Zooming in for Clarity: Towards Low-Code Modeling for Activity Data Flow

Ali Nour Eldin[1,2(✉)], Jonathan Baudot[2], and Walid Gaaloul[1]

[1] Telecom SudParis, Institut Polytechnique de Paris, Palaiseau, France
{ali.nour_eldin,walid.gaaloul}@telecom-sudparis.eu
[2] Bonitasoft, Grenoble, France
{ali.nour-eldin,jonathan.baudot}@bonitasoft.com

Abstract. Business Process Modeling and Notation (BPMN) is a widely used standard workflow language for modeling business processes. However, there is a growing need to integrate data and process models to enable a more holistic view of business processes to reducing implementation time through a clear understanding of the mode, and BPMN has limitations in representing data-related concepts. To address this, we propose an extension to BPMN called DataFlow BPMN (DF-BPMN), which is a low-coding visual solution, for modeling and analyzing the relationship between process and data. Low-code is a growing development approach supported by many platforms. It fills the gap between business and developer. Indeed, it enables quick generation and delivery of business applications with minimum effort. We introduce the Activity DataFlow, an extension of activity that allows zooming into the data manipulated within it, which enable different levels of granularity: control-flow perspective and data perspective. Additionally, we developed a tool for creating a model with the DF-BPMN. Our approach has been evaluated quantitatively and qualitatively, and the results demonstrate that DF-BPMN offers significant advantages over BPMN.

Keywords: BPMN · Low-Code · Process Modeling · data perspective · data models

1 Introduction

A business process is a set of activities within an enterprise that follows a defined logical order and dependency, with the objective of producing a desired result. Process models provide a comprehensive understanding of a process [3], and can be understood from different perspectives [2]. The *control-flow perspective* describes tasks and their execution order through different constructors, which can be modeled using BPMN [8]. The *data perspective* deals with business and processing data, similar to UML [9]. Integrating these perspectives within the

C. Di Francescomarino et al. (Eds.): BPM 2023 Forum, LNBIP 490, pp. 267–282, 2023.
https://doi.org/10.1007/978-3-031-41623-1_16

same workflow modeling language is essential for success, as it enhances information representation, reduces implementation time, and enables effective decision-making and resource allocation.

Low-Code Development Platforms (LCDPs) provide development environments for creating software applications using visual interfaces, rather than traditional manual coding methods. These platforms use process modeling languages to define business processes and automate related tasks, such as data modification, scheduling, or linking data flows to external services. Process executions help bridge the gap between business and developer [10]. Figure 1 illustrates the steps involved in LCDP development compared to the Business Process Lifecycle: (i) language specification, (ii) language compilation, and (iii) execution and monitoring. Furthermore, BPMN elements are commonly used as the language specification in LCDP-based business processes [10].

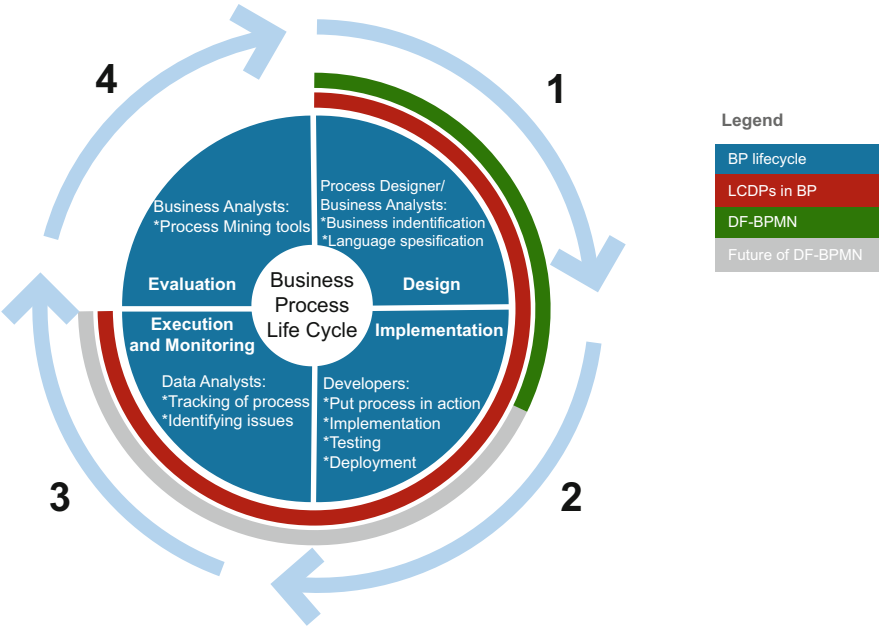

Fig. 1. Comparison of the Business Process Lifecycle vs. Low-Code Process Development (LCPDs) in Business Process (BP) vs. the DF-BPMN approach

Business Process Modeling and Notation (BPMN) is a widely-used standard for modeling business processes, designed to be comprehensible by various stakeholders such as analysts, developers, and business people [8]. However, BPMN process models have limited detail on persistent data structures and struggle to represent interactions between data objects and data stores. As a result, the data flow perspective has been neglected in BPMN automation, in comparison to the

control flow. This can lead to misunderstandings and errors during implementation by technical developers. For example, in the BPMN model shown in Fig. 2, the activity "Complete quotation" has an input and output "DB" data store representing a database, without detailing the object name and their attributes. Therefore, the representation of "DB" can be ambiguous and prone to mistakes during process implementation.

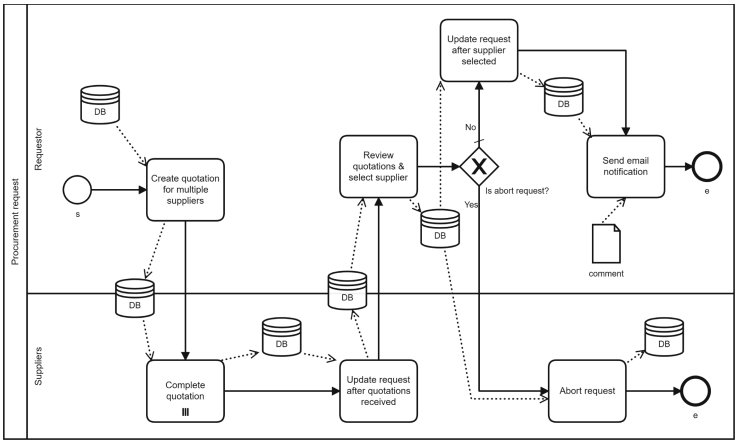

Fig. 2. A procurement request using BPMN 2.0.

This paper focuses on the first step of low-coding, which is language specification. We propose an extension to BPMN called DataFlow BPMN (DF-BPMN) that integrates a data perspective with the control flow perspective. DF-BPMN bridges the gap between process and data diagrams, enabling a clear understanding of the data involved in business processes (see Fig. 1). With DF-BPMN, developers can represent data in a graphical format within BPMN, zooming into the activity for modeling manipulated data and offering different levels of granularity for a better understanding of the interactions between control flow and data perspectives. To evaluate our language, we conducted a quantitative analysis based on real-life business process models and analyzed the results statistically using the Paired t-test [5].

The remainder of this paper is organized as follows. In Sect. 2, we discuss the limitations of BPMN in terms of the data-flow perspective. In Sect. 3, we review existing works related to our contribution. In Sect. 4, we present the zoom-in approach of DF-BPMN. In Sect. 5, we discuss the novel insights of merging data and processes. Then, in Sect. 6, we present the evaluations of DF-BPMN before concluding in Sect. 7.

2 Motivating Example and Limitations of BPMN as a Data-Flow Language

Figure 2 represents a business process model modeled in BPMN 2.0. A user fills a procurement request and identify potential suppliers. This request is sent to those suppliers for quotation. After completed, the quotations are sent back to the requestor for review and selection. To effectively implement the process in Fig. 2, developer must understand how the information is conceptually structured and arranged within each class (suppliers, quotation and request) and how the process interacts with it. *Request* has connected to *Quotation* and *Supplier*. Each *Request* has one or more *Quotation*. Each *Quotation* has one *Supplier* and also the *Supplier* is selected for a *Request*. All these necessary information are concealed in a simple *data store* "DB" in Fig. 2.

We identified conceptual limitations in BPMN **(L1-L3)** that hinder model understandability and quality, requiring a well-structured approach to address these issues and provide clearer guidance for developers.

L1. BPMN Data Stores are Underspecified. Data stores in BPMN process models represent persistent data sources [8], but they lack details on the conceptual structure of a database. For example, in the activity "Create quotation for multiple suppliers" it is unclear whether the input classes are *Supplier*, *Request*, or both, and whether the output class is *Quotation*. This ambiguity complicates the use of BPMN for process models connected to database systems, hinders tracking instance manipulation, and negatively impacts understandability between different developers. For instance, some developers can understand from the data store connection in "Create quotation for multiple suppliers" activity, that the request is already created and they need to select it from the database, while others may understand that they need to create a new instance object of *Request*, which can lead to time loss during implementation.

L2. Interaction Between Data Instances is Not Clear. Data objects represent volatile process data and are connected to activities through associations [8]. For example, in the "create quotation for multiple suppliers" example, the interaction between data instances, such as the "DB" data store used as input and output, is unclear. This ambiguity, coupled with the lack of explicit representation of data cardinality [2], like multi-instance data, complicates the model's interpretation. These limitations can result in development inefficiencies, as developers struggle to determine involved data objects, required information, and data relationships. Understandability of the BPMN model can also vary among developers based on their experiences and perspectives, highlighting the need for a single visual language that conveys all necessary information, including data cardinality.

L3. Data From/To External Environment are Not Supported. Information systems are often connected to external environments, such as user interfaces and services, making it crucial to represent the interaction between process activities and external resources graphically for effective process modeling and

development. However, BPMN models lack support for this representation, making it challenging to determine which activities are connected to external data or services. For instance, the activity "Complete quotation" requires the user (a supplier) to provide necessary quotation information through an external user interface, which is not supported in BPMN. Therefore, while implementation, the developers required textual infractions to understand the process model. The graphical representation of external resources is important for effective process implementation, as it helps prevent misunderstandings and errors during implementation.

Since process models and data schema are conceptualized independently, it is necessary to support designers in understanding and capturing the relationship between processes and different data types in order to develop data aware process modeling.

3 Related Work

Low-Code Development Platforms (LCDPs) currently focus on the execution of the process model, with little attention given to extending process modeling languages to make them easier for developers to use [10]. In this section, we not only present LCDPs, but also illustrate some approaches that extend BPMN to link data and processes [4,6,7].

LCDPs are defined as "platforms that enable rapid delivery of business applications with a minimum of hand-coding." Most of these platforms use similar concepts in graphical user interfaces, allowing developers to define and manipulate data specified through tables, forms, reports, and other types of representation [10]. As the author said, "we are comparing BPMN modeling elements to build the case study application with its equivalent modeling constructs using different LCDPs associated with the data handling mechanism and their implementation." which mean that most of platforms used BPMN elements. For example, Bonita[1] is an open-source and extensible platform for business process automation and optimization. They use BPMN diagrams to describe the business processes. The structure of the business data is presented by Business Data Model. Also, they allow developers to define forms, reports, and other types of representation. Another tool called Mendix[2] allows the user to build processes using the available Microflow modeling language, which is based on the BPMN standard and helps to extend or change the default behavior of the developed application. This language defines the same elements as BPMN but with different symbols. For example, an event in Mendix is equivalent to an event in BPMN (details in [10]). However, these representations cannot help developers in their implementation by low-coding. LCDPs in business processes focus on the execution of the process and facilitate the execution of the process to the organization. But developers still spend time on the implementation, and they cannot track their business data using the existing workflow languages.

[1] https://bonitasoft.com.
[2] https://www.mendix.com.

Various approaches have been developed to extend BPMN and add support for data modeling. These approaches introduce new concepts, elements, and mechanisms for modeling data entities, relationships, and constraints, thus enabling a more comprehensive and integrated representation of both process and data aspects. In [7], the information model of a process is connected to its BPMN process diagram through OCL expressions. The information model is represented by a class diagram that includes a "Artifact" class containing process variables. The process diagram is formalized as a Petri net, and BPMN activities are defined using OCL operation contracts. These contracts are converted into logic derivation rules that can be easily translated into SQL queries. Another framework defines new variables, pre-conditions, and effects on activities by adding new properties and accessing data objects and data stores to modify them [6]. This framework uses a verification model to parametrically verify data-aware processes with respect to read-only relations. However, these approaches neglect the important aspect of process visualization, which is necessary for developers to easily understand and use the model. In [4], "Activity Views" proposes a new extension to bridge the gap between process and data modeling, with a focus on databases. Moreover, [4] only considers the databases aspect of the model. However, The data in Business process does not restricted to database, and there are different data types used in BP, like data object in BPMN.

However, our approach introduces a new visual concept that allows developers to easily model data in BPMN. Furthermore, it enables developers to zoom in on activities to manipulate and interpret data. Additionally, the ability to zoom in and out of activities allows for exploration of different levels of granularity, a feature that is not supported in existing works. Indeed, our approach takes into account most of the data required in business processes.

4 DF-BPMN: DataFlow in Business Process Modeling and Notation

In this section, we will present the graphical elements of our language, followed by an example that addresses all the limitations of BPMN using DF-BPMN, and then proceed with the formal definition of DF-BPMN.

4.1 Graphical Process Modeling Using DF-BPMN

In this section, we present a first step to low-coding approach that aims to establish a conceptual link between BPMN process models and data models, in order to capture the connection between them more effectively, which helps developers better understand the model. Our proposed concept, ***Activity DataFlow***, allows for a more detailed representation of how process activities interact with and manipulate different types of data, such as database, process variable, and external environment data. By specifying the inputs, outputs, and data operations performed by each activity, the method provides a comprehensive and

detailed representation of the process and its relationship with data. This allows zooming into the activity for early identification of potential data-related issues and better optimization of the data flow within the process. This helps reduce implementation time and minimize mistakes during execution. Additionally, zooming out of the activity presents the control flow perspective of the process, while developers can zoom in on activities as needed.

Our approach to modeling data in BPMN starts with process activities, which are a common starting point for data modeling. The main objective of the *Activity DataFlow* extension is to visualize how data flows into a specific activity based on various sources, and to capture important details about the data operations that are performed. This allows us to better understand the flow of data throughout an activity.

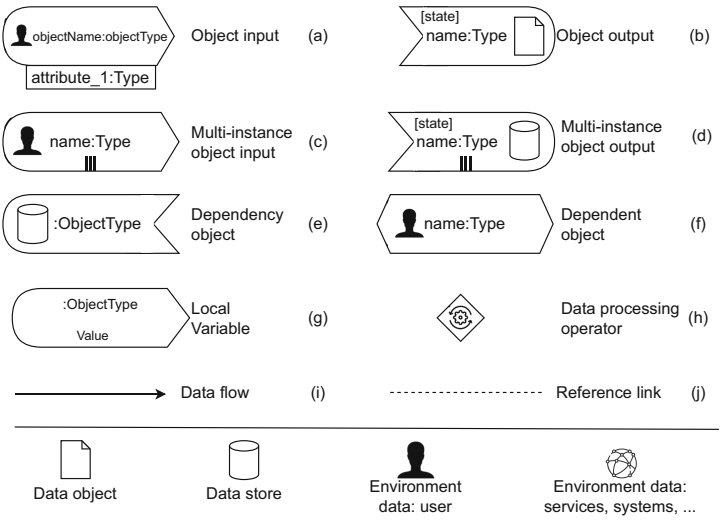

Fig. 3. Symbols used in DF-BPMN.

Figure 3 represents the symbols proposed to allow to build your BPMN models injected by data[3]. The symbols include inputs and outputs that represent the data objects manipulated within the activity. Each object can have a type, and there are various types of objects such as process variables (also known as data objects in BPMN), databases (also known as data stores in BPMN), two types of environment data: (1) related to user operations, such as user forms and websites; and (2) external resources, such as services and systems; as well as local data, which is a static value used within the activity. The icons in the second part of Fig. 3 represent several types of data objects (input or output). Except for local data, which was displayed without an icon.

[3] You can find the detailed documentation here.

The input shape in this language is a semi-circle with an arrow (see Fig. 3 (a)). Each input can have one or more attributes, which are presented as rectangles. The input shape can represent one of several types, as indicated by corresponding icons. Each input also includes the name and type of the object it represents. Similarly, each attribute has a name and type associated with it. The output shape is the complement of the input shape, starting with a left arrow and semi-round shape. It has several types, like the input, and includes a name and type, as well as the option to attach attributes. Additionally, the state of the output object is indicated by "[]" at the top of its name, indicating any operation performed on it during the process (see Fig. 3 (b)).

To facilitate interaction between the input and the output, a dataflow sequence is introduced to represent the transfer of input from the source to the destination, it's a sequence flow presented by an arrow. The dataflow sequence visually connects the input and output, indicating the direction of data transfer. In this way, the flow of information between the input and the output can be easily understood by the process designer (see Fig. 3 (i)). We also use a reference link (as shown in Fig. 3 (j)) to represent equivalent data objects. The reference link is displayed using a dotted line. In addition, the representation of data also needs to be considered. Sometimes, the input may read multiple instances of the same data object, while the output may write or update multiple objects. To address this, the multiple-instance object is represented by adding three bars "|||" to the input or output shape (as shown in Fig. 3 (c-d)).

Moreover, our DF-BPMN language includes three other types of input: dependency, dependent, and local data objects. The dependent and dependency objects are always interdependent, representing situations where the user needs to select certain data from the input in order to modify the output. The dependency object is always represented as a user (f) who needs to read and select data from the dependency object (data object/store) (e), therefore, there are dependent together. The local data object is a variable that can be used as a static value within the activity (g). These different types of inputs expand the range of use cases for our language, providing more flexibility and precision in modeling complex processes.

Finally, in the context of our DF-BPMN language, complex operations such as arithmetic operations, logical operations, and conditions are represented using data processing operators. These operators are denoted by the symbol (h) and can be used in combination with input and output objects to create complex processing logic. For example, if an activity requires the addition of two input objects, it can be represented using a data processing operator denoting addition, with the two input objects as inputs and the output object as the result. These data processing operators add another layer of flexibility to the DF-BPMN language, allowing for the representation of more complex business processes.

4.2 DF-BPMN in Action

Figure 4 depicts several instances of Activities DataFlow for the example shown in Fig. 2. These visualizations provide a way to zoom in on specific activities

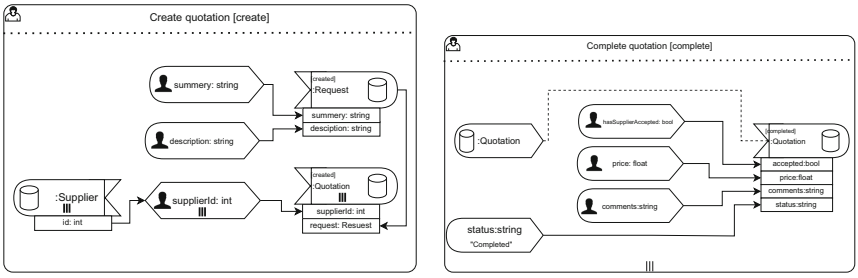

Fig. 4. Activities Modeled with DF-BPMN Language: (a) Create Quotation, (b) Complete Quotation.

and understand how data is manipulated within them without ambiguity[4]. For example, the activity "create quotation" in Fig. 2 has one input data store "DB" and one output data store "DB", making it impossible to determine which data object is in the input and which are in the output. However, the Activity DataFlow "create quotation" (shown in Fig. 4(a)) extends the "create quotation" activity to include all data related to it. It has four inputs: one input from the data store called "supplier" object, which has one attribute (i.e., "id"), and three inputs from a user (environment data) (**L3**). It also has two output objects, one of which is a multiple-instance object (cf. **L1**).

Furthermore, in the same activity, the representation of inputs and outputs addresses **L2** from Sect. 2. The *supplierId* object is a list of integers selected by the user based on the list of objects *supplier* already stored in the data store. Thus, we have two input data objects that are dependent on each other, called the dependency and dependent objects, respectively: *supplier* from the database and *supplierId* list selected by the user. Then, the *summary* and *description* are used to create a *Request* object, which currently has a *[created]* state. Moreover, this created object (*Request*) is used as input for the *Quotation* objects. The *Quotation* is a multiple-instance object based on the list of *supplier* selected by the user, creating a *quotation* object for each *supplierId*. This situation provides a solution for **L2**. This indicates that the goal of the "create quotation" activity is to create two objects with states *[created]*, with respect to the milestone of the activity, which is *[create]*, visualized around the name of the activity. This case resolves the ambiguity of the activity milestone presented in **L4** in Sect. 2.

In addition, the attributes of each object refer to the conceptual model of UML. This helps to resolve **L1** by providing information on the required attributes for each UML class that is being processed in the activity. The association between inputs and outputs represents the interaction between data within the activity (**L2**). The Activity DataFlow "completed quotation" (in Fig. 4(b)) has different input types. It includes a local data input *status*, which takes a value of "completed" to modify the value in the *Quotation* object.

[4] You can find all the process models here.

4.3 The Missing Link: Uniting Process and Data for Clarity

Starting from the BPMN definition in [8], we extend an activity to include a data flow into it. Therefore, Definition 3 indicates the activity's extension into *Activity DataFlow*. Each activity that has data can be enhanced to a *Activity DataFlow*. The latter consists of: (i) a set of inputs defined in Definition 1, which are data instances from various resources, such as a local data, a data object (process variable), a data store (database), or a data environment (other resources: user or other); (ii) a set of outputs defined in Definition 2, which are data instances are written/modified by the activity; (iii) a set of operators, which are the data processing operation from one or multiple inputs to one or multiple outputs, to represent that are operations do in this activity, e.g., condition, arithmetic's operations,... ;(iv) and (v) a collection of dataflow and a set of references that reflect the interaction of data instances into the activity; a dataflow sequence represents the correlation of a data instance; and a reference represents the equivalent of data instances[5]. For example, "Create quotation" of Fig. 2 is connected to data, therefore it can be extended to *Activity DataFlow*. In which, this Activity DataFlow has: (i) Four inputs: one database input (supplier), and three user inputs; (ii) Two databases outputs (request,quotation). The interactions between these data instances are represented through a collection of dataflow and a set of references in the Activity DataFlow.

Definition 1 (Activity Input). *Let* $Input = GI \cup LI$ *as an input of an activity, where:*

- $GI = (objectName, objectType, type, Att_{set}, isMuliple)$ *as a Global Input where:*
  - *objectName is a name to describe the input objectType can be any object type you need (e.g. integer, string or complexType)*
  - $type \in \{data\_object, data\_store, user, systems\}$
  - $Att_{set} = \{att_1, ..., att_k\}$ *where* $att_k = (attributeName, attributeType)$ *such as attributeName is the description of attribute attributeType is any type (e.g. integer, string, complexType, ...)*
  - *isMuliple is a boolean variable to describe if the object is represent multiple-instance object*
- $LI = (objectName, objectType, objectValue, isMuliple)$ *as Local Input such as* $objectValue$ *represents the static value of this input.*

Definition 2 (Activity Output). *Let Output = (objectName, objectType, type, state,* Att_{set}*, isMuliple) as an output of an activity, where objectName, objectType, type,* Att_{set}*, isMuliple are represented in the Definition 1, and state define the state of each object data during the execution time (like created, updated, deleted, ...)*

Definition 3 (Activity DataFlow). *Given an activity ac in a process model. Let Activity DataFlow* $ad_{ac} = (state, I_{set}, O_{set}, Operator_{set}, DF_{set}, R_{set})$ *is a tuple to represents the data flow into the activity, where:*

[5] You can find the implementation details here.

- $I_{set} = \{i_1, \ldots, i_j \mid i_j \in Input\}$, *is a set of inputs accessed by process activity ac.*
- $O_{set} = \{o_1, \ldots, o_l \mid o_l \in Output\}$, *is a set of output produced by process activity ac.*
- $Operator_{set} = \{operator_1, \ldots, operator_m\}$ *is a set of operators in which represent the existing of data processing operations where* $operator_m$ = *operaterName and operaterName is a name to describe the operator.*
- $DF_{set} = \{df_1, \ldots, df_l\}$, *is a set of dataflow sequence to connect different data objects, where* df_l = *(sourceObject, targetObject) such as* $\{sourceObject, targetObject\} \subset \{I_{set} \cup O_{set} \cup Att_{set} \cup Operator_{set}\}$ *where* $Att_{set} \subset \{i_j.Att_{set} \cup o_l.Att_{set} \mid i_j \in I_{set}$ *and* $o_l \in O_{set}\}$ *and* $sourceObject \neq targetObject$ *and vise versa.*
- $R_{set} = \{r_1, \ldots, r_n\}$, *is a set of references to represents the equivalent between objects, where* $r_n = (o_1, o_2)$ *such as* $\{o_1, o_2\} \subset \{I_{set} \cup O_{set} \cup Att_{set}\}$ *where* $Att_{set} \subset \{i_j.Att_{set} \mid i_j \in I_{set}\}$ *and* $o_1 \neq o_2$ *and vise versa.*

Activity DataFlow represents an integration of two established standards, namely BPMN and UML. It combines the concept of activity from BPMN with the class, attribute, and relationship concepts from UML. This feature of the Activity DataFlow method addresses several outstanding limitations (L1-L4)(Sect. 2) related to the representation of data in BPMN processes. Moreover, the definition of inputs(cf. Definition 1) and outputs(cf. Definition 2) resolve the underspecification of data store. Indeed, there are different types of it, included the external environment, which means, the representation of data from/to external environment are supported (cf. L3). Additionally, in Def 3, the dataflow sequence, the reference, and the data processing operator resolve the ambiguity of the interaction between different data instances type (cf. L2). Finally, the state of the activity and the state of each data output instance represent the milestone for each one (cf. L4, resolved).

5 When Processes and Data Meet: Integrating Analysis and Deployment

This section discusses some features of the research design area that resulted DF-BPMN and demonstrates the novel perspectives that may be discovered by using the DF-BPMN during process design, analysis, and deployment.

Understanding the Process Model Using Different Granularity. The concept of granularity is essential in human cognition, as it relates to the production, interpretation, and representation of granules [11]. A granule is a group of objects or points that are connected by either their familiarity, proximity, or utility. This process of granulation results in the formation of granules. When it comes to process modeling, granularity refers to the level of detail at which a process is represented. In the case of DF-BPMN, there are two granularities available: (i) The first granularity represents the entire control-flow of a model

using BPMN2.0. For example, a standard BPMN diagram can be used to represent a quotation request. (ii) The second granularity is facilitated by the DF-BPMN model, where each activity can be either expanded or collapsed, allowing the reader to focus on the specific data being manipulated in those activities. Figure 5 is an example of a DF-BPMN that expands one activity and collapses all others in the model to represent the same quotation request example. These different granularities and representations provide valuable insight into the process, making it easier to analyze and communicate with stakeholders.

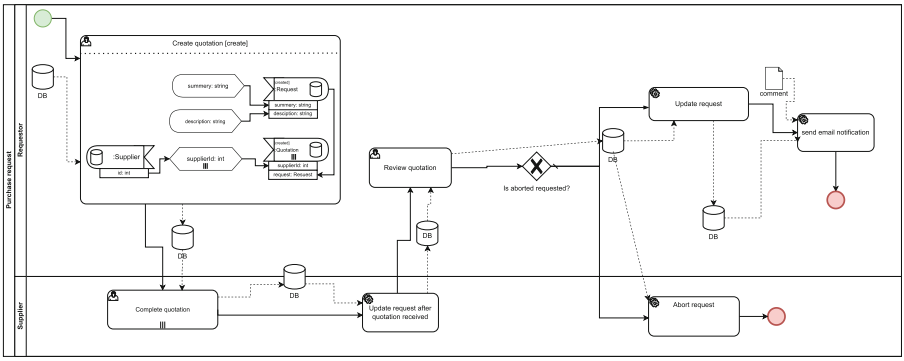

Fig. 5. DF-BPMN example which expand "create quotation" activity and collapse the others activities.

Tracking Data Objects in Business Process. Our extension increases the expressiveness of a BPMN process model with information about process-data-correlation on instance level. As such, it does not interfere with standard BPMN semantics. We defined a state for each data object, thus allow the developer to easily track the operation of each object by extracting its lifecycle based on the state changes. For instance, Fig. 6 represents the lifecycle of the object "Request", including the activity in which it was started and finished. In addition, the object-centric [1] has emerged recently to represent each object instance. This lifecycle can helps in the extraction of Object-Centric Event Data. OCED captures events and activities on specific objects in a system and provides a more detailed view of how objects are processed, making it useful for analyzing complex systems and processes and improving system performance using process mining tools.

6 Evaluation

The aim of this section is to evaluate the understandability of DF-BPMN by process designers and developers, comparing it to BPMN and Activity View in terms of presenting different data types. We will first outline the evaluation steps, and then present the results obtained from the study.

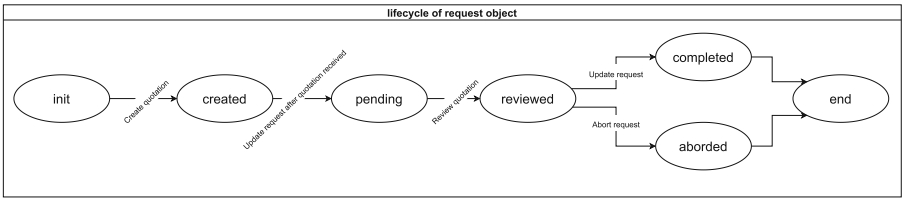

Fig. 6. Request object lifecycle.

6.1 Experiment Description

We conducted an evaluation of our DF-BPMN language in two phases. In Phase 1, we provided a tutorial to introduce DF-BPMN and its usage, with no prerequisite knowledge of UML. Additionally, we presented another work [4] in the tutorial for experimentation and comparison purposes. To ensure comprehension of DF-BPMN and the other work, we conducted a brief quiz. In Phase 2, we aimed to evaluate the understandability of DF-BPMN by presenting three exercises from different domains, with each exercise having different models (BPMN, DF-BPMN, and Activity View) for evaluation. We developed a web application tool[6] available on GitHub at https://github.com/NourEldin-Ali/open-bpmn, using the open-source project Eclipse GLSP[7] to use our language.

In order to evaluate the understandability of DF-BPMN, a human-oriented experiment was conducted, similar to a previous study [4], with a single controlled variable. The goal was to measure the understandability of DF-BPMN by process designers and developers, comparing it with BPMN and other works, particularly with respect to the relationship between a process and data. Two hypotheses were formulated to analyze these improvements quantitatively and qualitatively. The first hypothesis (H1) tested the perceived ease of understanding, suggesting that DF-BPMN improves the visualization of data objects, leading to a better understanding of the data required for activities and how it is utilized in a process, without any textual information or UML. The second hypothesis (H2) tested the perceived ease of understanding, suggesting that DF-BPMN is a better solution for modeling data in BPMN.

The evaluation was conducted by five PhD students who had basic knowledge of BPMN and three professional developers working on a Business Process Management System editor. In Phase 1, all the participants attended a 30-minute tutorial, consisting of a video and a short quiz, on how to use DF-BPMN and Activity View.

In Phase 2 of the evaluation, participants were divided into three groups to validate Hypotheses (H1) and (H2) through three exercises, each featuring a questionnaire with 12 questions. The exercises provided participants with a textual process description, data operations, BPMN, and UML diagrams. Each

[6] Online Demo: https://github.com/NourEldin-Ali/open-bpmn#start-the-online-demo,.

[7] https://www.eclipse.org/glsp/.

group used a different model: BPMN, DF-BPMN, or Activity View, with groups rotating between exercises to avoid learning effect bias. The DF-BPMN group had two iterations, one with text and UML and the other without. The application domains included shipping orders from a website, triage in an emergency room, and loading a book. The goal of this phase was to validate Hypothesis (H1) by comparing DF-BPMN results with and without text and UML, and to validate Hypothesis (H2) using a paired t-test [5], a statistical method to determine significant differences between the means of two related groups while accounting for individual variability.

6.2 Results

The outcome of the study indicates that DF-BPMN has a significant impact on streamlining the design and comprehension of processes and their associated data. This leads to a reduction in the time spent on tasks and improved task accuracy.

During Phase 2, we conducted a quantitative assessment of the effectiveness of DF-BPMN by comparing it to BPMN and Activity View. We measured the time taken to complete each task for each participant and counted the number of accurately answered questions, adhering to strict criteria for both accuracy and completeness of responses. Finally, we employed a paired t-test to analyze the results, comparing DF-BPMN to BPMN and DF-BPMN to Activity View, where the execution times of one exercise using DF-BPMN were contrasted with those of the same exercise using another model (BPMN/Activity View).

The results displayed in Fig. 7 show that the exercises with DF-BPMN took an average of 12 min, and 71% of the answers were evaluated as correct. In contrast, the results with Activity View took an average of 17 min, with only 40% of the answers being correct. With BPMN, the results took 22 min, with only 38% of the answers deemed correct. These results support hypothesis **H2**. In fact, by applying the paired t-test to the measured correct answers between DF-BPMN and BPMN, the p-value $= 8 \times 10^{-5} < 0.05$. Furthermore, between DF-BPMN and Activity View, the p-value $= 0.003 < 0.05$, indicating that the obtained results illustrated in Fig. 7 are statistically significant, and hypothesis **H2** is satisfied. Indeed, the comparison of DF-BPMN results with and without textual descriptions reveals that without text, 74% of answers are correct, while with textual description, 71% are correct. This suggests that the textual description in our approach is not necessary to understand the model, and hypothesis **H1** is satisfied.

Indeed, all the participants stated that completing the first experimental task without the aid of DF-BPMN was more challenging, and 90% of them responded positively when asked if DF-BPMN improved the modeling of the relationship between processes and data. Next, we asked the participants to rate the usability of DF-BPMN on a scale of 1 to 5, where 1 indicated "strongly disagree" and 5 indicated "strongly agree". The average results of this questionnaire-based interview are presented in Fig. 8.

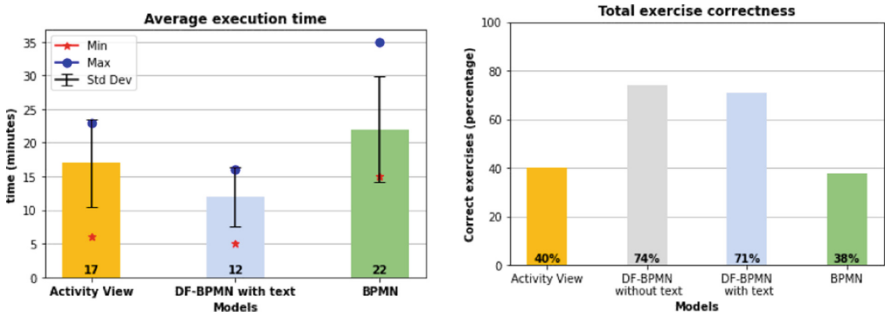

Fig. 7. Average execution time with standard deviation (left) and total percentage of the correct answers (right) for the whole the PHASE 2.

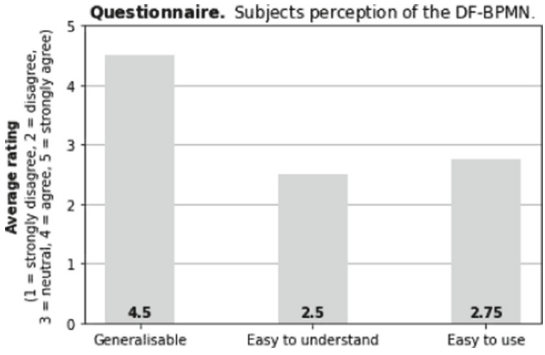

Fig. 8. Average rating of subjects perception of the DF-BPMN.

7 Discussion and Conclusion

Low-code development platforms (LCDPs) aim to simplify software systems' development by providing easy-to-use graphical interfaces. The system behaviors are defined through available data handling and workflow mechanisms enabling the specification of business processes from users that do not have strong programming skills. Moreover, a clear understanding of the data involved in business processes is critical to reduce the mistake in the implementation of the model.

Although LCDPs in business process are most widely used BPMN as business process model, but it has limitations as a data-flow language. Specifically, BPMN underspecifies the data store and does not support the relationship between different data types. Additionally, it does not represent users and external systems, which can lead to misunderstandings in the process model.

To address these issues, we proposed DF-BPMN, a first step in low-code solutions that connects process and data diagrams by using the Activity DataFlow, an extension of the standard BPMN activity. DF-BPMN provides insights into how data flows through a process and identifies areas for data-related improvements, enabling process designers to model the complex relationships between

processes and data. DF-BPMN allows developers to represent data in a graphical format, improving collaboration between business and developers.

Based on our evaluation in Sect. 6, DF-BPMN is a promising approach for supporting process designers in modeling the complex relationships between processes and data. DF-BPMN is simple to understand without requiring additional information, and because it is a visual language, we are confident that the information will be understood by everyone. However, humans still require assistance in creating a model using DF-BPMN, which can be resolved using AI assistance. Indeed, we are working on the second step of low-coding by generating an execution code to be used in the engine for the execution of the process.

In conclusion, by integrating low-code and business process modeling, DF-BPMN provides the first step of low-code solution that bridges the gap between process and data diagrams and enables a clear understanding of the data involved in business processes. It has the potential to improve overall efficiency and effectiveness by identifying areas for data-related improvements.

References

1. van der Aalst, W.M.P.: Object-centric process mining: dealing with divergence and convergence in event data. In: Software Engineering and Formal Methods - 17th International Conference, SEFM (2019)
2. van der Aalst, W.M.P., van Hee, K.M.: Workflow management: models, methods, and systems. Cooperative information systems (2002)
3. Aguilar-Savén, R.S.: Business process modelling: review and framework. Int. J. Prod. Econ. **90**, 129–149 (2004)
4. Combi, C., Oliboni, B., Weske, M., Zerbato, F.: Conceptual modeling of processes and data: Connecting different perspectives. In: Conceptual Modeling - 37th International Conference, ER (2018)
5. Field, A.: Discovering statistics using IBM SPSS statistics. SAGE (2013)
6. Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Delta-BPMN: a concrete language and verifier for data-aware BPMN. In: Business Process Management - 19th International Conference, BPM 2021 (2021)
7. De Giacomo, G., Oriol, X., Estañol, M., Teniente, E.: Linking data and BPMN processes to achieve executable models. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 612–628. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_38
8. OMG: Business Process Model and Notation (BPMN), Version 2.0 (2011). http://www.omg.org/spec/BPMN/2.0
9. OMG: OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1 (2011). http://www.omg.org/spec/UML/2.4.1
10. Sahay, A., Di Ruscio, D., Iovino, L., Pierantonio, A.: Analyzing business process management capabilities of low-code development platforms. Practice and Experience, Software (2022)
11. Zadeh, L.A.: Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. Fuzzy sets and systems (1997)