IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 25 July 2022, accepted 24 September 2022, date of publication 28 September 2022, date of current version 6 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3210530

RESEARCH ARTICLE

A Domain-Specific Language for the Document-Based Model-Driven Engineering of Business Applications

ONUR LEBLEBICI[1], GEYLANI KARDAS[2], AND TUGKAN TUGLULAR[3], (Member, IEEE)

[1]Univera, Inc., 35560 Izmir, Turkey
[2]International Computer Institute, Ege University, 35040 Izmir, Turkey
[3]Department of Computer Engineering, Izmir Institute of Technology, 35430 Izmir, Turkey

Corresponding author: Geylani Kardas (geylani.kardas@ege.edu.tr)

This work was supported by Univera Inc.

ABSTRACT To facilitate the development of business applications, a domain-specific language (DSL), called DARC, is introduced in this paper. Business documents including the descriptions of the responsibilities, authorizations, and collaborations, are used as the first-class entities during model-driven engineering (MDE) with DARC. Hence the implementation of the business applications can be automatically achieved from the corresponding document models. The evaluation of using DARC DSL for the development of commercial business software was performed in an international sales, logistics, and service solution provider company. The results showed that the code for all business documents and more than 50% of the responsibility descriptions composing the business applications could be generated automatically by modeling with DARC. Finally, according to the users' feedback, the assessment clearly revealed the adoption of DARC features in terms of the DSL quality characteristics, namely functional suitability, usability, reliability, maintainability, productivity, extensibility, compatibility, and expressiveness.

INDEX TERMS Business application, domain-specific language, DARC, model-driven engineering.

I. INTRODUCTION

Software developers may face various difficulties during the development of business applications [1]. Usually, these difficulties emerge from the continuously changing business requirements and environments as well as the processing and storage complexity of large amounts of data. Moreover, the need to support the interoperability with other systems, necessity of adapting to rapidly changing technologies, maintenance issues and providing improved user interfaces may also complicate the development of these applications. To cope with these challenges and facilitate the design and implementation of the business applications by raising the abstraction level of the development, many software companies follow model-driven engineering (MDE) [2] approaches mostly supported with the use of domain-specific languages (DSLs) [3]. From the practical perspective, many market analysis results highlight significant investments are being made by the vendors for the model-driven business applications as indicated in [4] and it is predicted that this trend will continue in the next several years in line with the widespread use of low-code development platforms [5].

Since the design process within a software project or a company is usually regularized with the document concept known to any parties involved such as customers, analysts, architects, developers, and testers, we believe that the documents, specifying the business applications with including the descriptions of the responsibilities, user authorizations and the collaborations with each other, can also be evaluated as the main entities of modeling these applications. Hence, MDE of the business applications can be possible by first creating business document models and then automatic code generation from these models. Although various studies already exist for the application of modeling techniques to the development of business applications (e.g. [6], [7], [8], [9], [10], [11]), none of them consider the use of business documents as the first-class elements of modeling combining the design of business responsibilities, authorizations and

The associate editor coordinating the review of this manuscript and approving it for publication was Amedeo Andreotti.

collaborations altogether and this restricts the wide adoption of these MDE techniques by the above mentioned parties who take significant role in the business application development. With the motivation of bridging this gap and supporting such an MDE process, a DSL, called DARC (an acronym for Document, Authorization, Responsibility, Collaboration), is introduced in this paper.

DARC specifies a design template with the document concept in its core. Developers use this template to describe various business model instances each conforming to a business document with its properties. The translational semantics of the DSL leads to the generation of code from these models to create the required business application. We describe and exemplify how MDE of business applications is possible with DARC DSL in the paper. Moreover, the results achieved from a comparative evaluation of using this new language are also discussed. The evaluation was performed within one of the international software companies having various well-known sales, logistics and service solutions in retail technologies worldwide.

The remainder of the paper is organized as follows: Section II gives the related work while Section III introduces the language constructs. MDE methodology based on using DARC DSL is discussed in Section IV with a case study. Evaluation of the language and the validity threats to this evaluation are given in Section V and Section VI, respectively. Finally, we conclude the paper with Section VII.

II. RELATED WORK

For more than two decades, the researchers have been working on developing new MDE techniques and/or DSLs to facilitate the design and implementation of enterprise business applications for various domains [2], [12]. We can group the related studies under two subjects: adaptation of the well-known generic modeling approaches and creating new languages or MDE methods.

In the automatic creation and management of the business applications, studies like [9], [10], and [13] consider benefiting from the notation and structures of the well-known generic modeling or workflow languages. For instance, Lu *et al.* [10] extends the OMG's Business Process Model and Notation (BPMN) [14] to specify interactions between business processes and fungible/non-fungible asset registries for the development of blockchain applications. Smart contract generation methods can be achieved in their MDE approach to automatically transform models into code. Likewise, Yousaf *et al.* [13] propose a model-based methodology to automatically generate test case documents from the initial collected web application requirements represented through the Interaction Flow Modeling Language (IFML) [15] models. MDE of business process rules is also considered in Soleymanzadeh *et al.*'s work [9] where the logic rules in JsonLogic structures, built on JavaScript Object Notation (JSON) [16] can be generated from the graphical business application models, as well as the recovery of visual models from the existing JsonLogic structures is provided with a web-based tool.

Taking into consideration the second group, we can see that the researchers propose new languages or MDE methods (e.g. [6], [8], [11], [17], [18], [19], [20]) more specific to the targeted business domains. For example, model-driven development of form-based or data-based business applications that interact with back-end systems is described in [17]. How the application-specific functions of business applications can be defined and generated automatically on a platform-independent level with a DSL and toolkit, called IIS $*$ CFunc-Lang, is described in [6] with a case study considering the development of an educational information system. Challenger *et al.* [18] investigate the model-driven development of composite content applications and propose a modeling language, called MDD4CCA, which has a metamodel composed of viewpoints such as form, navigation, workflow, and content. Inside MDD4CCA's IDE, it is possible to generate business applications running on Microsoft Sharepoint platform. Similarly, Bicevska *et al.* [21] enable business process modeling through the use of a DSL, and the definitions of executing business processes from models can be generated to include event-based architecture in their study.

Ferry *et al.* [19] provide a model-driven approach both to create cloud applications for the Infrastructure as a Service (IaaS) and to manage these applications independent from the cloud service providers. Another DSL, called MAML [8], enables both the model-driven development of business processes for different mobile applications and the automatic generation of local source code for these applications. On the other hand, the conceptual model-driven framework introduced in [20] for modeling and configuring DevOps engineering processes and platforms benefits from the model weaving mechanism to link elements from platform and process models. To build enterprise applications with an interpretive MDE approach, Oliveira *et al.* [7] define the system development steps from requirement specification to the delivery of the developed application to the customers. Efficiency and the financial gains of the proposed approach are shown in comparison with a classical generative MDE approach that considers code generation from business models. The effect of improving the notation of the business process technology built in OutSystems, which is one of the well-known low-code platforms for developing omnichannel enterprise applications, is investigated by Henriques *et al.* [22]. The study confirms that the process is effective and that the new notation supporting the MDE of business applications has a higher usability rating. Finally, MultiProLan DSML, introduced in [11], facilitates creation of a framework for the formal description and automatic execution of production processes. It is shown that production processing models designed using MultiProLan are suitable for the automatic generation of executable code.

DARC DSL, introduced in this paper, contributes to the above-mentioned noteworthy efforts by providing a new MDE methodology for business applications and it

differentiates from the current approaches by having a language syntax and translational semantics derived from the specifications of business documents and their properties which are the first-class entities of the modeling. To the best of our knowledge, DARC presents the first unique model of the business documents in which the responsibilities, authorizations, and collaborations on them can be designed all at once and the exact implementation of the business applications can be automatically generated from these designed models.

It is worth indicating that MDE based on the business documents as proposed by DARC can also be related to the studies like [23] and [24] where ontologies are used in business modeling. However, these studies mainly consider the system design for business processes (such as the overall description of service-oriented architectures or process modeling) while DARC directly aims at providing the automatic creation of business applications from design models reflecting the specifications of business documents. Similarly, responsibility modeling brought by DARC enables the representation of business functionalities such as triggering various events for the business documents while executing the applications. This kind of responsibility modeling is different from the modeling of distributing user responsibilities in the business process (e.g., see [25]).

III. DARC DSL

DARC provides a design template, which is general enough to capture the semantics of many different domains. Figure 1 shows this design template for a DARC-Document (DARC-D). A DARC-D stores data and operations as well as its contract(s) and workflow(s). There are five compartments on a DARC-D template, which are titled as Name, Responsibilities, Authorizations, Collaborations, and Fields.

The *name* represents a document. Names must be clear and fit with the mission of the document. A *field* is an attribute or related data along with its validation rules. The fields must be coherent with the responsibilities and the responsibilities must be coherent with the collaborations. However, a field does not necessarily have a direct connection with the responsibilities. For instance, a field called Explanation can be added to the document and it may not be related to any responsibility; it exists only as data in the document.

Responsibilities represent business functionality, which are attached to a document. Those functionalities can be triggered on before/after update, create, delete, persistence successful, and persistence failed events. DARC defines two kinds of responsibility. One is directly related to a document, and the second one listens to another document to perform some operations on the document. As can be seen in Figure 1, many responsibilities can be defined in a DARC document and given in a String format, i.e. numbered responsibilities in Figure 1 can be named in real implementations such as *LinePriceCalculator*, *OrderStatusPaid* and *StockLevelChanger*. For instance, to decrease the stock level, an order document should be listened to and when an order is

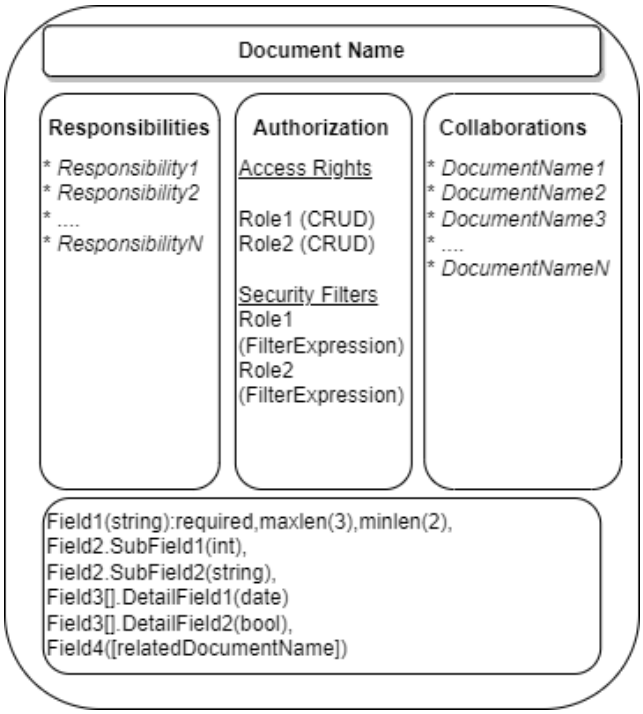

FIGURE 1. DARC Document Template.

received then corresponding products should be dropped out of stock. These can be defined in a responsibility instance called e.g. StockLevelChanger. The name of a responsibility should be written in an easily and clearly understandable fashion.

ACCESS CONTROL MATRIX				
Document	Create	Read	Update	Delete
Role1	✓	✓	✓	✓
Role2		✓		

a) DARC Access Control Matrix

SECURITY FILTERS	
Document	Filter Expression
Role1	IsActive = true && OrderedBy = @me
Role2	OrderDate > @now(-30)

b) DARC Security Filters

FIGURE 2. DARC authorization.

Authorizations specify operations allowed with respect to roles and filters. Operations that users can perform on the document are restricted through access rights by roles as seen in Figure 2(a). At the same time, access to the document records may be restricted through the security filters as given in Figure 2(b) among users with the same role. For instance, in Figure 2(b), the first security filter assures that the users playing the Role 1 have access only to the orders both being active and ordered by the same users. Similarly, the

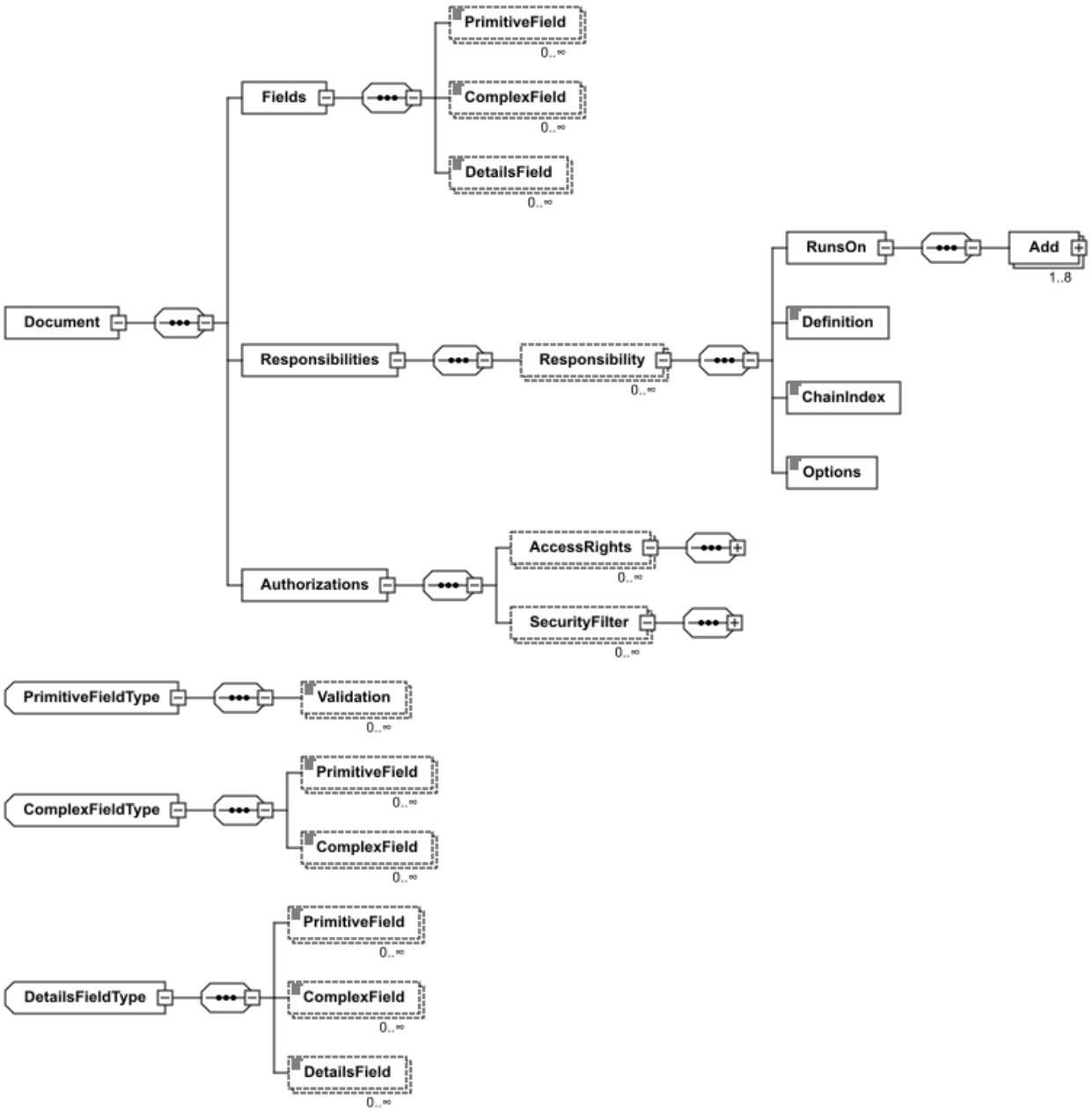

FIGURE 3. Fragment from the DARC metamodel.

second security filter allows the users having Role 2 to access only to the orders created in last 30 days. While reading, updating or deleting a document record, the responsibilities defined for that document are triggered based on the trigger events that have been registered in the order they are defined.

Collaborations are determined by checking out responsibilities and fields. One or more collaborations can be defined in a DARC document by indicating the names of the other DARC documents having a role in the related collaboration (see Figure 1 for the template example and Figure 6 and Figure 7 for the real implementation examples). For instance, a stock document has a responsibility to listen to an order document and therefore the stock has a collaboration with the order, i.e. the name of the document (*Order*) will be placed into the collaborations compartment of the stock document. Another example is that a product document has a productCategory field and therefore the product is in collaboration with the productCategory. Similarly, the document name *ProductCategory* is placed into the collaborations compartment of the product document. The Collaborations compartment outlines interactions among documents as well as their interdependencies. This compartment should be watched out for low coupling.

Syntax of the DARC-D template is based on a metamodel encoded in XML Schema to be processed by the current implementation of the DARC modeling tool (which will be discussed at the end of this section). However, the structured data model provided by this metamodel is also Eclipse Modeling Framework (EMF) [26] compliant and hence it can be used to create Eclipse-based modeling and code generation tools or DSL workbenches, i.e. DARC DSL can also be implemented easily on environments such as Sirius [27] or Papyrus [28]. Derivation of the metamodel entities and their relations was performed as the result of a feature-oriented domain analysis. With the collaboration of business application developers, both features of the document-based business applications and the dependencies among those features were determined. While studying the features, we also specified the system constraints as again will be discussed at the end of this section.

Each business document required for a business application is created as an instance model of the derived metamodel when the DARC-D template is used. Figure 3 shows a fragment from this metamodel showing the important meta-elements and their relations. These elements are explained below:

Document/Fields: Fields are attributes and/or related data of the documents. They can be primitive types like string, integer etc. or formed as complex types. If related data have to be represented as a list, there is also a detailed collection available. For instance, an order document has a primitive *order date* field, an *address* complex type field, and *order lines* detail collection fields. Fields also have validation annotations.

Document/Responsibilities: DARC defines the following characteristics for a responsibility:

- One responsibility belongs to only one document; one document can have more than one responsibility.
- Each responsibility must specify whether it's mandatory or not.
- Each responsibility must specify one or more trigger events. Available trigger events (for the RunsOn element) are:
    1. BeforeCreate
    2. AfterCreate
    3. BeforeUpdate
    4. AfterUpdate
    5. BeforeDelete
    6. AfterDelete
    7. PersistedSuccessfully
    8. PersistenceFailed
- If the execution order is important with respect to the triggering events, then the responsibility should define its chain index.

Document/Authorizations: Create, Read, Update, and Delete operations can be restricted depending on the role and the authenticated user's requests. There are two kinds of authorization for a document:

- Access Right restricts the roles for CRUD operations. Default behavior is restricted. To give access a role, both the name of the role and the permitted CRUD operations must be specified. For instance, if we specify CR for the customer role, that means a user who plays the customer role can create or read that document.
- Security Filter restricts the records that a role can access. Default behavior is allowed. To create a security filter for a role, both the name of the role and the filter expression must be specified. For instance, if we specify ''IsActive=true'' for the role customer, that means a user who is the member of the customer role can access only active records.

As discussed in [29], concrete syntax style of a DSL can be textual, graphical, form-/table based or textual with generation of visualizations. To facilitate both the use of DARC document templates and the formalization of DARC-D model instances according to the above abstract syntax definitions and constraints, we provide a form-/table based concrete syntax to the DARC DSL users via an Integrated Development Environment (IDE). Figure 4 shows a screenshot taken from this web-based IDE. This IDE was implemented on ASP.NET using C# language according to the well-known model-view-controller design pattern. As can be seen, documents, responsibilities, authorizations and collaborations needed for a business application are all created in DARC language's IDE by just preparing the related forms and tables which conform to the DARC metamodel specifications.

Constraint-checks and static semantics controls for all instance models are performed inside the IDE according to DARC DSL definitions, i.e., control on the multiplicity of model instances and checking naming conformances and types. In fact, developers do not need to know both the definition and the structure of all these controls since they are automatically applied on a DARC model without any user intervention.

We defined a series of model-to-text (M2T) transformation rules to generate business applications from DARC model instances. Table 1 lists the definition of some of these rules. Full specification of all these M2T rules can be found in the accompanying Mendeley data repository [30]. These M2T transformation rules were implemented as a C# program, which parses DARC models. The semantics of DARC language is provided over the application of these rules at run time on document models conforming to DARC syntax. Hence, the generation of.NET source code from DARC models is possible. For instance, it is possible to generate various .NET classes for document responsibilities and authorizations (see [30] for examples).

IV. MDE OF BUSINESS APPLICATIONS WITH DARC

Use of both DARC DSL and its IDE leads to the formalization of an MDE methodology in which developers can design and implement the enterprise business applications. The proposed methodology includes system modeling and automatic code

TABLE 1. Some M2T rules to generate .NET code from DARC instance models.

DARC Model Element	M2T Rule Description
Document.Name	Class name of the document and the DocumentAttribute.Name value. For example, If the documet is a shadow, then use the ShadowOf attribute for the DocumentAttribute.Name value For example; [Document("Order")] public class Order { .....
Document/Fields/PrimitiveField.MeasurementType	If the primitive field's FieldType is MeasurementType then set the value to the MeasurementMemberAttribute for the generated property. For example [MeasurementMember(EmeasurementTypes.Volume)] public double Capacity { get; set; } ....
Document/Responsibilities/Responsibility	Create a class that implements IDocumentResponsibility<T> or IDocumentResponsibility<TChange,TListen>
Document/Responsibilities/Responsibility/RunsOn	Adds multiple trigger event registrations for the generated responsibility. For example; If BeforeDelete and BeforeUpdate items are added to RunsOn list, then public EDocumentObservationEvents RunsWhen => EDocumentObservationEvents.BeforeDelete | EDocumentObservationEvents.BeforeUpdate; will be generated.

Document/Authorization	Authorization registrations defined under this path. Generates an ApplyAuthorization function and adds registration code for each access rights and security filter definitions.
	For example;
	DocumentAuthorization auth = new DocumentAuthorization();
	auth = repository.Query().FirstOrDefault(d=> d.Identity == "Role1" && d.DocumentName == "Order");
	auth = auth ?? new DocumentAuthorization();
	auth.DocumentName = "Order";
	auth.Identity = "Role1";
	auth.CreatePermission = EPermission.Allow;
	auth.ReadPermission = EPermission.Allow;
	auth.UpdatePermission = EPermission.Allow;
	auth.DeletePermission = EPermission.Deny;
	repository.Upsert(auth);

Document/Authorization/AccessRights.Role	Name of the role to set the access rights

generation for exact implementations. Figure 5 shows the steps followed and tools used in this methodology.

In the system modeling step (Step 1), a developer creates document models by using DARC concrete syntax inside the fully functional IDE. As discussed in Section III, this IDE does not only offer a computer-aided design for system modeling, but also supports various automatic controls which lead the designers into creating accurate models. The main outcome of this step is DARC document models (including the definitions of the authorizations, responsibilities, and collaborations as well as their inter-relations) conforming to DARC metamodel specifications, i.e. each DARC document model is the instance of the DARC metamodel.

The next step (Step 2) is the code generation from DARC instance models. The output of the previous step will be the input for the execution of this step. Here, DARC models are converted into the.NET code for the targeted system. The M2T transformation rules are automatically executed on the DARC instance models by the DARC DSL parser and the code is obtained for the implementation. The developer does not need to know about both the context of these M2T rules and their execution details. The code generation feature over the IDE is only selected by the developer without intervening in the rest of the code generation process. The result of the automatic code generation step is the source files, i.e. generated application.

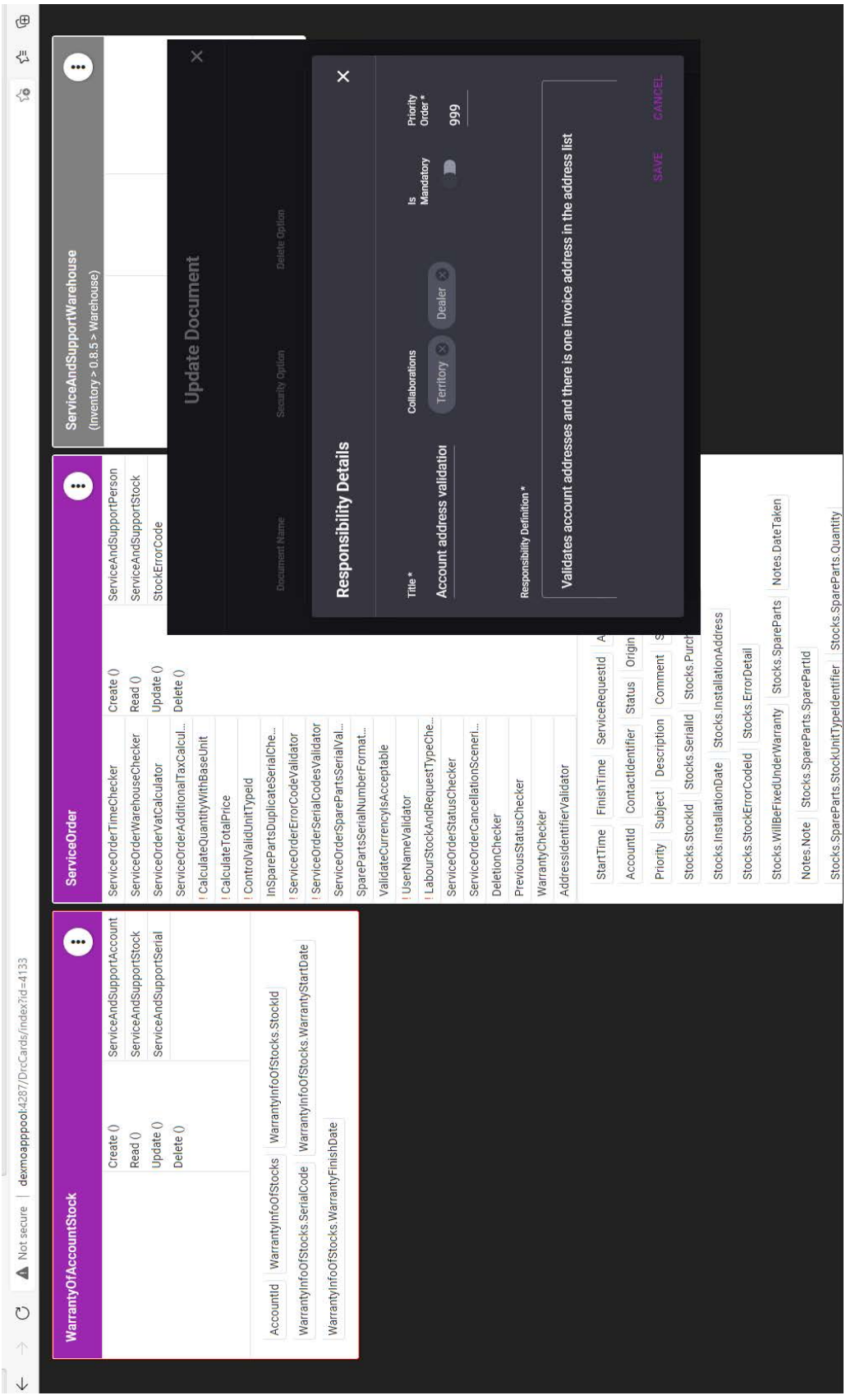

FIGURE 4. Screenshot from DARC IDE.

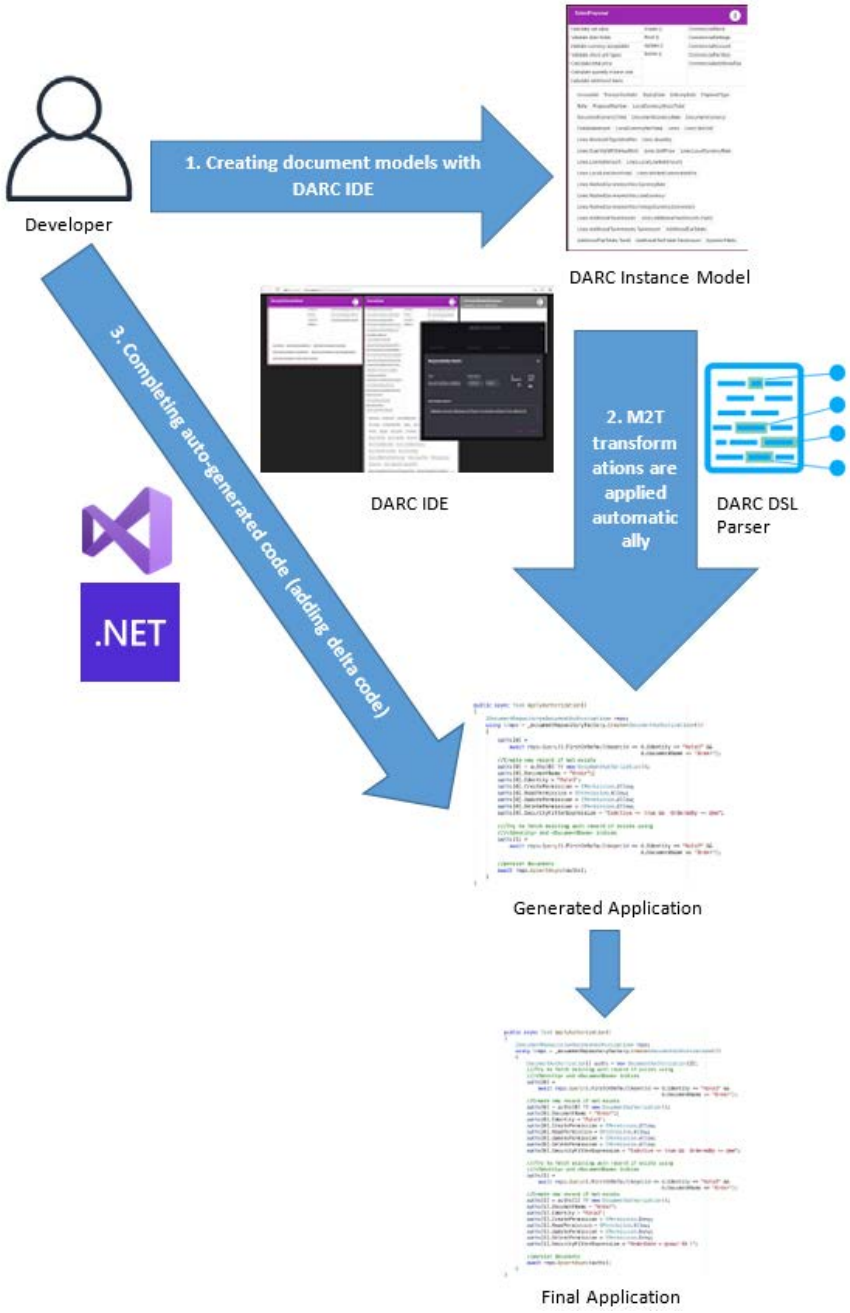

FIGURE 5. MDE with DARC.

Finally, in the third step, the developer completes the auto-generated code with using .NET framework to achieve the full implementation of the targeted business application. It is worth indicating that this step can be optional in some situations since DARC IDE succeeds in generating the complete forms of many model instances, i.e. the developer does not need to add delta code as will be discussed in Section V.

As a case study, we discuss MDE of an e-commerce sales and inventory management application using DARC DSL. Generated application is one of the important commercial products currently sold to over 120 enterprise customers.

First, we present only user stories that are related to the sales management and inventory management sub-domains. Then, we exemplify how documents of both sub-domains are modeled with DARC. Finally, to give some flavor of the automatic M2T transformations, the generation of the targeted business application's.NET code from the order document, which is an instance DARC model, is explained. All automatically generated .NET classes for this case study are available in [30].

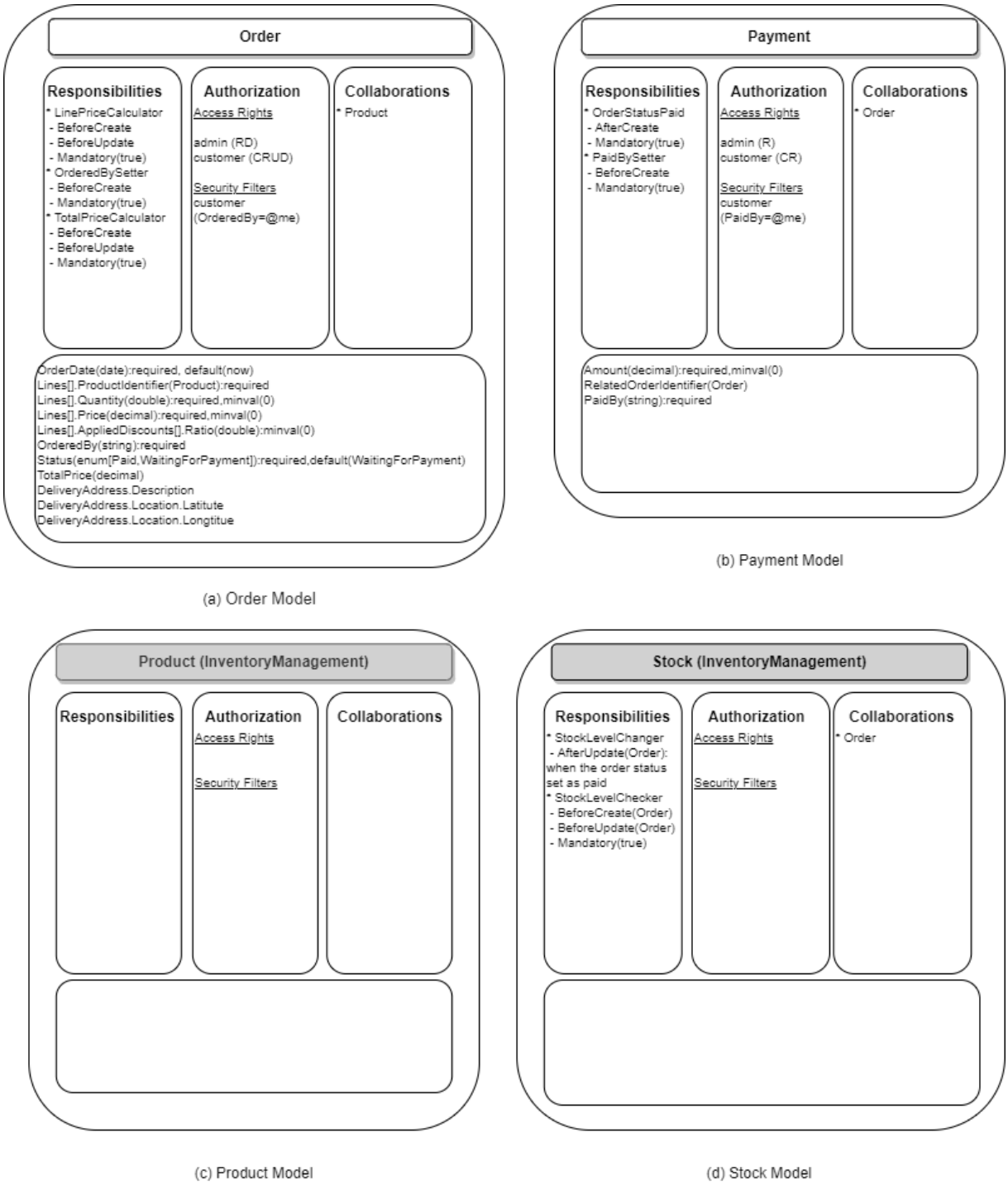

FIGURE 6. **DARC models designed for the sales management.**

We selected a limited number of user stories for an order business process for the sake of brevity. They are listed as follows:

- As an admin user, I should define products, so that customer users can buy them.
- As an admin user, I should define the product with name, price and tax ratio.
- As an admin user, I should define product categories and add products to these categories, so that users can look up products by categories.
- As an admin user, I should define discounts for products, so that I can get customers' attention.
- As an admin user, I should set the stock levels of products, so that I can control out-of-stocks and new stock orders.

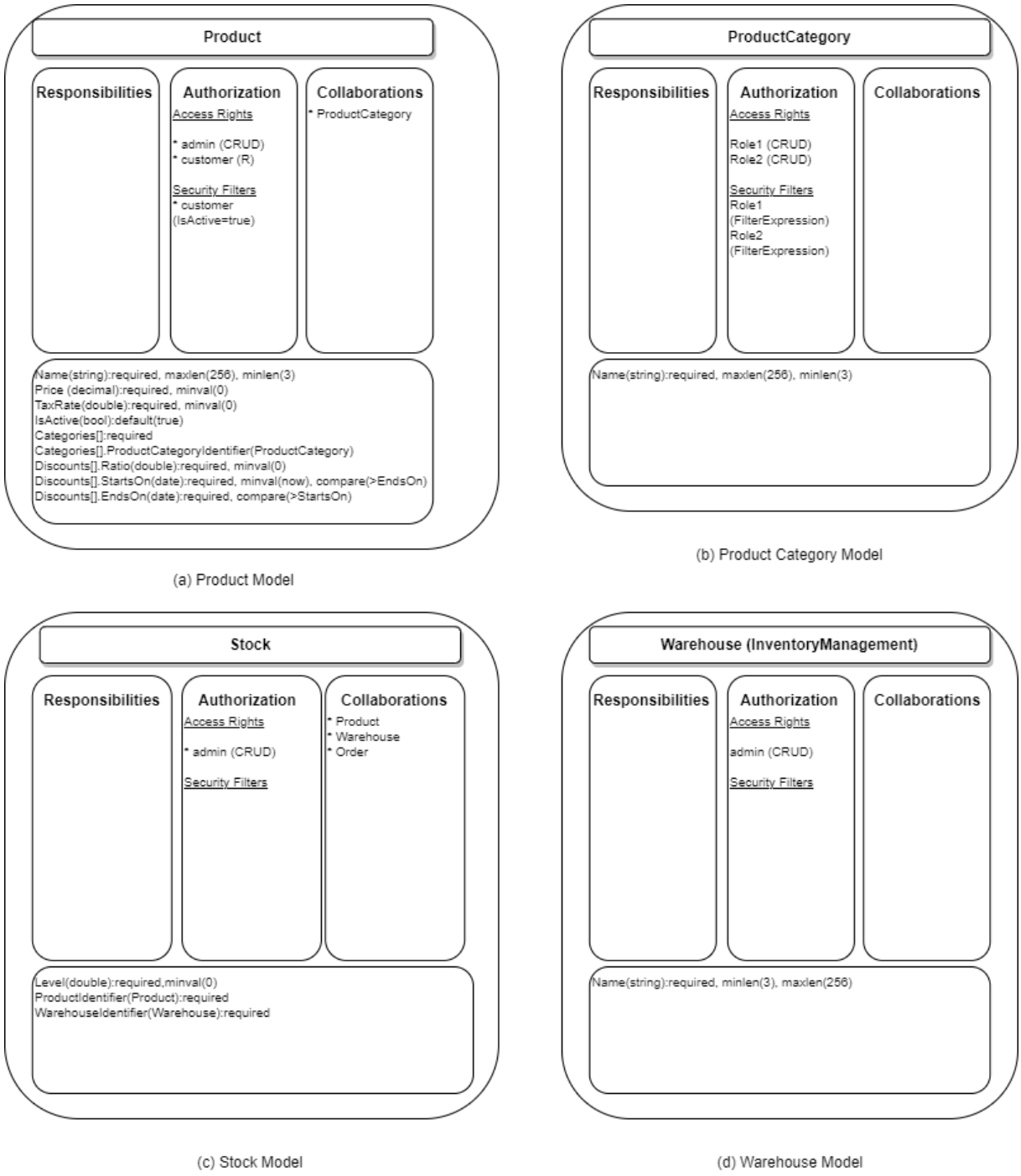

FIGURE 7. DARC models designed for the inventory management.

• As a customer user, I should list products, so that I can add them to my shopping cart.

• As a customer user, I should add any allowed amount of product to my shopping car, so that I can buy them.

• As a customer user, I should view my shopping summary for total amount and discounts.

• As a customer user, I should make payment for my shopping cart, so that I can purchase my goods.

Models designed with DARC DSL for this application are given in Figure 6 and 7. Each model is a DARC-D instance conforming to the template introduced in Section III and prepared according to the syntax and semantics definitions of DARC language as expected. Screenshots from the IDE for these models, which could not be shown here due to the space limitations, are available in [30]. When customer users want to purchase products, the order document is used for

this process. Basic order model for the sales management sub-domain is shown in Figure 6(a). It is worth indicating that a sub-domain in the DARC model is a set of coherent documents. Cohesion is the measure we suggest when grouping documents into a sub-domain. A complementary inventory management sub-domain presented in Figure 7 is also necessary to fulfill the requirements defined in user stories. Inventory management sub-domain includes documents, with which order documents collaborate.

The responsibilities defined on this model for keeping the information of the order can be accessed through the fields and the roles that are authorized to perform necessary operations on this document. When a new order record is created, a responsibility that updates the stock level of the product can be seen on the Stock model (Figure 6(d)). For instance, when a customer wants to create a new order record, price calculations are performed for the products (Figure 6(c) and 7(a)) to be purchased through LinePriceCalculator responsibility (see Order model in Figure 6(a)), the authenticated user is set as the person who placed this order through OrderedBySetter, and the total price information of the order is calculated through TotalPriceCalculator and it is set on the document. Then, StockLevelChecker responsibility, which is specified on the Stock model given in Figure 6(d) and 7(c), checks the stock levels for the products purchased and their categories (Figure 7(b)), and if there are insufficient stocks, the user is warned. Likewise, when a customer wants to update an order (who can only operate on their own orders), this time LinePriceCalculator and TotalPriceCalculator responsibilities come into play. If there are newly added, removed, or modified products, the calculations related to them are performed again. Then StockLevelChecker, which is defined on the Stock model, is activated again to control the stock levels for the products to be purchased.

Finally, in case the customer pays through the Payment document to complete the purchase, PaidBySettter responsibility on the Payment model (Figure 6(b)) sets the authenticated user as the payer and updates the order status given by the customer paid with the OrderStatusPaid responsibility.

The order update process triggers update responsibility on the Order document (Figure 6(a)) again, and stock level checks are made once again with line price and total price calculations. Then, the StockLevelChanger responsibility specified on the Stock model (Figure 6(d)) for the Order document, whose Status is paid, is activated, and it updates the stock levels via Warehouse model (Figure 7(d)) for the ordered products. The order update process triggers update responsibility on the Order document (Figure 6(a)) again, and stock level checks are made one more time with line price and total price calculations. Then, the StockLevelChanger responsibility specified on the Stock model for the Order document, of which Status is paid, is activated, and it updates the stock levels for the ordered products.

After the design of the application documents in these two sub-domains using DARC DSL, M2T transformation rules are executed on these DARC models to obtain corresponding .NET code automatically. Figure 8 includes a fragment of the code generated for the Order model

```
namespace DARC.SalesManagement
{
    public enum EOrderStatus { WaitingForPayment = 1, Paid = 2 }

    [Document("Order")]
    public class Order : DocumentBase
    {
        [Required]
        [DateTimeMember(DefaultValue = 0)]
        public DateTime OrderDate { get; set; }
        [Required]
        [DocumentDetailMember]
        public DocumentDetailElementCollection<OrderLine> Lines { get; set; }
        [Required]
        [StringMember]
        public string OrderedBy { get; set; }
        [Required]
        [EnumMember(DefaultValue = EOrderStatus.WaitingForPayment)]
        public EOrderStatus Status { get; set; }
        [DecimalMember]
        public decimal TotalPrice { get; set; }
        [Required]
        [ComplexTypeMember]
        public Address DeliveryAddress { get; set; }
    }
    public class OrderLine : DocumentDetailElementBase
    {
        [Required]
        [RelationMember(nameof(Product))]
        public string ProductIdentifier { get; set; }
        public ShadowProduct Product { get; set; }

        [MinValue(0)]
        [Required]
        [DoubleMember]
        public double Quantity { get; set; }
        [MinValue(0)]
        [Required]
        [DecimalMember]
        public decimal Price { get; set; }
        [DocumentDetailMember]
        public DocumentDetailElementCollection<OrderLineDiscount> Discounts { get; set; }
    }
    public class OrderLineDiscount : DocumentDetailElementBase
    {
        [DoubleMember]
        public double Ratio { get; set; }
    }
    public class Address : ComplexTypeElementBase
    {
        [StringMember]
        public string Description { get; set; }
        [ComplexTypeMember]
        public GeoLocation Location { get; set; }
    }
    public class GeoLocation : ComplexTypeElementBase
    {
        [LongMember]
        public long Latitute { get; set; }
        [LongMember]
        public long Longtitute { get; set; }
    }
}
```

FIGURE 8. Excerpt from the code generated for the order model.

```
{
  "OrderDate" : "",
  "OrderedBy" : "onur.leblebici",
  "Status" : "WaitingForPayment",
  "TotalPrice" : 37.67,
  "DeliveryAddress" : {
    "Description" : "Ordu Blv Karşıyaka / İzmir",
    "Location" : {
      "Latitute" : 38.321044,
      "Longitude" : 26.640581
    }
  },
  Lines : [
    {
      "ProductIdentifier" : "224e2504-cf51-4f04-ba53-09ec3ed83fd6",
      "Quantity" : 3,
      "Price" : 12.1
      "Discounts" : [
        {
          "Ratio" : 10.0
        }
      ]
    },
    {
      "ProductIdentifier" : "dcf23773-60a7-46be-bc35-2388a5d5b31e",
      "Quantity" : 1,
      "Price" : 5
    }
  ]
}
```

FIGURE 9. Json sample for the order model.

of the sales and inventory management application. The pseudocode of the executed model transformation rules is given inside the descriptions for the corresponding rules as previously listed in Table 1. For instance, the class for the Order Document in Figure 8 is generated by running the model transformation algorithms defined for the DARC model elements Document.Name, Document/Fields/PrimitiveField.MeasurementType and Document/Authorization/AccessRights.Role (see Table 1). The order document and all required fields of this document are determined by the DARC DSL parser and code for them is generated by executing the transformations defined for these DARC model elements (as discussed in Section III). Complete specification of the executed transformation rules and the code generated for all above discussed DARC instance models can be found in [30]. When this generated code for the order application is executed and a new order instance is created, it will be stored in a NoSQL document-oriented database using Json format. A Json sample for this order model is given in Figure 9.

V. EVALUATION

A comparative evaluation of using DARC DSL and its IDE during MDE of business applications was performed in this study. For the construction of the evaluation study and the assessment of the achieved artifacts, we followed the multi-case evaluation method proposed in [31] which we previously applied in the assessment of various DSLs / DSMLs and MDE processes such as [32], [33], [34], and [35] for different industrial domains. Quantitative analysis and qualitative assessment steps of this method were updated and improved in this study for the usability evaluation of DARC as will be discussed below.

The scope of our evaluation here covers Development Sub-dimension (under Execution Dimension) and User Perspective Sub-dimension (under Quality Dimension) of Challenger *et al.*'s [31] method. Therefore, the evaluation criteria pertaining to these dimensions, called Output Performance (Generation Performance) and Qualitative assessment by a questionnaire, are taken into consideration. We aimed at finding answers to the following research questions (RQs):

RQ1: Is it efficient to use DARC in MDE for business applications?

RQ2: Do developers adopt using DARC?

To find answers for the above RQs, our evaluation consists of two parts: 1) quantitative analysis, including generation performance evaluation 2) qualitative assessment within user perspective.

The evaluation was performed in Univera Company [36] (hereafter shortly Univera), which produces various business software for sales, logistics and service processes and offers solutions to more than 25000 customers worldwide for the management of multi-channel sales, partners, mobile teams, warehouses, production, field data collection and procurements. According to the Promotion Optimization Institute (POI)'s Retail Execution Reports [37], Univera ranks among the world's best companies in retail technologies regularly since 2016.

Ten software developers working in the R&D center of the company voluntarily participated in our study as the evaluators. All evaluators had at least a B.Sc. degree in computer engineering / software engineering except three of them have B.Sc. in electrical engineering, mathematics and business administration respectively. Two of them also hold M.Sc. degrees in computer engineering while one of them was a Ph.D. candidate in computer engineering at the time of this evaluation. All evaluators had an average of about 9 years of software development experience in different industries varying from 15 months to 21 years. Moreover, they had an average of approximately 7.5 years of experience on developing business processes / enterprise automation software. Although they were mostly familiar with the UML, none of them previously used MDE techniques or had an experience on using DSLs / DSMLs before using DARC.

A. QUANTITATIVE ANALYSIS

To answer RQ1, the generation performance of DARC was calculated. For this purpose, we considered the production of documents and responsibilities for an enterprise field service application used by one of Univera's customers, one of the biggest multinational household appliances manufacturers, active in more than 100 countries.

Development of six different documents for the above-mentioned business application with varying complexities were selected as the case studies. They are briefly described below:

1. AccountProducts: Description of the customer products are made into this document. Records can be added into this document via predefined customer cards or templates for mapping between the customers and the products. Data includes customer code, product code, serial number (if any), setup date and purchase date.

2. ServiceAndWarrantyAgreement: It provides the creation of service contracts or access to existing contracts for a specific customer. Definition of the warranties, creation of the automatic service orders and the reservations settings are all provided by these documents.

3. ServiceOrder: A service order is created after a service request is made and planned. Such orders are directed to the ServiceOrder documents automatically. These documents only enable editing and view operations for the orders.

4. ServiceOrderLabor: Labor addition for a related record is made inside this document via the service orders screen. It is also possible to list the labor entries mapped with the product groups

5. ServiceRequest: A request for a service is created according to a malfunction or maintenance reported by a customer. Related records also include the device information. If this request is moved to the planning step, it turns into a service order. Update operations are also possible with service request documents.

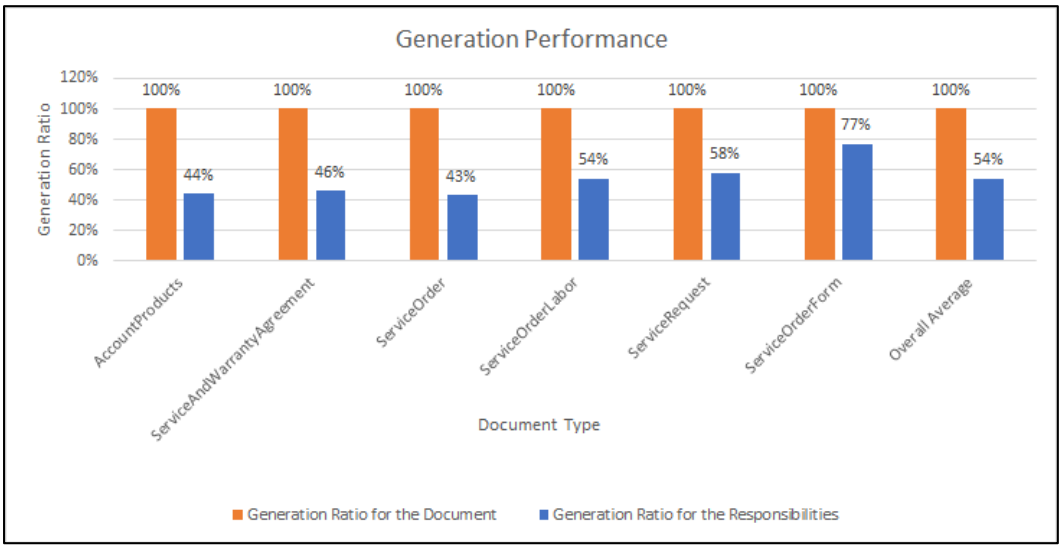

FIGURE 10. Percentage of the auto-generated LoC for the business application documents and responsibilities.

6. ServiceOrderForm: Service request forms, which are usually filled from the mobile applications during the field service operations, are listed here. It is also possible to add new records via the web interface. Listed forms can be edited for further use based on the service form templates.

All evaluators used DARC to develop documents and automatically generate the corresponding code for the responsibilities required for each case study and then completed this code to build the system. Performance evaluation is fulfilled by comparing the percentage of artifacts automatically generated and manually developed. These artifacts are Lines of Code (LoC) generated both for the business application documents and the related responsibilities. The bar chart in Figure 10 shows the comparison results for generating both documents and responsibilities of all case studies. The LoCs listed for each case study is the average of all developers in the evaluator group. For instance, all evaluators obtained auto-generated code depending on their DARC models for the ServiceOrderForm document's responsibilities with varying ratios. 77% LoC is their average which means they needed to manually add 23% LoC on average to complete the description of the related responsibilities. The Overall Average (shown at the rightmost) is the LoC average obtained from all case studies.

As can be seen from Figure 10, DARC DSL succeeded in generating all documents of this enterprise field service application completely from all user design models in this study. That means document descriptions made by all evaluators using DARC syntax can be automatically converted to the corresponding code. However, generated LoC for the responsibilities for each document varies between 43% and 77%. The main reason for having a low ratio of LoC generation for responsibilities in comparison with the documents is the need for the developers' intervention to complete some of the complex logic rules for the responsibilities (e.g. stock control) since they cannot be fully formalized just by modeling with the current implementation of DARC. Moreover, we realized that the number of the required responsibilities and the diversity seen in their types also affected the LoC generation performance. For instance, the ServiceOrder document required the creation of 33 different responsibilities and 43% of them were automatically generated on average from the DARC responsibility models created by the developers who participated in our evaluation. On the other hand, a big portion of the responsibilities (approximately 77%) for the Service OrderForm document were generated just by modeling in DARC IDE since it was enough to create 5 different responsibilities for this document. In addition, the developers' knowledge and experience on the business domain, comprehensiveness of each document as well as the quality of models created by the developers naturally have an effect on the generated LoCs. In order to keep this effect minimum, we conducted this multi-case evaluation with varying document and responsibility structure complexities instead of considering an application having only one document.

Considering all measurements made during this evaluation, we can conclude that, on the average, more than half of the responsibilities for a business application can be automatically created by just modeling with DARC while use of this language enables generating 100% of the business document descriptions. The result of this evaluation also leads us having some clues on the accuracy of the generated code in comparison with the code written manually. Business documents automatically generated by only using DARC are completely same with the ones achieved by coding manually, i.e. obtained business documents are syntactically same for both processes. Moreover, code generated from DARC models for the description of the business documents and responsibilities is automatically compiled and directly executed for all of the

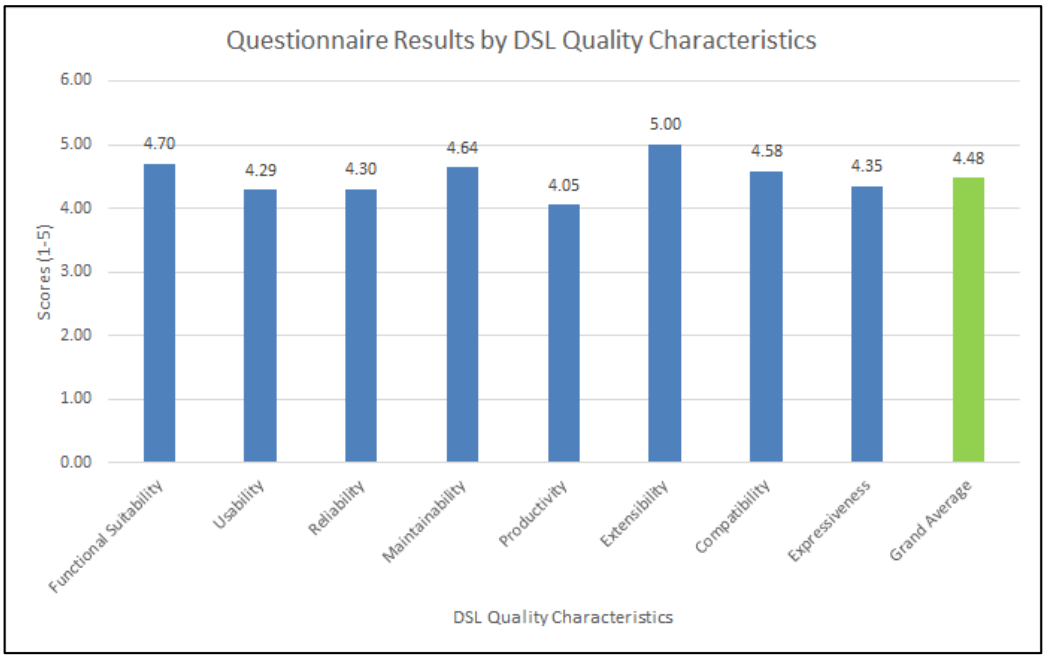

FIGURE 11. Average scores received for the DSL quality characteristics.

case studies just like the code developed manually without using DARC.

B. QUALITATIVE ASSESSMENT

The software developers in Univera, participating in this evaluation, were also requested to answer a questionnaire after they experienced using DARC in the above discussed case studies. The questionnaire has two parts: 1) scoring DARC according to a set of DSL characteristics and 2) answering 6 open ended questions to criticize the usability of DARC and its IDE. Achieved scores and feedback gained from this questionnaire enabled us to answer the RQ2.

To prepare the scoring part of the questionnaire, the Framework for Qualitative assessment of Domain-specific Languages (FQAD), introduced in [38] was adopted and customized to the DARC specifications. DSL quality characteristics in this part were scored by the participants in the range of 1-5 on a Likert scale where one means ''Very Bad'' and five means ''Very Good''. When an evaluator thought assessing DARC according to a specific quality characteristic is not applicable and hence preferred not scoring it, he/she simply wrote N/A. In this case, this scoring was omitted while calculating the final average point for this quality characteristic achieved in the whole evaluator group.

The first part of the questionnaire consists of scoring 24 sub-characteristics categorized into 8 different quality characteristics for evaluating the DSL, namely Functional Suitability, Usability, Reliability, Maintainability, Productivity, Extensibility, Compatibility and Expressiveness. Due to the space limitations, descriptions of all these 24 sub-characteristics inside this questionnaire cannot be listed in this paper. However, the whole questionnaire including all these descriptions and the answers received from all evaluators are available in [30].

Figure 11 shows the distribution of the average scores received from the evaluators for each DSL quality characteristic. It is worth indicating that average scores for each sub-characteristic were calculated first and then these average scores were equally weighted to calculate the average score for the quality characteristic covering all these sub-characteristics. For instance, the score for the Usability characteristic is the average of the scores achieved for its sub-characteristics, namely *Comprehensibility*, *Learnability*, *Number of activities for task achievement*, *User perception*, *Operability*, *Attractiveness* and *Compactness*. From these scores, we can see that the evaluators generally found the features of DARC useful in business application development since all of the scores are above 4 over 5 points. The grand average of scores for all responses is 4.48 which can be said quite high especially considering the wide set of the sub-characteristics.

By having one of the highest average scores, the functionality of DARC was confirmed which means the evaluators agreed on DARC's support on expressing the domain-specific concepts and its suitability for the specifications of the business applications as well as the document-oriented design. For instance, the language was found functional in expressing the responsibilities or indicating the authorizations for a business document. Taking into consideration the points given to the quality measures of the Usability characteristics, it seems that the evaluators mostly agreed on the simplicity of the provided notations and the operability of the language construct hence the application documents can be created and put into practice with a minimum effort. Only one evaluator

found the user-friendliness and the comprehensibility of the language as moderate. Reliability of the language was also acknowledged in terms of the model checking and correction features, i.e. the evaluators mostly found that DARC includes correct elements and relations between them for a business model and it prevents the unexpected interactions between its elements during document design. Similarly, DARC got high scores for the maintainability which means the evaluators thought it is easy to add new functionalities or modify the existing ones while the change in a language construct has a minimum effect on the rest.

Productivity by means of shortening the design process and improving the amount of human resource used in programming was generally evaluated as satisfactory. Moreover, DARC was found compatible with various business domains and the document models designed with DARC can be utilized in the existing development processes. Similarly, Expressiveness of the language was also confirmed since the orthogonality of the business concepts defined in DARC was found sufficient by the evaluators as well as the level of abstraction from the underlying implementations and deployment platforms such as Microsoft .Net. Finally, an interesting score was achieved for the Extensibility characteristic. As can be seen from Figure 11, all evaluators gave the highest point 5 to the language's extensibility. This consensus can be originated from the design of DARC document templates since it facilitates adding new responsibilities, collaborations, and authorizations over an extensible document template.

In the second part of the questionnaire, the following questions were asked to the developers to get their feedbacks:

1. Does DARC make software development easier?
2. Do you find DARC suitable/useful for the development of software for business processes?
3. Do you think DARC is strong enough to model overall business structure?
4. Do you think DARC IDE is easy to use?
5. Are there any difficulties you encountered while using DARC? If so, do you have any suggestions to solve it?
6. Please write your suggestions and other comments for improving DARC's features.

All evaluators responded to the first question positively and highlighted the efficiency and speed-up in business application development brought by modeling with DARC. Specifically, many of the evaluators also indicated their reason for finding DARC facilitating the software development as it provides a very standardized way of defining documents and responsibilities as well as creating and managing the authorizations. Finally, one of the experienced evaluators stated that modeling with DARC may help dealing with the complexity of integrity in document concepts inside large-scale business applications.

Regarding the second question, the evaluators generally found DARC feasible especially by means of supporting abstraction from the technical issues of the underlying business process deployment platforms during system development. One evaluator stated that it was too comfortable to concentrate on only the requirements of the business domain and the specifications of the documents without dealing with the implementation details at the first stage. Another evaluator underlined how DARC models are suitable for managing document responsibilities, i.e., responsibilities can be created or modified before/after the creation of the documents and responsibilities can be related with the documents anytime. She also added that triggering the processing of one document from another document is possible over the management of the responsibilities by DARC models. In one of the interesting answers for this question, an evaluator indicated that he believes DARC provides a common design language shared among the big development teams having team members with varying domain knowledge and experience. The only negative answer for this question took the notice of the overengineering that may arise in the small projects modeling with DARC.

Third question aimed at gaining user feedback on whether DARC provides an all-embracing model of enterprise business applications. All evaluators except one confirmed the comprehensiveness brought by the business document definitions with DARC models. Support for different industry domains was acknowledged in most of the answers. However, one evaluator having a negative opinion on this issue indicated that DARC's applicability for different industries and different application scales is up for debate since we need to be sure of the completeness of the requirement analysis of the system prior to the business modeling.

In most of the answers given to the fourth question, it seems the evaluators agreed that DARC IDE provides a simple design where users do not spend much effort, i.e. the IDE facilitates the document design with a handy interface to build the document functionality in addition to the access and modify authorizations. However, one evaluator found the UI design of the IDE not user-friendly, and he stated that coloring and the placement of the components inside the IDE needed further improvements. One response partially confirmed the usability of the IDE since the initialization step for document modeling was found a little bit confusing with this IDE.

For the fifth question, which focuses on the problems faced by the users, the most complained issue was the initial adaptation to the business modeling with DARC. One evaluator indicated that the transition stages from other approaches to the MDE brought by DARC can be painful, while another evaluator found it is quite challenging at the beginning to model the documents instead of programming. However, the same evaluators also agreed that once this adaptation step for modeling is completed, it is more effective to create the applications with DARC.

Finally, the sixth question enabled us to obtain valuable improvement suggestions from the evaluators that can be considered in the future versions of DARC DSL and its IDE. For instance, one evaluator suggested that a mechanism can be built to support adding documents and the modification of document components at runtime. Another suggestion was about improving the modeling interface of the IDE as it

will lead to adding comments and textual explanations to the designed responsibilities. One remarkable suggestion was about supporting the automatic processing of the results of the queries on the responsibilities inside the IDE when all responsibilities of a document are modeled.

VI. THREATS TO THE VALIDITY

There are some threats to the validity of the performed evaluation which can be assessed according to the four main types of the validity threats, namely internal validity, external validity, construct validity, and conclusion validity as described in [39].

Internal threats to the validity of our experiments relate much to the selection and grouping of the software developers participating as the evaluators. We paid attention to conducting this evaluation only with developers who actively implement commercial business applications in the industry. As indicated at the beginning of this section, the developers participating in this study have significant experience on industrial scale business application development and we believe that both their experience and feedback contributed much to the evaluation of the DARC DSL. On the other hand, only a single evaluator group was used instead of two different groups, which could pose a threat to the execution phase. In our previous studies for other application domains (e.g. [32], [33], [34], [35], [40], [41]), we experienced using both single and double evaluator groups. Using a single group may raise the risk that the evaluators take advantage of their prior development experience using DARC DSL while developing the same business application without using DARC (or vice-versa). Using two groups may minimize this risk. However, in case of two groups, the qualitative evaluation based on the user feedback will not be completed in a fruitful way since the groups with or without using DARC will be different. For the questionnaire-based comparison, it is crucial that a single group implements the same software with or without using DARC. There is also the difficulty of creating two homogeneous groups which have almost the same level of domain knowledge, experience and skills. Randomizing the order of the evaluator groups and/or the applied case studies can also be an option (e.g. see [42]) However, we could not follow the same approach here mainly due to the need of completing the implementations of six different document-based applications from scratch instead of just modeling as is the case in [42]. We also had time limitations for the conducted experiments; and finally there would be the unavailability of all evaluators for such an extensive repeating model of evaluation inside the company.

For the threats to the external validity, generalization of the achieved results should be considered. The experimental environment needs to be more realistic. We believe that this threat was mitigated in this study first by selecting professionals who are qualified for the tasks of development and evaluation inside our experiments. Second, the experimental setting exactly represents the industrial practice since the evaluations took place inside a company producing various business processes software which is the domain of the language being evaluated. Rather than being trivial examples, the multi-case studies herein consider actual business application developments leading to the achievement of the commercial products for this company.

Construct validity refers to what extent the operational measures and the cases that are studied really represent what the researchers have in mind. We aimed at evaluating DARC and its IDE taking into consideration its capability on the automatic generation of business applications and adoption of the language and its features by the developers. For this purpose, design and implementation of six different documents and the related responsibility definitions for a large-scale business application were completed inside the experiments where the related tasks were realized with and without using DARC. Moreover, LoC generalization was evaluated according to the all experiment outputs received from the repetition of the same activities by ten different evaluators with varying domain experiences. Feedback from the same developers were obtained via a questionnaire to determine the advantages and disadvantages of using DARC. Hence, the application of this multi-case evaluation methodology enabled us to reach the aim determined at the beginning.

Finally, for the conclusion validity, we need to consider the credibility of the achieved results. Within this context, we believe that selecting multiple case studies with varying complexities on the designed documents and the responsibilities also helped to minimize the risk on the conclusion validity. For example, as the result of each individual business document development with DARC, it was determined that the number and the diversity in type of the required responsibilities encountered in each experiment directly affected the LoC generation performance whereas the amount for the generated document parts were the same independent from the complexity of each case study. Another conclusion threat can be on the number of the evaluators since only ten developers could participate in this evaluation. However, instead of selecting inexperienced users or even students, we specifically paid attention to conduct this evaluation only with developers who have experience in the related field and actively working in the production of various industrial business applications. In addition, the length and the comprehensiveness of multi-case studies also affected the number of volunteers since they were requested to develop a full-fledged business application with six different documents from scratch by applying the whole MDE process and this resulted in more work on the normal workloads of these employees.

VII. CONCLUSION

To facilitate the development of business applications, a DSL called DARC has been introduced in this paper. In MDE with DARC, business documents, specifying the business applications including the descriptions of the responsibilities, authorizations, and collaborations, are used as the first-class entities of system modeling and hence the implementation of

the business applications can be automatically achieved from these document models via model to text transformations built in the DSL.

The comparative evaluation of using DARC DSL during the development of an enterprise field service application inside an international sales, logistics, and service solution provider company showed that all business document descriptions could be fully generated automatically just by modeling with DARC. Moreover, on the average, code for more than 50% of the responsibilities can be automatically created from the corresponding DARC model instances. Finally, according to the users' feedback, the assessment clearly revealed the adoption of DARC features in terms of the DSL quality characteristics, namely functional suitability, usability, reliability, maintainability, productivity, extensibility, compatibility, and expressiveness. Some of the evaluators found it quite challenging at the beginning to adapt modeling instead of programming although they also acknowledged the effectiveness brought into the development once they familiarize themselves with the modeling.

DARC DSL is now being used in Univera during the development of various commercial products. Although the evaluation of using DARC for the development of an enterprise field service application of a household appliances manufacturer is exemplified in this paper, it is worth emphasizing that DARC model and the DSL is used in the design and implementation of various applications for more than 20 companies (Univera's corporate clients) operating in sectors including telecommunication, energy, industrial machine production, hardware services and heating, ventilation, air conditioning, and refrigeration (HVACR). In these applications, business needs covering the management of the workforce, inventory and organization, the collection of client data as well as the procurement of the services and support are met. For instance, automation of service contracts for the life of service contracts and the contract term fulfillment features in a HVACR application are modeled by DARC while the application of managing the shifts/working hours, the skills, the certificates and the trainings of the employees of a manufacturing company is built by the utilization of DARC DSL. Another interesting use of DARC DSL is the development of a business application for the combination of both hardware installation instructions and parts inventories and the automation of warranty follow-ups.

To increase the DARC's current-generation performance for the business documents, which have very complex logic rules for the responsibility definitions, we aim at extending the current DARC metamodel with new entities and relations as the future work. However, our previous experience on the MDE of logic rules in various domains, including business applications (e.g., see [9], [32], [35]) showed that a perfect balance needs to be set up between increasing the ratio of the automatic generation of logic rules and keeping the abstraction of system modeling at the desired level. Therefore, although extending the related metamodels may lead to an increase in generating artifacts, it may also cause both the creation and comprehension of system models to be very complicated. Hence, this issue needs further investigation for DARC models.

ACKNOWLEDGMENT

(Onur Leblebici, Geylani Kardas, and Tugkan Tuglular contributed equally to this work.) The authors would like to thank all staff in Univera Company for their valuable collaboration during the evaluations performed in this study and the directorate of Univera Company for their permission on conducting the evaluation studies.

REFERENCES

[1] N. Kryvinska and A. Poniszewska-Maranda, *Developments in Information & Knowledge Management for Business Applications*. Cham, Switzerland: Springer, 2021.

[2] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, 2nd ed. San Rafael, CA, USA: Morgan & Claypool Publishers, 2017.

[3] T. Kosar, S. Bohra, and M. Mernik, "Domain-specific languages: A systematic mapping study," *Inf. Softw. Technol.*, vol. 71, no. 7, pp. 77–91, Mar. 2016.

[4] A. Bucchiarone, F. Ciccozzi, L. Lambers, A. Pierantonio, M. Tichy, M. Tisi, A. Wortmann, and V. Zaytsev, "What is the future of modeling?" *IEEE Softw.*, vol. 38, no. 2, pp. 119–127, Mar./Apr. 2021.

[5] D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, and M. Wimmer, "Low-code development and model-driven engineering: Two sides of the same coin?" *Softw. Syst. Model.*, vol. 21, no. 2, pp. 437–446, Apr. 2022.

[6] A. Popovic, I. Lukovic, V. Dimitrieski, and V. Djukic, "A DSL for modeling application-specific functionalities of business applications," *Comput. Lang., Syst. Struct.*, vol. 43, pp. 69–95, Oct. 2015.

[7] A. Oliveira, V. Bischoff, L. J. Gonçales, K. Farias, and M. Segalotto, "BRCode: An interpretive model-driven engineering approach for enterprise applications," *Comput. Ind.*, vol. 96, pp. 86–97, Apr. 2018.

[8] C. Rieger and H. Kuchen, "A process-oriented modeling approach for graphical development of mobile business apps," *Comput. Lang., Syst. Struct.*, vol. 53, pp. 43–58, Sep. 2018.

[9] K. Soleymanzadeh, Y. Bul, S. Bagci, and G. Kardas, "A tool for modeling JsonLogic based business process rules," in *Proc. 1st Int. Informat. Softw. Eng. Conf. (UBMYK)*, Ankara, Turkey, Nov. 2019, pp. 263–267.

[10] Q. Lu, A. B. Tran, I. Weber, H. O'Connor, P. Rimba, X. Xu, M. Staples, L. Zhu, and R. Jeffery, "Integrated model-driven engineering of blockchain applications for business processes and asset management," *Softw., Pract. Exper.*, vol. 51, no. 5, pp. 1059–1079, May 2021.

[11] M. Vještica, V. Dimitrieski, M. Pisarić, S. Kordić, S. Ristić, and I. Luković, "Multi-level production process modeling language," *J. Comput. Lang.*, vol. 66, Oct. 2021, Art. no. 101053.

[12] V. Kulkarni, "Model driven development of business applications: A practitioner's perspective," in *Proc. 38th Int. Conf. Softw. Eng. Companion*, Austin, TX, USA, May 2016, pp. 260–269.

[13] N. Yousaf, F. Azam, W. H. Butt, M. W. Anwar, and M. Rashid, "Automated model-based test case generation for web user interfaces (WUI) from interaction flow modeling language (IFML) models," *IEEE Access*, vol. 7, pp. 67331–67354, 2019.

[14] Object Management Group (OMG). (2010). *Business Process Model and Notation*. Accessed: Sep. 25, 2022. [Online]. Available: https://www.omg.org/spec/BPMN/2.0/About-BPMN/

[15] M. Brambilla and P. Fraternali, *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps With IFML*. Burlington, MA, USA: Morgan Kaufmann, 2014.

[16] JsonLogic. (2017). *JsonLogic: Build Complex Rules, Serialize Them as JSON, Share Them Between Front-End and Back-End*. Accessed: Sep. 25, 2022. [Online]. Available: https://jsonlogic.com/

[17] T. A. Majchrzak and J. Ernsting, "Achieving business practicability of model-driven cross-platform apps," *Open J. Inf. Syst.*, vol. 2, no. 2, pp. 3–14, 2015.

[18] M. Challenger, F. Erata, M. Onat, H. Gezgen, and G. Kardas, "A model-driven engineering technique for developing composite content applications," in *Proc. 5th Symp. Lang., Appl. Technol.*, Maribor, Slovenia, 2016, pp. 11:1–11:10.
[19] N. Ferry, M. Almeida, and A. Solberg, "The MODAClouds model-driven development," in *Model-Driven Development and Operation of Multi-Cloud Applications* (SpringerBriefs in Applied Sciences and Technology), E. Di Nitto, P. Matthews, D. Petcu, and A. Solberg, Eds. Cham, Switzerland: Springer, 2017, pp. 23–33.
[20] A. Colantoni, L. Berardinelli, and M. Wimmer, "DevOpsML: Towards modeling DevOps processes and platforms," in *Proc. 23rd ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst., Companion*, Oct. 2020, pp. 1–10.
[21] Z. Bicevska, J. Bicevskis, and G. Karnitis, "Models of event driven systems," *Commun. Comput. Inf. Sci.*, vol. 615, pp. 83–98, Jan. 2016.
[22] H. Henriques, H. Lourenço, V. Amaral, and M. Goulão, "Improving the developer experience with a low-code process modelling language," in *Proc. 21st ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst.*, Copenhagen, Denmark, Oct. 2018, pp. 200–210.
[23] F. Benaben, S. Truptil, W. Mu, H. Pingaud, J. Touzi, V. Rajsiri, and J. P. Lorre, "Model-driven engineering of mediation information system for enterprise interoperability," *Int. J. Comput. Integr. Manuf.*, vol. 31, no. 1, pp. 27–48, 2018.
[24] Y. Cao, Y. Liu, H. Wang, J. Zhao, and X. Ye, "Ontology-based model-driven design of distributed control applications in manufacturing systems," *J. Eng. Des.*, vol. 30, nos. 10–12, pp. 523–562, Dec. 2019.
[25] C. Campos and R. Grangel, "A domain-specific modelling language for corporate social responsibility (CSR)," *Comput. Ind.*, vol. 97, pp. 97–110, May 2018.
[26] Eclipse Foundation. (2022). *Eclipse Modeling Framework*. Accessed: Sep. 25, 2022. [Online]. Available: https://www.eclipse.org/modeling/emf/
[27] Eclipse Foundation. (2022). *Eclipse Sirius Domain-Specific Modeling Tool*. Accessed: Sep. 25, 2022. [Online]. Available: https://www.eclipse.org/sirius/
[28] Eclipse Papyrus. (2022). *Eclipse Papyrus Modeling Environment*. Accessed: Sep. 25, 2022. [Online]. Available: https://www.eclipse.org/papyrus/
[29] U. Zdun and M. Strembeck, "Reusable architectural decisions for DSL design: Foundational decisions in DSL development," in *Proc. 14th Annu. Eur. Conf. Pattern Lang. Program.*, Irsee, Germany, 2009, pp. 1–37.
[30] (2022). *DARC DSL Repository*. Accessed: Sep. 25, 2022. [Online]. Available: https://data.mendeley.com/datasets/6d9nv4gk24/1
[31] M. Challenger, G. Kardas, and B. Tekinerdogan, "A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems," *Softw. Quality J.*, vol. 24, no. 3, pp. 755–795, 2016.
[32] G. Kardas, B. T. Tezel, and M. Challenger, "Domain-specific modelling language for belief–desire–intention software agents," *IET Softw.*, vol. 12, no. 4, pp. 356–364, Aug. 2018.
[33] S. Arslan and G. Kardas, "DSML4DT: A domain-specific modeling language for device tree software," *Comput. Ind.*, vol. 115, Feb. 2020, Art. no. 103179.
[34] O. F. Alaca, B. T. Tezel, M. Challenger, M. Goulão, V. Amaral, and G. Kardas, "AgentDSM-Eval: A framework for the evaluation of domain-specific modeling languages for multi-agent systems," *Comput. Standards Interface*, vol. 76, Jun. 2021, Art. no. 103513.
[35] H. Marah, G. Kardas, and M. Challenger, "Model-driven round-trip engineering for TinyOS-based WSN applications," *J. Comput. Lang.*, vol. 65, Aug. 2021, Art. no. 101051.
[36] Univera Co. (2022). *Univera Bilgisayar Sistemleri Sanayi ve Tic A.Ş.* Accessed: Sep. 25, 2022. [Online]. Available: https://www.univera.com.tr/
[37] Promotion Optimization Institute (POI). (2021). *POI 2021 Retail Sales Execution Report*. Accessed: Sep. 25, 2022. [Online]. Available: https://poinstitute.com/
[38] G. Kahraman and S. Bilgen, "A framework for qualitative assessment of domain-specific languages," *Softw. Syst. Model.*, vol. 14, no. 4, pp. 1505–1526, 2015.
[39] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering*. Berlin, Germany: Springer, 2012.
[40] O. Yildirim and G. Kardas, "A multi-agent system for minimizing energy costs in cement production," *Comput. Ind.*, vol. 65, no. 7, pp. 1076–1084, Sep. 2014.
[41] H. B. Saritas and G. Kardas, "A model driven architecture for the development of smart card software," *Comput. Lang., Syst. Struct.*, vol. 40, no. 2, pp. 53–72, Jul. 2014.
[42] T. Miranda, M. Challenger, B. T. Tezel, O. F. Alaca, A. Barisic, V. Amaral, M. Goulao, and G. Kardas, "Improving the usability of a MAS DSML," in *Proc. 6th Int. Workshop Eng. Multi-Agent Syst. (EMAS), 17th Int. Conf. Auton. Agents Multiagent Syst. (AAMAS)*, in Lecture Notes in Artificial Intelligence, vol. 11375. Stockholm, Sweden, 2019, pp. 55–75.

ONUR LEBLEBICI received the B.Sc. degree in computer engineering from Hacettepe University, Turkey, in 2006, and the M.Sc. degree in computer engineering from the Izmir Institute of Technology, Turkey, in 2020. Since 2006, he has been worked as a software engineer, a research and development manager, a software development manager, and a software development consultant at various national and international companies. He is currently the Chief Software Architect at Univera Company. He is responsible for managing research and development projects, designing and building brand new software development platforms, and determining architectural compliance within the context of the system architecture and best practices. His research interests include enterprise business applications, software architectures, and model-driven engineering.

GEYLANI KARDAS received the B.Sc. degree in computer engineering and the M.Sc. and Ph.D. degrees in information technologies from Ege University, Turkey, in 2001, 2003, and 2008, respectively. He is currently an Associate Professor with the International Computer Institute (ICI), Ege University, and the Head of the Software Engineering Research Laboratory (Ege-SERLab), ICI. He works as the principle investigator, a researcher or a consultant in various research and development projects funded by governments, agencies, and private corporations. His research interests include agent-oriented software engineering, model-driven engineering, domain-specific (modeling) languages, and low-code software development. He has authored or coauthored over 100 peer-reviewed papers in these research areas. He is an Associate Editor of the *Journal of Computer Languages* (Elsevier).

TUGKAN TUGLULAR (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer engineering from Ege University, Turkey, in 1993, 1995, and 1999, respectively. He worked as a Research Associate at Purdue University, from 1996 to 1998. He has been with the Izmir Institute of Technology, since 2000. He is currently working as an Associate Professor at the Department of Computer Engineering. His research interests include software testing, model-based software development, and machine learning augmented software engineering.