level is defined.
- The bold line is the driving contexts (D) of Behaviors levels for the MAAD structure. It drives the Logical level of the RFLP structure. The dashed line is the information retrieved by the CxiC objects from the LiC objects of the Logical level.

In the case of the Logical layer of RFLP structure, it retrieves the *data model* attribute of the LiCL object {$LiCL_1$, $LiCL_2$, .. $LiCL_o$} and corresponding *data model* attribute of CiC objects {CiC_1, CiC_2, .. CiC_m}. Here, m is the number of CiC objects in a LiCL object and o is the total number of LiCL objects in the

logical layer. The information retrieved is the concept behavior, activity, adaptive and product feature contexts, connection behavior definitions, model definition activities, contexts for an adaptive drive, and context for physical level product and knowledge features. The retrieved CxiC objects are represented as {$CxiC_1$, $CxiC_2$,... $CxiC_h$}, where, h is the number of CxiC objects in the Contexts substructure.

4.2 Info-Chunk Objects-based IBCA Structure

The driving generation of the RFLP element is done by the Intelligent Property (IP). Human-initiated engineering activities with the company IP by using IBCA structure for the generation of RFLP elements. It leads to the analysis of self-adaptive product lifecycle management (PLM) modeling. The Info-Chunk objects based IBCA structure drives the RFLP structure as shown in Fig. 5. The solid lines are the interaction between the IBCA structure and RFLP structure. The dashed lines are the information retrieved by the BiC objects and CxiC objects from the LiC objects of the Functional level and Logical level. On the Behavior (B) level of the IBCA structure, situations defining behaviors (SB) substructure are configured to define behaviors by a set of BiC objects.

- In the Logical level of the RFLP structure, the *situation* attribute & *behavior* attribute of the LiCL object {$LiCL_1$, $LiCL_2$,... $LiCL_d$} and the corresponding *behavior* attribute of the CiC objects {CiC_1, CiC_2,... CiC_a} are stored in the BiC objects of the SB element. Here, a is the number of CiC objects in a LiCL object and d is the total number of LiCL objects.
- In the Functional level of the RFLP structure, *requirement* attribute of the LiC object {LiC_1, LiC_2,... LiC_c} and corresponding *elements description* attributes of the SFiC objects {$SFiC_1$, $SFiC_2$,.. $SFiC_b$} are stored in the BiC objects of the SB element. Here, b is the number of SFiC objects in a LiC object and c is the number of LiC objects in the functional level.

The stored information in the BiC objects is behavior definition (IEBD) and the related situation (IEBT) [12]. The total BiC objects obtained from the LiC objects of the functional and logical layer is represented as {BiC_1, BiC_2,... BiC_n}. Here, j is the number of BiC objects in the SB element. On the Contexts (C) level of the IBCA structure, *product definition activity contexts (AC)* level, *adaptive drive contexts (D*C*)* level, and *product feature contexts (FC)* level are configured to define behaviors by a set of CxiC objects. In the logical level of the RFLP structure, the *data model* attributes of LiC objects {LiC_1, LiC_2,... LiC_d} & CiC objects {CiC_1, CiC_2,.. CiC_a} are stored by the CxiC objects of AC, DC and FC elements. Here, a is the number of CiC objects in a LiC object and d is the total number of LiC objects. The stored information in the CxiC objects is the product behavior (IE_{CB}). The total CxiC objects obtained from the LiC objects of logical

layer is represented as {CxiC1, CxiC2, .. CxiCx}, {CxiC1, CxiC2, .. CxiCy}, {CxiC1, CxiC2, .. CxiCz}. Here, x, y, z are the number of CxiC objects stored in the AC, DC and FC elements.

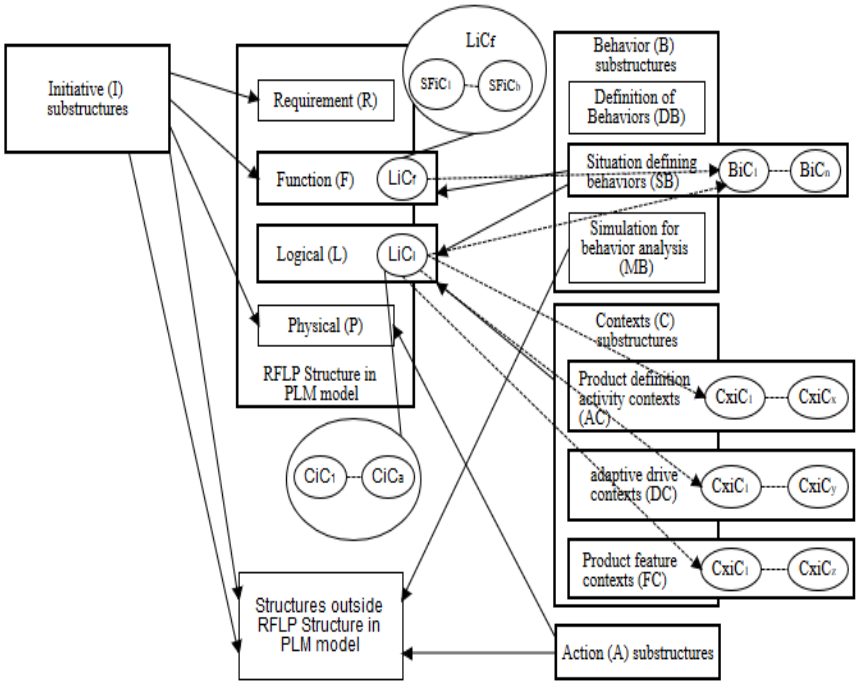

Figure 5
Communication between RFLP and IBCA structure at Behavior and Contexts substructure

5 Info-Chunk Objects-based Information Content

Behavior models with intelligent content involve specifications and knowledge for the design processes. The most appropriate forms of knowledge are formulas, rules, and checks. In the following sections, this work focuses on Info-Chunk object activities in the information content (IC). Here, the MAAD structure is the driving factor for representing the behavior of the RFLP structure.

5.1 Rules for the Generation of Info-Chunk Objects

Rules are the set of instructions that can be executed for generating and storing the Info-Chunk objects in the MAAD and IBCA structure. Rules are defined by using pseudo-codes.

- In the case of the MAAD structure, the behavior objects {BiC_1, BiC_2,... BiC_j} are stored in the behaviors level and the context objects {$CxiC_1$, $CxiC_2$,... $CxiC_h$} are stored in the contexts level. The Process plane of IC [13] can elaborate on the BiC and CxiC objects for the behavior representation of the multidisciplinary product. After the analysis process, the analyzed objects are stored with the nomenclature of BiC_{ab}. If a human wants to evaluate the context of one analyzed object on the other analyzed object, the context object undergoes the effect process. The resultant objects are stored as $CxiC_{ec}$. Further, If a human wants to optimize the contextual object, it is stored as BiC_{ob} after the optimization process. It is also possible to optimize the behavior of an object without analysis. Information content (IC) retrieve and store required objects at the *Engineering objectives* level to drive the behavior of RFLP structure.
- In the case of the IBCA structure, the behavior objects {BiC_1, BiC_2 , .. BiC_n} are stored in the behavior substructure and the context objects {$CxiC_1$, $CxiC_2$, .. $CxiC_x$}, {$CxiC_1$, $CxiC_2$, .. $CxiC_y$}, {$CxiC_1$, $CxiC_2$, .. $CxiC_z$} are stored in the contexts substructures. IP could retrieve and store these objects to drive the behavior of the RFLP structure. The IP level and process plane of IP are not defined yet. The behavior representation for IP is the topic of future work.

Pseudo Codes for BiC & CxiC objects

- BEGIN LOOP
- **Initialize** a Process
- IF 'Process' is 'Analysis'
    - BEGIN LOOP
    - Store 'BiC_{ab}' in 'Behaviors level' where $1 \leq ab \leq j$
    - IF 'Process' is 'Effect'
        - BEGIN LOOP
        - Store '$CxiC_{ac}$' in 'Contexts level' where $1 \leq ac \leq h$
        - IF 'Process' is 'Optimization'
            - BEGIN LOOP
            - Store 'BiC_{ob}' in 'Behaviors level' where $1 \leq ob \leq j$
            - END LOOP
        - END LOOP
    - END LOOP
- IF 'Process' is 'Optimization'

- BEGIN LOOP
- Store 'BiC_{ob}' in 'Behaviors level' where $1 \leq ob \leq j$
- END LOOP

6 Overview of the InfoChunkLib API

The *InfoChunkLib* API is coded in the JavaFX application as shown in Fig. 6. It consists of two Java packages. The *informationcontent* Package consists of all the classes related to the Information Content (IC) like *MAADStructure* class, *BiC* class, *CxiC* class, and *CommunityZone* class. The *rflp* Package consists of all the classes related to the RFLP structure like *LiCL* class, *LiCF* class, *CiC* class, and *SFiC* class.

6.1 Demonstration of Info-Chunk Objects in the RFLP Structure

To explain the proposed concepts in the system behavior, let us consider a car as an example. According to the community concepts [17], a car system is the combination of various communities where the *Electrical supply system* is one of the community. It consists of components like battery, starter, alternator, heater, fan, distributor, etc. Here, battery and alternator components are used for the Info-Chunk objects concept explanation. Then, the following scenario is considered as an example: The oil consumption of car depends on the engine behavior that must be modeled according to the situation such as the experience of the driver with the sets of circumstances like path traveled by car, the condition of the car, surrounding environment, etc.

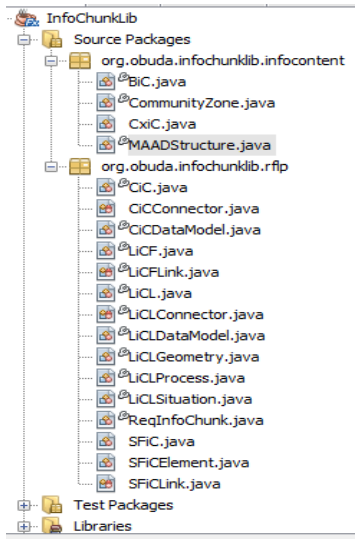

Figure 6
InfoChunkLib API

Here, the dynamic behavior of the engine strongly is influenced by the situation and weakly influenced by the circumstances. The parameters of the LiCL objects and CiC objects are described in the paper [9]. Also, the parameters of LiCF objects and SFiC objects are described in the paper [19]. The descriptions of LiCF

class and LiCL class are shown in the code below. The *LiCF* class is written to show the layer Info-Chunk object of the functional layer in the RFLP structure. Here, the array of SFiC objects, ReqInfoChunk object, LiCFLink object and String parameters are used as an argument in the constructor. The constructor arguments of the ReqInfoChunk object are populated from the Requirement layer of the RFLP structure. The *ReqInfoChunk* class is written to show the customer requirements in the requirement layer of the RFLP structure. The *SFiC* class is written to show the sub-functional Info-Chunk object in the functional layer of the RFLP structure. The LiCFLink is the enumeration class to store connector information. The concepts of constructor overloading are used so that LiCF can accept various sets of the argument depends on the initialization of the object.

```
// LiCF.java
/**This class is written to show the layer Info-Chunk
object of the functional layer in the RFLP structure
*@param func_name This parameter stores the name of a
function
*@param func_descrip This parameter stores the
description of a function
*@param comm_name This parameter stores the community
name of a function
*@param func_input This parameter stores the inputs to
a LiCF object
*@param func_output This parameter stores the outputs
from a LiCF object
*@param arrySFiC This parameter stores the array of the
SFiC(Sub-Function InfoChunk) objects
*@param req This parameter initializes the
RFiC(Requirement InfoChunk) object
*@param funct_link This parameter stores links between
the two function of LiCFLink type. It could be Data
flow or Control flow
*/
package org.obuda.infochunklib.rflp;
public class LiCF {
 String func_name, func_descrip, comm_name, func_input,
func_output; SFiC[] arrySFiC = null; ReqInfoChunk req =
null; LiCFLink func_link;
/**
*This constructor is used to initialize the LiCF
objects without information from the requirement layer
*@param name_func This parameter defines the name of a
function
*@param descrip_func This parameter defines the
description of a function
```

```
*@param name_comm This parameter stores the community
name of a function
*@param link This parameter stores links between the
two function of LiCFLink type. It could be Data flow or
Control flow
*@param input_func This parameter stores the inputs to
a LiCF object
*@param output_func This parameter stores the outputs
from a LiCF object
*@param subArry This parameter initialize the array of
the SFiC(Sub-Function InfoChunk) objects
*/
    public LiCF (String name_func, String descrip_func,
String name_comm, LiCFLink link, String input_func,
String output_func, SFiC[] subArry) {
func_name = name_func; func_descrip = descrip_func;
comm_name = name_comm; func_link = link; func_input =
input_func; func_output = output_func; arrySFiC =
subArry;}
/**
*This constructor is used to initialize the LiCF
objects with the information from the requirement layer
*@param spec_LiC This parameter stores the
specification of a LiCF object
*@param design_LiC This parameter stores the design of
a liCF object
*/
    public LiCF (String name_func, String descrip_func,
String name_comm, LiCFLink link, String input_func,
String output_func, SFiC[] subArry, String spec_LiC,
String design_LiC) {func_name = name_func; func_descrip
= descrip_func; comm_name = name_comm; func_link =
link; func_input = input_func; output_func =
func_output; arrySFiC = subArry; req = new ReqInfoChunk
(spec_LiC, design_LiC);
    }}
```

Further, the *LiCL* class is written to show the layer Info-Chunk object of the logical layer in the RFLP structure. Here, the array of CiC objects, LiCLGeometry object, LiCLSituation object, LiCLProcess object, LiCLDataModel object, LiCF object, LiCLConnector object, integer parameter, boolean parameter, string parameters, and the array of string parameters are used as an argument in the constructor. The constructor arguments of a LiCF object are populated from the Functional layer of the RFLP structure. The concepts of constructor overloading are used so that LiCL can accept various sets of the argument depends on the initialization of the object. The *CiC* class is written to show the component Info-Chunk object in the logical layer of the RFLP structure. The *LiCLGeometry* class

is written to show the geometry of the multidisciplinary product. Here, it could be possible for data retrieval of the product model and part model's STEP files in the LiCL class. In that case, LiCLGeometry constructor's arguments *part_info* and *assembly_info* are converted from string types into the STEP file format. Here, JSDAI API could be the possible approach to read and write the STEP file format. Then, LiCLGeometry object, affect zone and array of circumstances are used as a constructor argument for the LiCLSituation object. The *LiCLSituation* class is written to show the situation with a set of circumstances applicable to a LiCL object. The *LiCLProcess* class is written to show the process plane of the IC. It accepts String and Boolean values of processes as a constructor argument. The string values store the name of the processes whereas Boolean value stores the status of a process. The *LiCLConnector* is the enumeration class to store connector information. The get method returns the value of objects required to the main application. It is used in the next subsection.

```
// LiCL.java
/** This class is written to show the layer Info-Chunk
object of the logical layer in the RFLP structure*/
/**This class is written to show the layer Info-Chunk
object of the functional layer in the RFLP structure
*@param comp_name This parameter stores the name of a
component
*@param community_name This parameter stores the
community name of a component
*@param descrp_CiC This parameter stores the
description of a component
*@param contrib_product This parameter stores the
contribution of the component in the product modeling
*@param type_output This parameter stores the outputs
from the LiCL object
*@param type_input This parameter stores the inputs to
the LiCL object
*@param comp_connected This parameter stores the number
of connected components in a LiCL object
*@param connect This parameter stores information of
connector type. It could be Inner connector or Extended
connector
*@param components This parameter stores the array of
the CiC(Component InfoChunk) objects
*@param functionality This parameter stores the feature
of a LiCL object with LiCF type
*@param gmtry This parameter stores the geometry of the
components in a LiCL object
```

```
*@param situation This parameter stores the information
of influenced components and geometry of the components
in a LiCL object at the given situation
*@param process This parameter stores the process
involved in a LiCL object
*@param data_model This parameter stores the detail
description of a LICL object in the context of the
physical object such as process, geometry, and
situation
*/
package org.obuda.infochunklib.rflp;
public class LiCL {
private String comp_name, community_name, descrp_CiC,
contrib_product, type_input, type_output; int
comp_connected; LiCLConnector connect; CiC[] components
= null; LiCF functionality = null; LiCLGeometry gmtry =
null; LiCLSituation situation = null; LiCLProcess
process = null; LiCLDataModel data_model = null;
/**
*This constructor is used to initialize the LiCL object
without the information to the physical layer
*@param name_comp This parameter stores the name of a
component
*@param name_community This parameter stores the
community name of a component
*@param connected_comp This parameter stores the number
of connected components in a LiCL object
*@param product_contib This parameter stores the
contribution of the component in the product modeling
*@param input_type This parameter stores the inputs to
the LiCL object
*@param output_type This parameter stores the outputs
from the LiCL object
*@param affect_zone This parameter stores the
influenced components during the analysis in a LiCL
object
*@param part_info This parameter stores part
information in the geometry of a LiCL object
*@param assembly_info This parameter stores assembly
information in the geometry of a LiCL object
*@param form_features This parameter stores form
feature information in the geometry of a LiCL object
*@param circum This parameter stores the array of the
circumstance of a situation in a LiCL object
*@param arryCiC This parameter stores the array of the
CiC(Component InfoChunk) objects
```

```
*@param function This parameter stores the feature of a
LiCL object with LiCF type
*/
    public      LiCL(String      name_comp,      String
name_community,     int     connected_comp,     String
product_contib, String input_type, String output_type,
String   affect_zone,   String   part_info,   String
assembly_info, String form_features, String[] circum,
CiC[] arryCiC, LiCF function) {
comp_name = name_comp; comp_connected = connected_comp;
community_name  =  name_community;  contrib_product  =
product_contib; type_input = input_type; type_output =
output_type;   functionality   =   function;   connect
=connection;  components  =  arryCiC;  gmtry  =  new
LiCLGeometry(part_info, assembly_info, form_features);
situation  =  new  LiCLSituation(affect_zone,  circum,
gmtry);
    }
/**
*This constructor is used to initialize the LiCL object
with the information to the physical layer
*@param process_analysis This parameter stores the
status of the analysis process in a LiCL object
*@param process_effect This parameter stores the status
of the effect/contextual process in a LiCL object
*@param process_optimization This parameter stores the
status of the optimization process in a LiCL object
*@param value_analysis This parameter stores the array
of analysis process values in a LiCL object
*@param value_effect This parameter stores the array of
contextual process values in a LiCL object
*@param value_optimization This parameter stores the
array of optimization process values in a LiCL object
*@param connection This parameter stores information of
connector type. It could be Inner connector or Extended
connector
*@param contexual_PO This parameter stores knowledge of
contextual Physical object/s in a LiCL object
*@param connected_PO This parameter stores knowledge of
connected Physical object/s in a LiCL object
*/
    public      LiCL(String      name_comp,      String
name_community,     int     connected_comp,     String
product_contib, String input_type, String output_type,
String   affect_zone,   String   part_info,   String
assembly_info, String form_features, String[] circum,
Boolean   process_analysis,   Boolean   process_effect,
```

```
Boolean process_optimization, String[] value_analysis,
String[] value_effect, String[] value_optimization,
LiCLConnector connection, String contexual_PO, String
connected_PO, CiC[] arryCiC, LiCF function) {
comp_name=name_comp; community_name=name_community;
comp_connected = connected_comp; contrib_product =
product_contib; type_input = input_type; type_output =
output_type; functionality = function; connect
=connection; components = arryCiC;
gmtry = new LiCLGeometry(part_info, assembly_info,
form_features);
situation = new LiCLSituation(affect_zone, circum,
gmtry); process = new LiCLProcess(process_analysis,
process_effect, process_optimization, value_analysis,
value_effect, value_optimization); data_model = new
LiCLDataModel(contexual_PO,process,situation,connected_
PO, type_input, type_output);
    }

    private String getSituation() {
    return situation.affect_zone;}
    private String [] getCircumstances() {
    return situation.circumtnces ; }
    private CiC[] array_Components(){
    return components;}
    private LiCF getLiCF(){
    return functionality;}
    private LiCLProcess getProcessInfo(){
    return process;}}
```

6.2 Demonstration of Info-Chunk Objects in MAAD Structure

The *BiC* and *CxiC* class are the application classes for the behavior representation of the multidisciplinary product model as shown in the code below. The output is the graph between the components of various disciplines. It is the outcome of the process plane of the IC. The *BiC* class accepts the LiCL and LiCF objects as a constructor argument. Using the LiCL object, the LiCLProcess object can check the status of the analysis and optimization process. If the value is true, then it can generate the graph related to the process. The outcome of the Analysis process is shown in Fig. 7. The graph explains the displacement of the battery, starter and attenuator components w.r.t to time after the Thermal Analysis process. The outcome of the optimization process is shown in Fig. 8. The graph explains the voltage required w.r.t time for the optimized battery response.

```
//BiC Class
/**
This class is written to show the behavior Info-Chunk
object of the Behaviors layer in the MAAD structure
*@param bfunc This parameter initializes the LiCF
(Layer InfoChunk in the functional layer) object.
*@param blogic This parameter initializes the LiCL
(Layer InfoChunk in the logical layer) object.*/
public class BiC extends Application {
   LiCF bfunc = null;
   LiCL blogic = null;
/**
*This is the only constructor used to initialize the
BiC (Behavior InfoChunk) object with the information of
LiCF and LiCL object*/
   public BiC(LiCF funct, LiCL logic) {
   funct = bfunc;  logic = blogic; }

/*This method is used to generate the graph obtained
from the analysis and optimization process. The
parameters could be the components, parts or expected
changes in the assembly. */
  @Override
 public void start(Stage stage) {
if(blogic.getProcessInfo().isAnalysisProcess()){
//generate graph
}
if(blogic.getProcessInfo().isAnalysisProcess()){
//generate graph
}
if(blogic.getProcessInfo().isOptimizationProcess()){
//generate graph
}}
```

The *CxiC* class accepts LiCL object as a constructor argument. Using LiCL object, the LiCLProcess object can check the status of the effect (contextual) process. If the value is true, then it can generate the graph based on the contextual relationship between engineering objects. The outcome of Effect process is shown in Fig. 9 where contextual relation between attenuator and battery is explained by varying the battery output current with the attenuator speed. Here, XYChart class is used for generating the Line Chart graph.

```
//CxiC Class
/**
This class is written to show the Contexts Info-Chunk
object of the Behaviors layer in the MAAD structure
*@param blogic This parameter initialize the LiCL
(Layer InfoChunk in the logical layer) object */
public class CxiC extends Application {
   LiCL blogic = null;
/**
*This is the only constructor used to initialize the
CxiC (Contextual InfoChunk) object with the LiCL
object*/
  public CxiC(LiCL logic) {logic = blogic;}

/**This method is used to generate the graph obtained
from the contextual process. The parameters could be
the components, parts or expected changes in the
assembly. */
    @Override
    public void start (Stage stage) {
if(blogic.getProcessInfo().isEffectProcess()){
//generate graph}
}}
```

The *MAADStructure* class is the main method class that launches the application by calling the objects of BiC and CxiC objects.

```
//MAADStructure Class
/** This class is written to show the main application
of the InfoChunkLib API. */
public class MAADStructure {

 /** This method is used to start the main application
by calling the BiC and CxiC objects and generate the
graphs*/
    public static void main(String args[]) throws
Exception {
    Application.launch(BiC.class, args);
    new Thread(){
    Application.launch(CxiC.class, args);
    }}   }
```

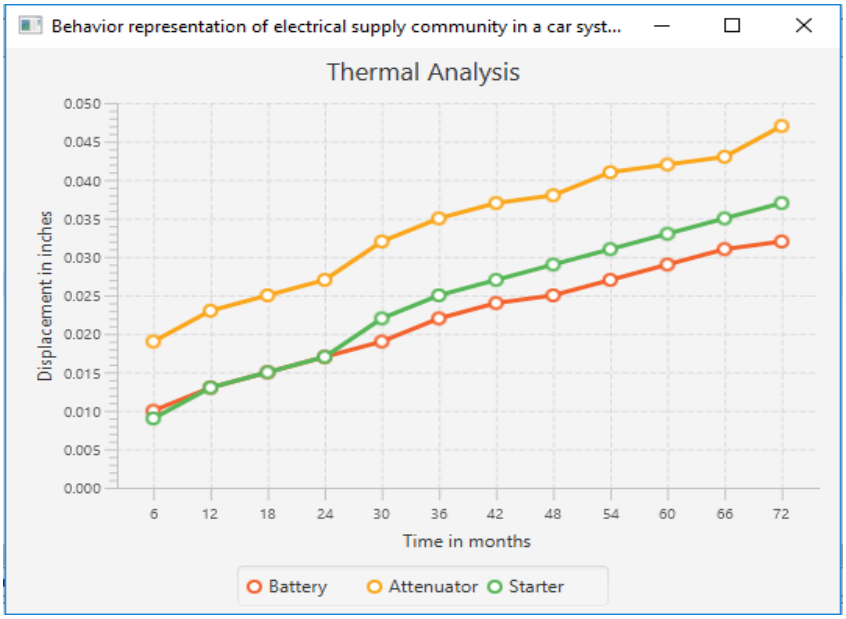

Figure 7
The graph of components after thermal analysis

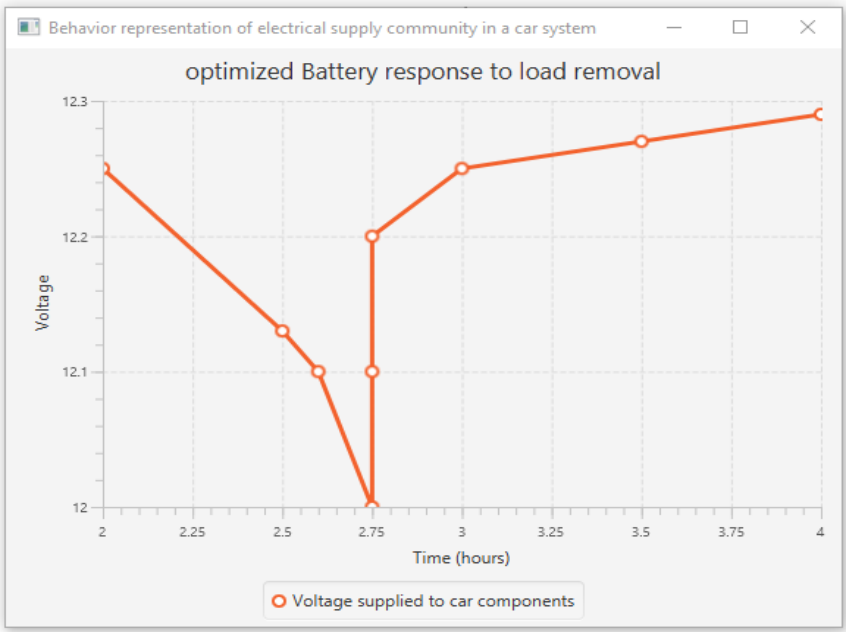

Figure 8
The graph of the battery component after the optimization process

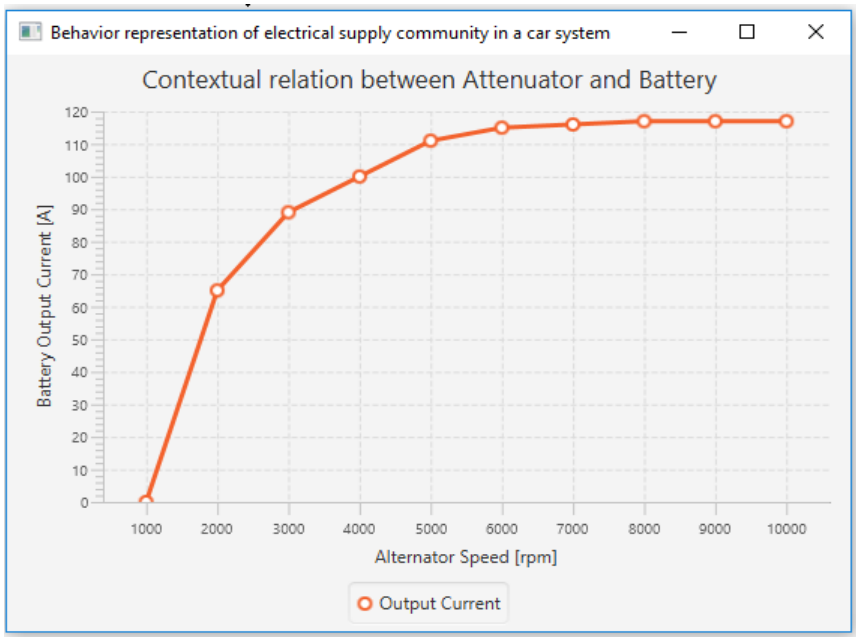

Figure 9
The graph between battery and attenuator component after effect/contextual process

6.3 Implementation of InfoChunkLib API in the Information Content

InformationContent class is the application which imports the InfoChunkLib API and handles the multidisciplinary product model. It could be a Java application or Web application. The output is stored in the database. As shown in the code below, SFiC objects and CiC objects are initialized first. Then, LiCF objects are initialized from SFiC objects and LiCL objects are initialized from CiC objects and LiCF objects. Then, BiC objects are initialized by LiCF objects and LiCL objects. CxiC objects are initialized by LiCL objects. Finally, the *MAADStructure* class is called by *InformationContent* arguments and graphs are generated for the behavior representation of multidisciplinary product model.

```
//InformationContent.java
/**This class is written to use InfoChunkLib API to
drive the multidisciplinary product model*/
import org.obuda.infochunklib.rflp.SFiC;
import org.obuda.infochunklib.rflp.SFiCLink;
import org.obuda.infochunklib.rflp.CiC;
```

```
import org.obuda.infochunklib.rflp. ConnectorCiC;
import org.obuda.infochunklib.rflp.LiCF;
import org.obuda.infochunklib.rflp. FunctionLink;
import org.obuda.infochunklib.rflp.LiCL;
import org.obuda.infochunklib.rflp.ConnectorLiC;
public class InformationContent{
public static void main(String args[]) throws
Exception{
//Extract SFiC object arguments information from the
functional layer and physical layer (.step file)
SFiC subfunc1 = new SFiC("To recharge the battery",
"Energize a field current that turns a rotor inside a
set of stators that can produce high current in
alternating directions", SubfunctionLink.DataFlow,
"Mechanical energy", "Electrical energy", "The
electrical system of a car is a closed circuit with an
independent power source the battery");

//Extract CiC object arguments information from the
logical layer and physical layer (.step file)}}
CiC comp1 = new CiC("Alternator", "Electrical supply",
"large BATT terminal connected to battery positive,
Relay Terminal connected to the connect to the dash
warning light, Sense Terminal connect the pigtail
directly to the BATT terminal", "provide power to the
car electrical system", subfunc1, "Magnet movement",
"Energy", "Battery", "Engine and Starter",
ConnectorCiC.Inner);
//Extract LiCF object arguments information from the
functional layer and physical layer (.step file)
LiCF funct = new LiCF("To power the car system", "The
battery provides juice to the starter. Then, the
alternator gives that battery the energy required to
power the car system", "Electrical Group",
FunctionLink.DataFlow, null, "Power", arrySFiC);
//Extract LiCL object arguments information from the
logical layer
LiCL logic = new LiCL("Electrical supply", "Electrical
Group", 3, "To supply power to Car system", "Mechanical
Energy", "Power", "Experienced_driver","Alternator,
Starter and Battery", assembly_info, null, true, false,
circumstnce, true, false, true, value_analysis_thermal,
null, value_optimization_global, ConnectorLiC.Extended,
"Lighting and signaling system", "Ignition electronic
system", arryCiC, func); }}
```

```
//Initialize BiC object and CxiC object from LiCL
object and LiCF object
BiC behav = new BiC(funct, logic);
CxiC context = new CxiC(logic);

//Call MAADStructure class for behavior representation
of multidisciplinary product model
String[] args = new String[0];
MAADStructure.main(args);}
```

6.4 Testing Phase of the Info-Chunk Objects

It is necessary to check the stored information in the Info-Chunk objects. In the OOP based language like Java, JUnit testing is a popular tool to check the behavior of an object. The behavior of a multidisciplinary product can be tested by varying the attributes and methods of the BiC and CxiC objects in the virtual environment. These values are compared with the values obtained from the physical environment. Further, formulas can be derived from the consistent values obtained from the virtual and physical environment.

Conclusion

This research work focuses on the behavior representation of a multidisciplinary product model by introducing Info-Chunk objects in the Information Content (IC). It started with the conversion of Info-Chunk entities into the Info-Chunk objects. Further, "InfoChunkLib" API is proposed based on the communication between the MAAD structure and RFLP structure in terms of LiC, BiC, and CxiC objects. Information Content (IC) is an application that imports InfoChunkLib API to handle and drive a multidisciplinary product model. The generated graph obtained from the IC evaluate the behavior of product components at various processes. The IC facilitate the HCI of multidisciplinary product model by initializing the parameters through the application. Info-Chunk objects provide necessary specification and knowledge representations to simulate the behavior of the complex multidisciplinary product model.

Future Work

A web server could be the next step for this research work, where a database is populated by the proposed API and a web application is used to access the IC. Dassault Systemes has implemented the RFLP structure in the CATIA V6 and 3DEXPERIENCE (3DXP) platforms for the multidisciplinary product model. Here, Dymola [16] is used to analyze the dynamic logical behavior and Modelica is used for logical and physical modeling of a product. The Java language and Modelica language are based on OOP concepts. Hence, API could be translated accordingly. The author could further update the API and database. Also, InfoChunkLib API can be extended and deployed in the IP.

Acknowledgment

The author gratefully acknowledges his supervisor, Dr. Horváth László, for guidance while writing this article.

References

[1] L. Horváth and I. J. Rudas: Towards the Information Content-driven Product Model, Proceedings of the IEEE International Conference on System of Systems Engineering, Singapore, 2-4 June 2008, pp. 1-6

[2] L. Horváth and I. J. Rudas: Systems engineering in product definition, Proceedings of the IEEE 13th International Symposium on Applied Machine Intelligence and Informatics (SAMI), Slovakia, 22-24 Jan. 2015, pp. 181-186

[3] H. F. Krikorian: Introduction to object-oriented systems engineering.1, Journal of IT Professional, V(2), pp. 38-42, 2003

[4] L. Horváth and I. J. Rudas: Multilevel Abstraction Based Self Control Method for Industrial PLM Model, Proceedings of the IEEE International Conference on Industrial Technology, South Korea, 26 Feb.-1 March 2014, pp. 695-700

[5] L. Horváth and I. J. Rudas: Active Driving Content in RFLP Structured Product Model, Recent Advances on Mechanics, Materials, Mechanical Engineering, and Chemical Engineering, MMMCE, Barcelona, 2015, pp. 123-131

[6] Peter Fritzson: Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach, Wiley-IEEE Press, John Wiley & Sons Inc, 2015, pp. 45-97

[7] Detterfelt, Jonas and Johansson, Gert: A UML Based Modeling Approach for Multi Domain System Products, Nordic Conference on Product Lifecycle Management - NordPLM, 2006, pp. 39-50

[8] John Stark: Product Lifecycle Management: 21st Century Paradigm for Product Realisation, Springer-Verlag, London, 2011, pp. 10-25

[9] Yatish Bathla: Conceptual Models of Information Content for Product Modeling, Acta Polytechnica Hungarica, XV (2), 2018, pp. 169-188

[10] Ou Y.: On Mapping Between UML and Entity-Relationship Model, The Unified Modeling Language, Schader M., Korthaus A. (eds), Springer Nature, Switzerland, 1998, pp. 45-57

[11] Bernhard Thalheim: Entity-Relationship Modeling: Foundations of Database Technology, Springer-Verlag, New York, 2000, pp. 124-145

[12] L. Horváth, J. Fodor, I. J. Rudas: Manufacturing Aspect of the IBCA Structure for Active Knowledge Content Representation in Product Model, Journal of IFAC- PapersOnLine, 48(3), 2015, pp. 1616-1621

[13] Yatish Bathla: Different types of process involved in the information content product model. In Proceedings of the IEEE 14th International Symposium on Intelligent Systems and Informatics (SISY), 2016, pp. 99-104

[14] L. Horváth and I. J. Rudas: Integrated Associative Modeling of Parts and their Machining Process by Features, Proceedings of the IEEE International Conference on Microelectronic Test Structures (ICMTS) conference, 2001, pp. 316-321

[15] Ian Sommerville: Software Engineering: 9th edition, Addison-Wesley, Pearson Education & Sons Inc, 2011, pp. 216-421

[16] Dassault Systemes AB: Dymola Dynamic Modeling Laboratory Getting started with Dymola. Dymola User Manual Volume 1, Dassault Syst emes, Lund, Sweden, 2013, pp. 23-48

[17] Yatish Bathla: Structured organization of Engineering Objects in the information content of the PLM system. In Proceedings of the IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI), 2016, pp. 473-478

[18] L. Horváth and I. J. Rudas: Behavior and Design Intent Based Product Modeling, Acta Polytechnica Hungarica, 1(2), 2004, pp. 17-34

[19] Yatish Bathla: Info-Chunk driven RFLP Structure based Product Model for Multidisciplinary Cyber Physical Systems, Proceedings of the IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY), 2018, pp. 000327-000332

[20] L. Horváth and I. J. Rudas: Bringing up product model to thinking of engineer, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 2008, pp. 1355-1360

[21] László Horváth: New methods on the way to intelligent modeling in computer integrated engineering. In Proceedings of the 36th Annual Conference on IEEE Industrial Electronics Society (IECON), 2010, pp. 1359-1364