Towards Automation of the FORM/BCS Method

Hayatou Oumarou[1], Ibrahim Moussa Mahamat[1], Donatien Koulla Moulla[1,2]

[1]*University of Maroua, P.O. Box. 46, Maroua, Cameroon*
[2]*University of South Africa, 28 Pioneer Avenue, Florida Park, 1709, South Africa*

Abstract

The software industry is facing more complex computer systems, with short development and sustainability issues. To deliver good software with these constraints, software reuse has become a central concept for minimizing design and realization costs. This study improves upon Feature-Oriented Reuse Method with Business Component Semantics (FORM/BCS), a software development method that produces adaptable architectures from reusable domain components. This is a promising method for reusable software assets and model creation. The objective of the FORM/BCS is to bring the industrial production chain to the software. This study proposes a model to automatically transform the FORM/BCS business subsystem component into a process business component. Two metamodels for business subsystems and process business components were developed. In addition, this study establishes correspondences between the source metamodel and target metamodel classes, transformation rules, and the instance of the source metamodel and generates the target metamodel instance. Detailed findings can help practitioners reduce software design costs and development time, and contribute to the advancement of knowledge in software engineering.

Keywords: *Software product line, model transformation, automation, FORM/BCS method.*

1. Introduction

The software industry faces a permanent crisis because of the quality, cost, difficult development process, and intangible nature of software. Software engineering has evolved to address new issues introduced by the penetration of computing into the industry. The problem is no longer to develop one software at a time but to design and develop a *line (or a family) of software.* To respond to these recurring problems, software engineers have developed new paradigms including procedural, object, component, and model technologies. One paradigm is Model-Driven Engineering (MDE) [1] using models. The MDE approaches describe systems under development and their environments at different levels of abstraction. These abstractions allow for the design of applications independent of the target platforms. The MDE provides a software development framework in which models move from a passive (contemplative) to an active (productive) state and become the first-class elements in the software development process [2]. As a continuation, many approaches have been developed, as in [3], [4], [5], [6]. Similarly, Fouda [7] proposed the Feature-Oriented Reuse Method with Business Component Semantics (FORM/BCS) method. Automating the FORM/BCS method allows easier maintenance, reusability, and flexibility in adapting to changes. Moreover, it enables developers to more efficiently deliver software with relevant business components. However, implementing the FORM/BCS method presents some limitations, including limited applicability (the method might be more suitable for certain types of projects or domains and may not be a one-size-fits-all solution) and added complexity to the development process, specifically for smaller projects. This study proposes a method that can

Corresponding author: *Hayatou Oumarou (oumarou.hayatou@univ-maroua.cm)*
Received: Received: *17 May 2023*; *Revised: 5 August 2023*; Accepted: *18 August 2023*; Published: *1 November 2023*

automatically transform a component with a high level of abstraction into a more tangible one. This saves time and improves quality and productivity during software development.

The remainder of the paper is organized as follows. Section 2 discusses the concepts of model transformation, model transformation automatization tools, and various model transformation approaches. Section 3 presents the FORM/BCS software product line method in detail. It then presents the specifications of the different assets in the FORM/BCS method. Section 4 presents the proposed meta-models (a metamodel of the subsystem business component, and a metamodel of the process business component of the FORM/BCS method). Then, it presents transformation rules to automatically convert a subsystem business component into a process business component. Section 5 evaluates and discusses the case study results. Section 6 concludes the study with a summary of the key findings.

2. Model transformation

Model transformation represents one of the significant challenges to be met from a technical viewpoint to consider the wide dissemination of MDE. The MDE fits naturally into the object approach and the evolution of the component models. The MDE is built around two fundamental concepts: models and transformations. Any production process can be considered a model linked by transformations.

Models are created for a specific purpose in this process and transformations produce new models [8]-[9]. From a general viewpoint, we call model transformation any artifact/program whose inputs and outputs are models. Automating transformations aims to make models more operational and increase development productivity using an MDE approach.

2.1. Definitions and terms clarification

To perform model transformations, we distinguished between endogenous and exogenous transformation*s*. A transformation is endogenous if the involved models come from the same metamodel; otherwise, it is called an exogenous or translation transformation [10].

a) Endogenous transformations

- *Optimization*: transformation, which aims to improve performance while maintaining semantics.
- *Refactoring*: transformation, which involves a change in the structure to improve certain aspects of the quality of the software, such as comprehension, maintenance, modularity, and reuse without changing the observable behavior.
- *Simplification or normalization*: transformation, whose goal is to reduce syntactic complexity.

a) Exogenous transformations

- *Synthesis*: transformation from a level of abstraction to a lower level of abstraction. A typical example of this is code generation.
- *Reverse engineering*: The process of analyzing a subject system to identify the system's components and their interrelationships and to create representations of the system in another form or at a higher level of abstraction. This is the reverse of the synthesis process described above.
- *Migration*:transformation of a program written in one language to another with the same level of abstraction.

Another important factor to consider in transformations is the abstraction level. Based on this, we distinguished between *horizontal* and *vertical* transformations. Horizontal transformation occurs when the source and target models are at the same level. In contrast [11]- [12], in vertical transformation, the models involved are of different levels of abstraction. A typical example of a vertical transformation is *refinement* [7].

The source and target models may or may not belong to the same *technological space* in a transformation. A technological space comprises a set of concepts, a body of knowledge, tools, skills, etc., defining an operational working context. For example, XML, MDA, and DBMS are technological spaces. When a transformation involves several technological spaces, import/export tools are required to bridge the gap between these different spaces.

2.2. Approaches and tools for model transformation

As previously mentioned, model transformations are important for software development in MDE. The effectiveness of this technology relies largely on the model transformations. Their application (or use) covers several aspects including [2]- [13]:

- Generation of lower-level or higher-level models;
- Synchronization of models;
- Reverse engineering;
- etc.

2.2.1. Transformation approaches

Several techniques have been used for model transformations. According to the classification proposed in [14]- [15], transformation approaches can be classified into two broad categories: "model to model" transformation approaches and "model to text" transformation approaches.

a) "Model to model" approaches

A "model to model" transformation, is defined by generating target models from one or more source models [16]. The level of abstraction can affect transformation classification. In this category, there are direct model manipulation approaches, relational approaches, approaches based on graph transformations, and approaches guided by the structure of models and hybrid approaches. For instance, Misbah et al. [11] proposed a metamodel of Z Notation to reduce comprehension overhead and facilitate model-to-model transformations. Through a case study, the proposed metamodel demonstrates its ability to facilitate shared understanding among stakeholders, leading to unambiguous requirements specifications that can also be automated for model-based development. However, the authors did not automate the metamodel for model-based development.

b) "Model to text" approaches

The "model to text" transformation, or model to code, is considered as a particular case of model-to-model transformations [17]. They often generate code in a text format for practical reasons related to the reuse of existing compilers. In this category, we distinguish two types of approaches: approaches based on "visitor" mechanism and approaches based on canvas or "templates."

2.2.2. Principles of model transformation

The transformation process comprises three stages as follows:

- Definition of transformation rules;
- Expression of transformation rules;
- Execution of transformation rules.

a) Definition of transformation rules

Given a source model in an L1 language (such as UML) and a target model in an L2 language (such as Java), this step involves developing a mapping of the concepts from L1 to L2 (e.g., a UML class corresponds to one or more Java classes). Thus, we employed a meta-modeling technique to establish a broad and generic rule base. Transformation rules are established between the source metamodel and the target metamodel between all source model concepts and that of the target model. The transformation process takes one or more models as input, conforming to the source metamodels. The process generates one or more additional models that adhere to one or more target metamodels by utilizing a predefined rule set.

b) Expression of transformation rules

To express the transformation rules, a rule-specification language is required. Transformation languages can be declarative, imperative, or hybrid [18]. In declarative programming, we describe the data and their constraints. Unlike a declarative program, an imperative program describes how a result can be obtained by imposing a series of actions that the machine must perform. A hybrid language combines both the declarative and imperative programming paradigms.

c) Execution of transformation rules

Once specified and expressed, rules require an execution engine to be performed. This engine takes a source model and metamodel, the target meta-model, and the transformation rules as inputs. This outputs the target model. Figure 1 illustrates the model transformation process.

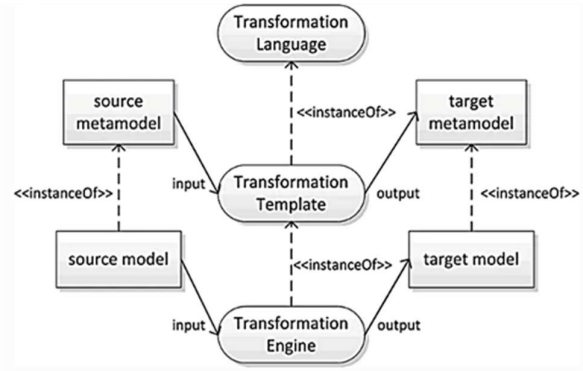

Figure 1. Basic diagram of a transformation [18].

2.2.3. Properties of transformations

The main properties that characterize model transformations are reversibility, traceability, reusability, scheduling, and modularity [8], [9].

- *Reversibility*: Transformations can be unidirectional or bidirectional. If a transformation is performed in only one direction, it is called a unidirectional transformation. Reversible transformations can be performed in both directions.
- *Traceability*: Traceability in transformations involves creating and recording links between the elements of the target models and those of the source models. Some transformation approaches do not offer a traceability mechanism, and the user must manage the trace links.
- *Reusability*: Reusability allows transformation rules to be reused in other model transformations.
- Scheduling: Scheduling represents the level of nesting (chaining) of transformation rules. Transformation rules can trigger other rules.
- *Modularity*: Modular transformation makes it possible to model the transformation rules by breaking down the problem. A model transformation language that supports modularity facilitates the reuse of transformation rules.

In this new perspective, the models occupy a space of the first level among the artifacts of system development. However, they must be sufficiently precise and rich to be interpreted or processed by machines. The system development process can then be seen as a sequence of partially ordered transformations, where each transformation takes one or more models as input and produces one or more output models, up to executable artifacts [19]. This transformation of the models is not an easy task. Therefore, it is necessary to have robust and flexible tools for managing templates and domain-specific languages for their transformations and manipulation throughout their life cycle.

3. The FORM/BCS method

The FORM/BCS [7] is a promising software product-line engineering method. The FORM/BCS has the advantage of integrating variability in diagrams or software development models, and explicitly provides variation points. This approach allows transposition of industrial production chains to the software world. The FORM/BCS [20] extends FORM to business components semantics. It transforms the design objects of a domain produced by FORM (feature models, subsystem models, process models, and modules) into "reusable" design objects of this domain, called "reusable enterprise components".

The FORM/BCS method is specific to product line engineering approaches in that its engineering process combines "reuse engineering" and "domain engineering" approaches. The horizontal FORM/BCS process, which corresponds to the "application engineering" approach, makes it possible to analyze a product line domain and develop fundamental reusable architectures. These abstract reusable models can be refined ("engineering by reuse" approach) using the vertical engineering process of FORM/BCS, with the aim of deriving the specific business components of an application domain of the domain that already has reusable business database components.

3.1. Horizontal engineering process of the FORM/BCS method

The objective of the horizontal engineering process is to analyze a domain to produce its reusable business components, which comprise the following [20]:

i. A functional business component;
ii. An enterprise business component of a sub-system;
iii. A business process component and;
iv. A module business component.

The horizontal engineering process has four independent activities:

- Domain analysis activity: The domain analysis activity, which is intuitive, produces a reusable functional business component. The produced component was stored in a reusable business component database.
- Business subsystem architecture component design activity: Considering the reusable functional business component selected from the database of reusable business components, the activity of designing business components of the sub-system architecture produces a reusable sub-system architecture business component, which is stored in a reusable business component subsystem architecture database.
- Process architecture business components design activity: This activity produces a set of business components and a reusable process architecture. These components are stored in a database of business components with reusable process architecture.
- Module architecture business component design activity: This activity produces a set of business architecture components for reusable modules. These components were stored in a database of the reusable business components of the module architecture.

3.2. The vertical engineering process of the FORM/BCS method

The objective of the vertical engineering process was to derive a reusable component database. The database is derived from a domain application with a reusable domain component database. The vertical engineering process has four independent activities: analysis of specific needs, design of specific subsystem architecture, design of specific process architecture, and design of specific module architecture [20].

By considering the application domain A of domain D and a commercial component of functionality F of D, the objective of the analysis of the specific needs of domain A is to derive a component functionality F' of A from F. For this, the activity, among others, makes choices in F to reduce the number of optional functionalities or grouping alternative functionalities in the solution decomposition. The derived functional business component F' is stored in the functional business component database of A.

By considering the business component functionality F' of an application domain A derived from a business component functionality F of a domain D and of an enterprise architecture subsystem S produced from D, the objective of the specific subsystem architecture design activity is to derive the sub-system architecture business component S' from A from F' and S. For this reason, the activity, among others, eliminates the functionalities in the subsystems of the business component of the architecture of subsystem S, which are absent in the basic business component F'.

Considering the architecture sub-system, an enterprise component S' of an application domain A derived from an architecture sub-system, an enterprise component S from a domain D, and a process architecture process P produced from D, the objective of the specific process architecture design activity is to derive a process architecture process P' from A from S' and P For this based on S', the activity adapts to the business component of the process architecture P.

By considering the process architecture process P' of an application domain A derived from architecture P, a domain D, and a module architecture module M produced from P, the objective of the specific module architecture design activity is to derive a business component architecture module M' from A from P' and M. Here, based on P', the activity adapts the architecture of enterprise architecture M. The possibility of successive refinements of reusable commercial components from one domain to more concrete components (vertical engineering) is the main improvement in the engineering application process of the Original FORM.

3.3. The formal model of FORM/BCS core assets

The asset specification of the FORM/BCS method was performed using the specification model defined by Ramadour and Cauvet [21]. Code 1 represents the extract of the model. Details of the specifications of the components of the FORM/BCS method can be found in [20].

Code 1. Extract from Specifications of business components.

```
ReusableBusinessComponent ==
[ name: Text; descriptor: Descriptor; realization:
Realization]
Descriptor == [intention : Intention ; context: Context ]
Intention == [action: EnginneeringActivity; target: Interest
]
Context == [domain: Domain; process: 𝔽 Context]
EngineeringActivity == AnalysisActivity ┃
DesignActivity
AnalysisActivity = {analyze,…}
DesignActivity = {design, decompose, describe,
specify…}
```

3.4. The case study context

Among the software product line methods, we note the Feature-Oriented Reuse Method (FORM) developed by Kang et al. [3]. The FORM extends Feature-Oriented Domain Analysis (FODA). It is a systematic method that emphasizes the similarities and variability of applications in a domain. These points are described in terms of "characteristics," and the obtained results are used to produce reference architectures. The main contribution of this approach is the decomposition of *the characteristic model* into layers, allowing the description of different points of view (e.g., services, operations, treatments, presentation, etc.) concerning the development of products. The model captures the points of similarities and differences, called the "*characteristic model*" and is used to support both the engineering of reusable domain artifacts and the development of applications using domain artifacts. Once a domain is described and explained in terms of similar and different "units" of processing, these are used to construct different "possible" configurations of reusable architectures [22]. Similarly, in 2009, Fouda and Amougou proposed an extension of the FORM**,** called Feature-Oriented Reuse Method with Business Component Semantics (FORM/BCS) method [7]. The specificities of FORM/BCS are as follows:

1. The integration of a business component semantics into the artefacts;
2. The proposition of rules for the systematic model's production;
3. It implicitly integrates *Model Driven Engineering*.

Therefore, our problem is a part of the automation *of the FORM/BCS method*. Indeed, it performs automatic component transformation from one level of abstraction to another using metamodels. In this document, our focus is on developing metamodels for the subsystem business component and the process business component of the FORM/BCS method, and the definition of the transformation rules, allowing the automated conversion of components [23]. These transformations are the horizontal engineering processes of the FORM/BCS Software Product Line method.

4. Automation of the FORM/BCS method

Models have become the central paradigm in the software industry, where researchers and practitioners enrich the models used in the design of applications and define new ones. This facilitates the creation of new technological spaces that are more suited to user needs and the different modeling stages necessary for product development. Thus, to obtain a product that meets user expectations, it is necessary to transform models from one level of abstraction to another, or from one technological space to another.

4.1. The business sub-system component of the FORM/BCS method

A subsystem business component of the FORM/BCS method is a reusable business component that describes a system in terms of abstract subsystems and relationships between them. Graphically, the solution is represented as a symmetric Boolean matrix, in which rows and columns represent the different subsystems of the business component. The values in the matrix indicate the existence of links between the subsystems. In Code 2, we implement the formal specification of the model of a business subsystem component.

The Architecture of a business subsystem component comprises a set of subsystems and the links between these subsystems. A subsystem comprises a name, characteristic, or functionality, and a subsystem has a string type and a *feature* of string type as parameters.

Code 2. Specifications of business subsystem component.

```
SubSystemBusinessComponent == [
    name: Name;
    descriptor: Descriptor;
    realization: Realization /
    ∀ ssbc: SubSystemBusinessComponent,
    (Solution (realization (CBMS)) ∈
    SubsystemArchitecture Adaptationpoints (realization
    (CBMS)) ∈ 𝔽 (SubSystem × 𝔽 SubSystem)]
    SubsystemArchitecture == [
    subsystems: SubSystem;
    links: 𝔽 (SubSystem × SubSystem)]
    SubSystem = 𝔽 Feature
```

A feature specifies business activities. An event applied to a set of *object* targets *(data) causes an activity*. This is a generalization in the sense of object-oriented analysis and decomposition. Thus, a feature takes as parameter the name of an activity, the object on which the activity operates, and decomposition. The decomposition yields all the components: Standard features that show the opportunity for reuse (optional features and all alternative functionality groups). We propose a meta-model represented in Figure 2 of the business component sub-system of the FORM/BCS method.

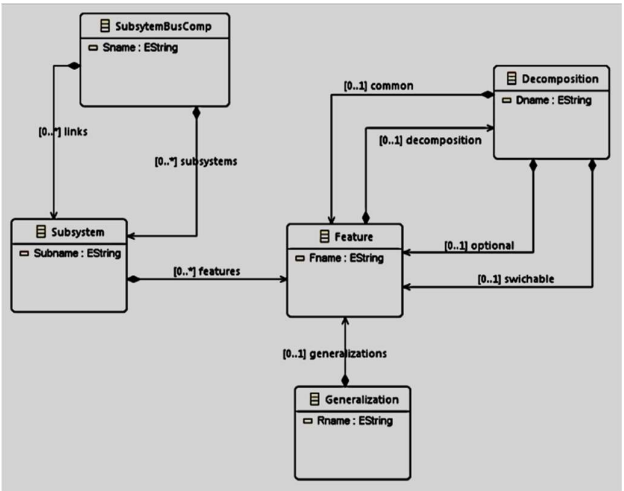

Figure 2. The meta-model of the sub-system business component.

Figure 2 presents the following classes:

- **SubsystemBusComp:** A subsystem business component has a name and comprises sub-systems and links between them.
- ***Sub-system:*** A subsystem comprises a name, a ***feature*** and a reference (link that links it to another subsystem).
- ***Feature:*** A ***Feature*** has a name and eventually ***decompositions***.
- ***Decomposition:*** Decomposition has a name. This is part of a characteristic. However, it can also comprise (sub) characteristics.
- ***Generalization:*** Generalization has a name and includes its characteristics.

4.2. Business process component of the FORM/BCS method

A business process architecture component is a reusable business component that represents a competitive structure in terms of concurrent business activities to which functional elements are assigned. Code 3 represents the process business component model specification.

A Process Architecture is a collection of business activities and objects (data). Business activities run on data and exchange messages across each other. They exchanged these messages as action calls or in a null environment.

Code 3. Specifications of process business component.

```
    ProcessBusinessComponent == [name: Name;
    descriptor: Descriptor;
    realization: Realization /]
    ∀ pbc: ProcessBusinessComponent,
(solution (realization (pbc)) ∈ ProcessArchitecture
Adaptationpoints (realization (pbc)) ∈ 𝔽
(BusinessActivity × 𝔽 BusinessActivity)]
    ProcessArchitecture == [
    tasks: 𝔽 BusinessActivity;
    datas: 𝔽 Class;
    messages: 𝔽 [name: Name;
    call: (BusinessActivity {null}) × (BusinessActivity
    {null})]]
```

Business activity is a set of activities (sub-activities) divided into three disjoint categories:

- All the joint commercial activities that indicate an opportunity for reuse (the typical character of the commercial activity),
- All options for the commercial activities of the activity (options of commercial activity), and
- Set of alternative business activity groups of the activity (the conversion capacity of the business activity).

The ability to have options and changes in business activities is its variability. A business activity is primitive (cannot be broken down). The metamodel of the process business component of the FORM/BCS method is shown Figure 3.

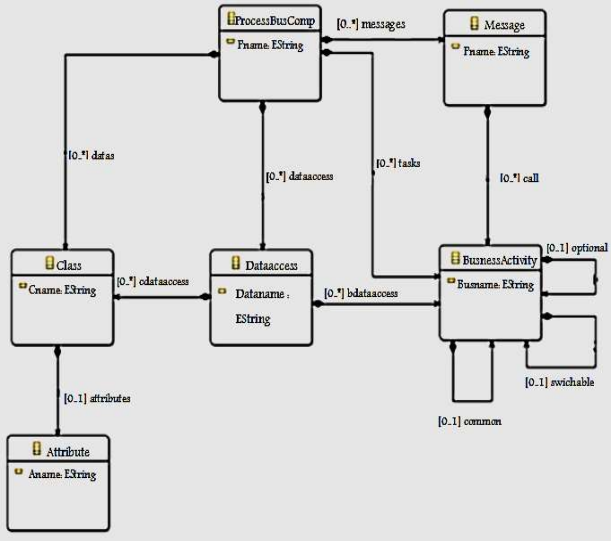

Figure 3. The business component meta-model.

Figure 3 presents the business component meta-model with the following classes:

- **ProcesBusComp:** A business process component includes a name, activities that communicate through messages and they store these in a database, classes (in the object-oriented sense), allowing the representation of processes.
- ***BusnessActivity:*** A ***BusnessActivity*** has a name and comprises the following three characteristics: Optional (represents the optional characteristics of the activities), Common (represents the typical characteristics of the activities), Switchable (represents the optional characteristics of the activities).
- ***Dataaccess*** class has ***Dname*** as a parameter: which represents the name of the stored activity.
- ***Message:*** This takes as a parameter ***Mname*** representing the name of an activity and a reference of each activity, which is a link between two activities with interactions.
- ***Class:*** represents an activity in the object-oriented sense (representing several similar activities).

4.3. Diagram of the subsystem business component model transformation into a process business component model

Figure 4 presents subsystem model to process model transformation where:

- **MMSubSystemBusinesComp** represents the metamodel of the subsystem business component;
- **MMProcessBusinesComp** represents the Meta model of the process business component,
- **MMSubSystemBusinesComp2MMProcessBusinesComp** represents the rules for transforming the model from a subsystem business component into a process business component, and
- **SubSystemBusinesComp** and **ProcessBusinesComp** represent the model of the subsystem business component and the model of the process business component, respectively.

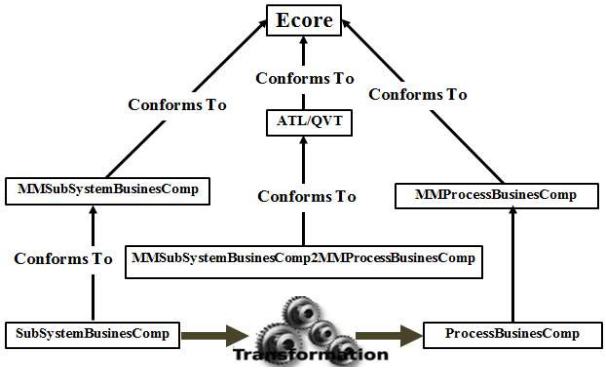

Figure 4. Subsystem model to process model transformation

The business process model transformation of the FORM/BCS method from the subsystem component model is performed at the same level of abstraction. This transformation occurred during the horizontal engineering process of the FORM/BCS method. To feed the translation engine, we need to bring the following information must be provided to the system:

- The meta-model of the source model : MM **SubSystemBusinesComp ;**
- The meta-model of the target model : MM **ProcessBusinesComp ;**
- Subsystem model (domain model conforming to subsystem metamodel or metamodel instance).
- The transformation code.
- These elements allow for automatic generation of the target model or an instance of the target model.

4.4. FORM/BCS business to process components transformations rules

We used the Atlas Transformation Language (ATL) to express the transformation rules. In addition to ATL, we also used the QVT tool to derive rules for transforming a business subsystem component into a process business component. We established correspondences between the elements of the source meta-model and those of the target meta-model.

4.4.1. Source and target Transformation

By considering the two proposed metamodels, the metamodel of the subsystem business component (of the source model), and the metamodel of the process business component (of the target model), we defined the following correspondences between the elements (classes):

1. *SubsystemBusComp to ProcessBusComp transformation:* Converts subsystems into activities. The links between subsystems become messages exchanged between business activities.
2. *Correspondence between the classes: Sub-system and BusnesActv transformation*: This transformation concert links into messages and transforms a subsystem into an activity.
3. *Sub-system to Message transformation*: This transformation converts each subsystem message as a link to identify the cooperating activities.
4. *Sub-system to Dataaccess transformation*: This transformation makes it possible to maintain the links established between the different subsystems. The links between the subsystems are represented using a Boolean matrix. The existence of links between subsystems is saved.
5. *Sub-system to Class transformation*: Subsystem transformation into a class makes it possible to standardize the subsystems. Each subsystem is assigned an attribute, which is a characteristic of the subsystem.
6. *Feature to BusnesActiv transformation*: The Class Feature is linked to an activity. Here, each characteristic becomes a specific activity.
7. *Feature to Message transformation*: The Feature class turns into a message. This means that they exchanged messages between the two activities with the same characteristics.
8. *Feature to Dataaccess transformation*: The Feature class becomes a Dataaccess class because the activities are stored and identified for their characteristics.
9. *Feature to Class transformation*: The Feature class transformation into class represents activities with the same characteristics.
10. *Decomposition to BusnesActiv transformation*: This transformation allows the conversion of the Decomposition class into the BusnesActiv class to allow the attribution of characteristics to activities, the characteristics a:: optional, switchable and standard.
11. *Decomposition to Message transformation*: This transformation allows each message to represent an activity with its characteristics.
12. *Decomposition to Dataaccess transformation*: This transformation allows access to activities with the same characteristics.
13. *Decomposition to Class transformation*: The transformation of the Decomposition class into a Class allows activities with the same characteristics to be grouped together.
14. *Generalization to BusnesActiv transformation*: This transformation allows the identification of commercial activities with the same characteristics.

15. *Generalization to Class transformation*: The transformation of the Generalization class into the Class class allows the attribution of a characteristic to the Class class (the attribute of the Class class becomes a characteristic).
16. *Generalization to Dataaccess transformation*: The transformation from Generalization class to Dataaccess class allows access to activities with the same characteristics.
17. *Generalization to Message transformation*: transforms the Generalization class into a Message class. Each message represented an activity. Each exchanged message between activities has the same generic characteristic.

This section presents our approach for transforming a subsystem business component into a process business component. In the next section, we present a case study that details the different classes of the source model into classes of the target model. Finally, we present the results obtained after executing these transformations.

5. Evaluation and discussion

The context of the case study is an actual model for which we can access the assets described in FORM/BCS. We explored the transition between the business subsystem component and the business process of the FORM/BCS method because they are our target, and we have specifications available. Therefore, it is more accessible to check the output of our tool. This choice of a system modeled by a third party allows us to reduce the influence we could have induced in the case study. Therefore, we evaluate our solution for the different cases presented in the Amougou thesis [22]. The modeling works concerned the retirement process for civil servants in Cameroon. This choice has several additional advantages. We discuss the quality of our results with the authors of real work using FORM/BCS. This is not a bias because we are working on modeling prior to this research, and we do not influence the way we described the solution; we only evaluated it.

5.1. Presentation of the results

We evaluate our three-step approach. First, we evaluate whether our approach can generate the business component's business process. Second, we evaluated the relevance of the results and compared them with the expected results. Third, we evaluated whether our approach helps developers improve the quality of their results and save time.

We used the career management specifications for Cameroonian civil servants proposed by Amougou [24]. We moved all management components - subsystem business components – to process components. This shows that our approach generates business processes from business components. Second, we compared and discussed the results obtained after automatic generation with those proposed by Amougou [24] and found that the two results are similar. This allows us to conclude that our approach allows automatic component generation.

In the third step, the transformation with our approach is instantaneous, whereas when manually performed, it is time consuming and can often lead to human errors.

5.2. Study limitations

As in any other empirical evaluation, the results of the case study are subject to threats of validity. The following notable threats were identified:

a) The systems studied may not be representative of a larger population of systems or other fields of application. This is always a complex threat to mitigate because there is little information about the system properties that are important for ensuring representativeness. A case study of replication in other systems should be performed. We believe that our approach is independent of the application field.
b) The approach does not allow traceability. There is a lack of means to determine when a transformation rule has been applied. This could allow us to know, for example, whether a sequence of application of the rules leads to a particular situation.
c) This approach does not allow for rolling back, which can result from a lack of traceability. Rolling back into a problem makes it possible to find a consistent system state.
d) The approach does not indicate priority/order when applying the transformation rule: there is no indication of how the rules are ordered. This

can prevent block situations and inconsistent states.

e) The approach does not clearly define any precondition for applying transformation rules, that is, there is no indication of the preconditions in the transformation rules. This can leave the system under modeling in an inconsistent state.

f) Internal threats to validity are linked to the implementation of the proposed approach. It is possible that our implementation of this approach contains errors that may affect the accuracy of our results. To counter this threat, we manually investigated a subset of the results and found no apparent errors.

However, the case study indicated the applicability of the proposed approach within the framework of FORM/BCS horizontal engineering. We believe that our approach is the first step towards automation of the FORM/BCS method and that the limits listed above are actual but do not hinder our approach contribution.

6. Conclusion

This study proposed a novel approach to automatically transform a high-level abstraction asset into a low-level abstraction using the FORM/BCS software engineering method. The metamodels of the subsystem business component and that of the proposed process business component enable seamless translation from the business subsystem to the process component. The proposed automation not only saves valuable time, but also enhances product quality and increases team productivity throughout the software development process. This study highlighted the automation potential of the FORM/BCS method and its applicability, and identified some shortcomings through an empirical evaluation. The findings of this study can offer valuable insights for researchers and practitioners, enabling them to improve product quality and the efficiency of the development process.

References

[1] C. A. Medeiros, A. Bandeira, P. H. Maia and P. Matheus, "MDE in the Wild: An Exploratory Analysis on What Developers are Discussing from Q&A Platforms," in *SBES '20: Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, 2020.

[2] A. Deniz, G. Vahid and D. Onur, "A survey on modeling and model-driven engineering practices in the embedded software industry," *Journal of Systems Architecture,* vol. 91, pp. 62-82, 2018.

[3] K. C. Kang, S. Kim, E. Shin and M. Huh, "FORM: A Feature Oriented Reuse Method With Domain-Specific Reference Architectures," *Annals of Software Engineering,* vol. 5, pp. 143-168, 1998.

[4] B. Young-Min, M. Zelalem, S. Yong-Jun and B. Doo-Hwan, "A Modeling Method for Model-based Analysis and Design of a System-of-Systems," in *27th Asia-Pacific Software Engineering Conference (APSEC)*, Singapore, 2020.

[5] J. Gonzalez-Huerta, A. Boubaker and H. Mili, "A business process re-engineering approach to transform BPMN models to software artifacts.," in *Aïmeur, E., Ruhi, U., Weiss, M. (eds) E-Technologies: Embracing the Internet of Things. MCETECH 2017. Lecture Notes in Business Information Processing*, vol. 289, Springer, Cham, 2017, pp. 170-184.

[6] C. B. Markus, V. Borozanov, S. Gokay and K.-H. Krempels, "Semi-automated Business Process Model Matching and Merging Considering Advanced Modeling Constraints," in *19th International Conferenceon Enterprise Information Systems (ICEIS 2017)*, 2017.

[7] N. M. Fouda and N. Amougou, "The Feature Oriented Reuse Method with Business Component Semantics," *International Journal of Computer Science and Applications,* vol. 6, no. 4, pp. 63-83, 2009.

[8] H. Barangi, R. Kolahdouz, B. Zamani and A. Khasseh, "Model-Driven Software Engineering: A Bibliometric Analysis," *Journal of Computing and Security,* vol. 8, no. 1, pp. 93-108, 2021.

[9] S. Shane and K. Wojtek, "Model transformation: the heart and soul of model-driven software development," *IEEE Software,* vol. 20, no. 5, pp. 43-45, 2003.

[10] P. D. F. C. Marco, "Model Assisted Software Development-a MDE-Based Software Development Methodology, PhD.," University of Hertfordshire, Hertfordshire, 2022.

[11] M. A. Misbah, A. Farooque, W. A. Muhammad and R. Yawar, "Formal Requirements Specification: Z Notation Meta Model Facilitating Model to Model Transformation," in *Proceedings of the 9th International Conference on Software and Information Engineering (ICSIE '20)*, New York, 2020.

[12] S. Umair, A. Farooque, U. H. Sami, W. A. Muhamad, H. B. Wasi and A. Anam, "A Model Driven Reverse Engineering Framework for Generating High Level UML Models From Java Source Code," *IEEE Access,* vol. 7, pp. 158931-158950, 2019.

[13] B. Losan, "Transformation de Modèles et Interopérabilité dans la conception des systèmes héterogènes sur puce à Base d'IP.PhD," Université Science et tecnologie de LILLE, Lille, 2006.

[14] B. Frank, S. David, M. Ed, E. Raymond and G. Timothy, Eclipse Modelling Framework, Addition Wesley, 2004.

[15] A. Bunker and G. Gopalakrishnan, "Formal Specification of the virtual component interface standard in the unified modeling language.," University of Utah, 2001. [Online]. Available: https://www.cs.utah.edu/docs/techreports/2001/pdf/UUCS-01-007.pdf.

[16] OMG, "MDA Guide," 12 June 2017. [Online]. Available: http: //www.omg.org/docs/omg/03-06-01.pdf.

[17] B. Annette, G. Gopalakrishnan and S. A. Mckee, "Formal hardware specification languages for protocol compliance verification," *ACM Transactions on Design Automation of Electronic Systems,* vol. 9, no. 1, pp. 1-32, 2004.

[18] Z. Zhu, Y. Lei, Q. Li and Y. Zhu, "Formalizing Model Transformations Within MDE.," in *Song, H., Jiang, D. (eds) Simulation Tools and Techniques. SIMUtools 2019. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol. 295, Springer, 2019, pp. 25-42.

[19] C. Bernard, S. Diaw and R. Lbath, "État de l'art sur le développement logiciel," Laboratoire IRIT, Toulouse, 2017.

[20] N. M. Fouda and N. Amougou, "A Rewriting System Based Operational Semantics fo the Feature Oriented Reuse Method," *International Journal of Software Engineering and Its Applications,* vol. 7, no. 6, pp. 41-60, 2013.

[21] C. Cauvet and R. Philippe, "Approach and Model for Business Components Specification," in *Hameurlain, A., Cicchetti, R., Traunmüller, R. (eds) Database and Expert Systems Applications. DEXA 2002. Lecture Notes in Computer Science*, vol. 2453, Berlin, Heidelberg, Springer, 2002.

[22] N. S. F. Merveille, "Conception et réalisation d'un éditeur des composants métiers caractéristique de la méthode FORM/BCS," Université de Douala, Douala, 2017.

[23] A. Ngoumou et M. Fouda Ndjodo, «A Command Oriented: Derivation Approach with product specific architecture optimization,» *International Journal of Software Engineering and Its Applications,* vol. 9, n° %12, pp. 23-40, 2015.

[24] A. Ngoumou, "Extension de la méthode FORM pour la Production des architectures adaptables de domaine," Universite de Yaounde I, Yaounde, 2011.