Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 7579515, 11 pages
https://doi.org/10.1155/2022/7579515

Research Article

Recovering Latent Data Flow from Business Process Model Automatically

Sheng Ye[1,2] **Jing Wang**[1,2] **Sikandar Ali**[3] **Hasan Ali Khattak**[4] **Chenhong Guo**[1,2] **and Zhongguo Yang**[1,2]

[1]*School of Information Science and Technology, North China University of Technology, Beijing 100144, China*
[2]*Beijing Key Laboratory on Integration and Analysis of Large-Scale Stream Data, Beijing 100144, China*
[3]*Department of Information Technology, The University of Haripur, Haripur 22620, Khyber Pakhtunkhwa, Pakistan*
[4]*School of Electrical Engineering and Computer Science, National University of Sciences and Technology (NUST), Islamabad 44000, Pakistan*

Correspondence should be addressed to Sikandar Ali; sikandar@uoh.edu.pk
and Hasan Ali Khattak; hasan.alikhattak@seecs.edu.pk

Received 18 February 2022; Accepted 4 June 2022; Published 20 June 2022

Academic Editor: Yingjie Wang

Process-driven applications evolve rapidly through the interaction between executable BPMN (Business Process Modeling and Notation) models, business tasks, and external services. Given these components operate on some shared process data, it is imperative to recover the latent data by visiting relation, which is known as data flow among these tasks. Data flow will benefit some typical applications including data flow anomaly checking and data privacy protection. However, in most cases, the complete data flow in a business process is not explicitly defined but hidden in model elements such as form declarations, variable declarations, and program code. Some methods to recovering data flow based on process model analysis of source code have some drawbacks; i.e., for security reasons, users do not want to provide source code but only encapsulated methods; therefore, data flows are difficult to analyze. We propose a method to generate running logs that are used to produce a complete data flow picture combined with the static code analysis method. This method combines the simple and easy-to-use characteristics of static code analysis methods and makes up for the shortcomings of static code analysis methods that cannot adapt to complex business processes, and as a result, the analyzed data flow is inaccurate. Moreover, a holistic framework is proposed to generate the data flow graph. The prototype system designed on Camunda and Flowable BPM (business process management) engine proves the applicability of the solution. The effectiveness of our method is validated on the prototype system.

1. Introduction

In the discipline of Business Process Management and Automation, BPMN, based on ISO standards, is widely adopted as a modeling language for workflow and executable business process models [1]. When used as a common business process modeling language for domain experts and others in the industry, BPMN helps to better business-IT alignment.

The business process model explicitly describes the control flow that consists of events (depicted by circles), tasks (by rectangles), and gateways (by diamond shapes). The data flow that is represented by data objects (depicted by parallelogram) and is associated with tasks as their input or output, respectively, is often overlooked. Recently, many researchers are starting to take the data flow seriously and use it to create greater value especially, in the data privacy protection and data anomaly detection tasks [2]. Referring to Figure 1, the black part is the business process diagram, which illustrates the execution sequence of the business process. The data flow is an ordered sequence of bytes with a start and an endpoint, including an input and output flow. The data involved in a business process may be read and written sequentially by multiple tasks. In Figure 1, the entire data flow is represented by blue. The blue part denotes the data flow involved in the business process, which can be variables, forms, or even other data structures permitted by the BPMN2.0 standard.

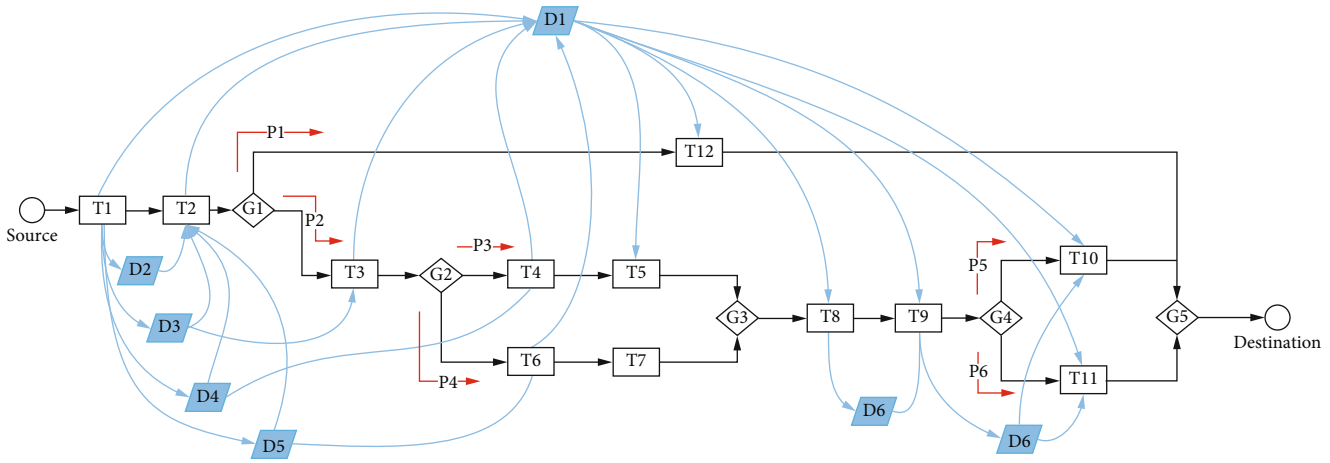

Figure 1: Sample diagram of complete data flow.

It is difficult to find all data flows in the business process. In related literature to BPM, there are some works [3] that are all about anomaly detection and processing of data flow in BPM. But unfortunately, most of them ignore the process of getting data flow from BPM. Schneid et al. [4] propose a method of using static code analysis tools to detect data flow in a simple business process without branches. The approach that writes rules through code analysis tools instead of human actions to find the data flow is partial, incoherent, cumbersome, and inaccurate when dealing with particularly complex BPM files [5]. We propose a new method to analyze data flow based on analyzing the reading and writing of data from logs of running processes. Static code analysis methods have some shortcomings such as inaccurate data flow detection, inability to adapt to complex business processes, and time-consuming. The method proposed in this paper not only solves these problems but also is efficient and convenient. Complete data flow requires complete business process logs. Some activities or tasks can generate data flow. Therefore, the challenge of this article is to ensure that every element of the business process is run once (or traveled along every path). As shown in Figure 1, two branch instances (P1, P2) are generated when the process engine passes through the G1 gateway, G2 gateway (P3, P4), and G4 gateway (P5, P6). Compared with previous work, this paper has the following three contributions:

(i) Suit for complex BPM: it is suitable for a variety of complex business process files, rather than a single branchless path as in previous experiments

(ii) A business process automation tool: automated deployment runs business processes faster and more accurately than manual judgment and simple static source code analysis

(iii) Complete data flow: it can analyze the complete-data flow by providing the executable file of the external agent task

We developed a tool (named BP-Dataflow) that consists of a front-end user interaction page, a business process automation running program, and a data flow drawing framework to generate a complete data flow diagram according to the necessary information submitted by users. Firstly, users need to submit corresponding BPM files, initial parameters, and some similar Jar packages. Moreover, the proposed program parses these files and parameters to drive the engine program that can be parsed by the process engine to run and generate log files. Finally, the drawing program will analyze the read/write records of data in the log to automatically draw a complete data flow graph.

The rest of this paper is structured as follows: Section 2 discusses the related work. Section 3 introduces the proposed method to implement the system. Section 4 evaluates the proposed method's effectiveness and applicability. Finally, Section 5 concludes the paper and gives a brief outlook on future work.

2. Related Work

2.1. Recovery of Data Flow. Most of the existing research on business process management focuses on the perspective of workflows and ignores the importance of the data flow perspective. However, various constraints in business processes and workflows often depend on the correctness of data. If the control flow is correct, then the data flow may not be correct; therefore, it is very important to analyze and restore the data flow in business processes and workflows [6]. Moreover, there is less research on data flow recovery. Recently, some studies [4] have been conducted to analyze the problem from the perspective of data flow recovery. By considering an integrated view, data flow recovery has been studied by Schneid et al. [4]. The main idea is to restore the data flow in the business process by merging the calling diagram of external service and the resulting diagram of associated data operation into the control flow of the process model. Chaim et al. [5] model data flow tests as a data flow analysis framework to quickly discover data flow relationships using efficient algorithms. Guo et al. [7] proposed a new data perspective of workflow management and a mathematical technique to solve the problem of data exchange in business process centralization in a dynamic environment to better recover data flow. In another study, Schneid et al. [8]

presented an integrated DFA diagram based on process models and artifacts (such as source code or user forms) to indicate operational relationships between data flows and nodes. Ji et al. [9] proposed a method to analyze data flow in the BPEL specification business process and ensure the correctness of data flow based on XCFG (eXtended Control Flow Graph). Amme et al. [10] proposed a CSSA method to extract data flow information from WS-BPEL, the Web Services Business Process Execution Language.

However, the above-mentioned methods own some obvious disadvantages. First of all, they research the anomaly detection and processing of data flow, but they do not elaborate on the method of data flow acquisition. Secondly, majority of these methods of obtaining data flow are through static code analysis, which requires a lot of time to fill in matching rules and is difficult to verify the accuracy systematically. Therefore, the integrity and accuracy of the data flow obtained through these methods are questionable.

2.2. Application of Data Flow. Modeling and validation of data flow are very important for anomaly detection; thus, Chadli et al. [2] explain three approaches that are used for detecting data flow anomalies and its proper method and tools. Tao and Fang [11] opt for workflow nets with tables (WFT-nets) to model workflow systems and detect inconsistent data. Liu et al. [12] have proposed a Petri net-based approach to model and analyze data flows. Ramon-Cortes et al. [6] build a Distributed Stream Library supporting the integration of workflow and data flow to meet the needs of new Data Science workflows. Xiang et al. [13] have proposed a PN-DOS model to reduce the accessibility of graphs to quickly detect data flow errors and ensure the correctness of business processes. Zhai et al. [14] have proposed a novel approach of data flow optimization to determine the optimal partition of data flow in BEPL processes, which is complex and accurate. Kabbaj et al. [15] have introduced an approach to detect data flow errors in business process models by validating and correcting fragments of the model as the model is modeled.

In a nutshell, there are a lot of literature on data flow application, such as data flow anomaly and privacy protection, so data flow is very important for some applications. However, the recovery of data flow is described in insufficient detail in the previous work. Therefore, this article targets those gaps in the recovery of data flow. The framework proposed in this article is described in detail in the next section.

3. Method

3.1. Framework. Figure 2 shows the overall operating architecture. This paper designs the architecture from the point of view of full automation. Moreover, a method is proposed to automatically generate the executable code depending on parsing the BPM file. The framework consists of three main steps: entering related files and parameters, algorithms for generating executable code, and generating a complete data flow diagram. In the first step, the user submits a standard BPM file, initial variable, and external agent files. Finally, the Pydotplus-Python library is utilized to generate a complete data flow diagram by analyzing process running logs.

3.2. Basic Concept. The business process engine used for the experiment in this paper is Flowable, and the tools presented in this article are capable to work with the most popular Camunda engine. BPM defines multiple data types, such as variables, forms, DMN, and so on. These can be resolved using specific matching rules. It is important to note that different activities in the business process have unique ways of reading and writing data.

(i) User task: as the name implies, the user assigned to the task must view the corresponding data view before deciding whether to complete the task. In some cases, completing the task requires writing data such as variables and forms. These can be explicitly parsed in a file or analyzed from the running log

(ii) Service task: the service task that is independent of the engine is customized by the user, which requires the user to provide the executable program that can be called so that the algorithm proposed in this paper can run smoothly and accurately find the corresponding data flow

(iii) Script task: it is written in scripting languages like JavaScript and Python. The solution is similar to a service task and requires the user to provide the corresponding execution file

(iv) DMN task (business rule task): as with the above activities, which are independent of the engine, the user is required to provide the corresponding DMN file. It should be noted that all data involved in this task will be read once and then written again after processing with corresponding rules. Therefore, it is better to analyze the data flow of this task

3.3. Work Details. This paper presents an algorithm, i.e., Algorithm 1, for combining static code analysis and dynamic log analysis. The static approach uses the regular expressions to match the explicit data flow as shown in Figure 3 while the dynamic approach addresses the implicit data flow. Data flow read/write relationships in logs can also be extracted using regular expressions. In order to better adapt to each BPM file, there are many points to consider. The key point is how to ensure that the complex path is covered. In fact, almost all the papers on recovering data flow do not consider multipath, since the BPM file used for the experiment is relatively simple. In this paper, we use a graph structure to temporarily store information in business processes and data flow information. Pydotplus, Python's powerful drawing library, outputs a complete data flow graph with the graph as input. In the following sections, I describe the three steps of the framework.

First step: the front-end interface: this paper uses the React front-end framework to write an interface to interact with users. The React framework is extremely powerful, with many ready-made component libraries that are closely aligned with background operations. The display interface is shown in Figure 4. Users just need to submit standard BPM files and process execution variables and independent external proxy class files.

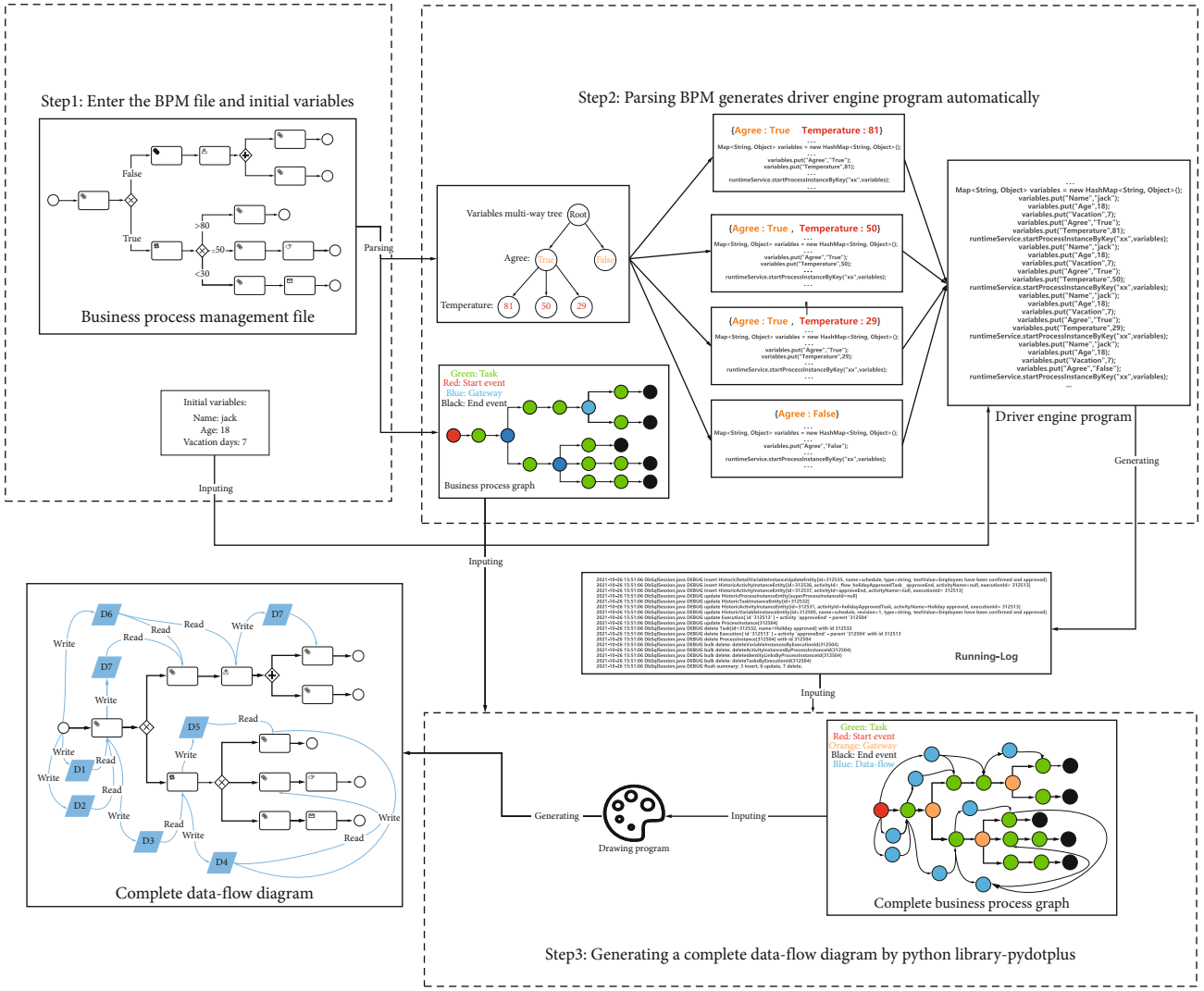

Figure 2: An automatic generating BPM data flow tool: BP-Dataflow.

Second step: parsing BPM to generate driver-engine program automatically: we need to parse the ID of process, task, and gateway in the business process file to facilitate the program to drive the engine.

$$\begin{aligned} &<\text{process id} = ''\text{holidayRequest}''\cdots> \\ &\text{runtimeService.startProcessInstanceBykey}\left(''\text{holidayRequest}'', \text{variables}\right); \end{aligned} \tag{1}$$

One graph can represent only one business process in a swimming pool. We use the API (Algorithm 1 line 1) provided by the engine to store the business process sequence in the graph structure as shown in Figure 5.

When a business process is running in the engine, only one instance is generated at a time. It is important to generate multiple instances to traverse all paths of the BPM. The type of gateway determines how many instances are launched when you code. The following describes two common gateways.

Exclusive gateways control different branches by depending on the value of a variable (Algorithm 1 line 7). This paper uses the multifork tree to store these variables. Later sections explain why this data structure is used, which is defined here as the multifork tree of variables. Boolean has only two branches, with true and false as the two child nodes of the variable multifork tree, as shown in Figure 6. It needs to find the corresponding condition and value in the file when exclusive gateways are of numeric types. As shown in Figure 7, when the condition of branching is greater than 80, the program automatically adds 1 to this value to get 81. If the branching condition is less than 40, it subtracts one from the value to get 39. If it is a range condition, the

Input: BPMN *bp*, Hashmap *iv*,Graph *cg*,Muti-Tree *mt*,JavaFile *jf*
Output: RunningLog *lg*
1: *cg* = *bp*.Deployment().FindSequence()//Deploy the BPMN file to get the process order
2: **while***temp* = *bp*.ReadLine() **do**
3: *cg*.AddInformation(temp.FindID)
4: if *temp*.ContainStaticData() **then**
5: *cg*.AddDataflow(*temp*.EtxractDataflow())
6: **end if**
7: **if***temp*.JudgeGateway() **then**
8: *mt*.Add(*temp*.GatewayInformation)
9: **end if**
10: **end while**
11: **for***m* = *mt*.TraverseAll() **do**
12: *iv*.Add(*m*)
13: WriteFile(Engine.RunAPI(*iv*), *jf*)
14: **end for**
15: **if** Engine.HasUserTask() **then**
16: WriteFile(Engine.CompleteAPI(), *jf*)//Complete human tasks
17: **end if**
18: WriteFile(FixedCode, *jf*)//Write the fixed template code to the Java file
19: RUN(*jf*)//Run the Java file

Algorithm 1: Generating driver-engine program automatically.

```
<flowable:executionListener expression=" ${execution.getVariable( 'employee' )}" event=" start" />
<flowable:executionListener expression=" ${execution.getVariable( 'nr0fHolidays' )}" event=" start" />
<flowable:executionListener expression=" ${execution.getVariable( 'description' )}" event=" start" />
<flowable:executionListener expression=" ${execution.getVariable( 'schedule' ,' Manager approval has been completed' )}" event=" end" />
```

Figure 3: Explicit data streams in BPMN files.

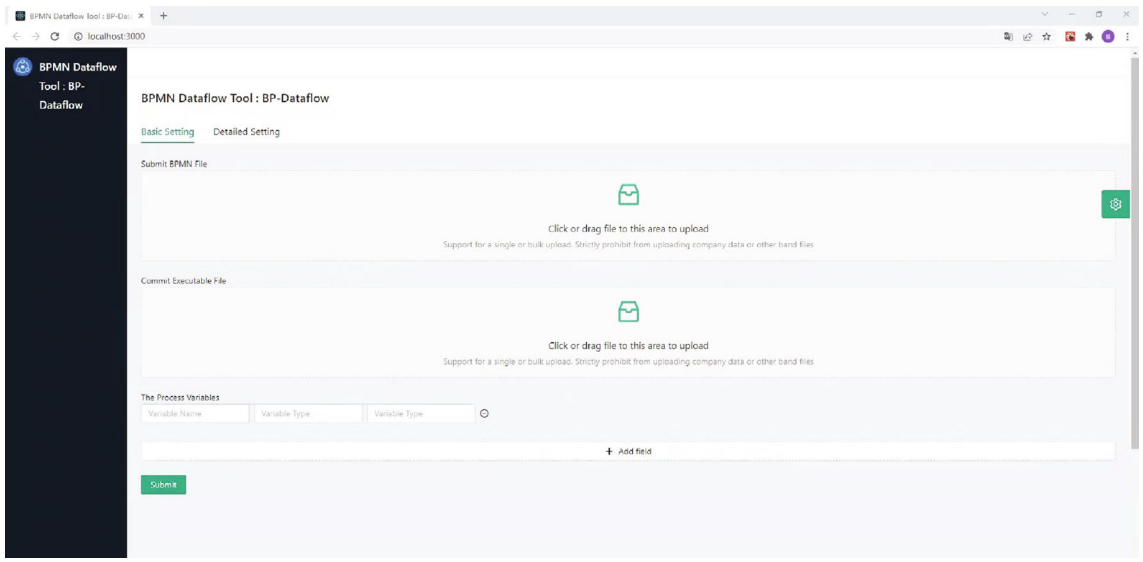

Figure 4: The front-end interface.

program will automatically take the value that meets the condition. The condition for branching is equal to 50, so it will take 50, eventually storing each of these variables on the children of the variable multifork tree.

Parallel gateways, as the name implies, allow the engine to continue execution separately along each branch. The engine automatically generates the number of instances of branches; therefore, there is no need for the variable multifork trees to assist storage. One of the most important purposes of this paper is to convert variables involved in the original business process into the multifork tree of variables as shown in Figure 8.

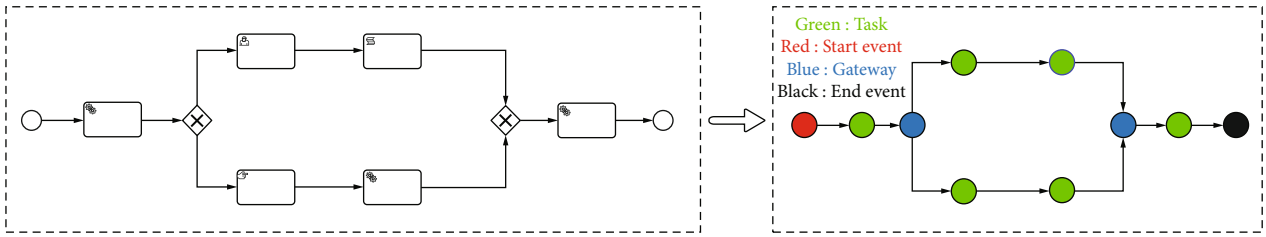

Figure 5: Store business process sequence by using a graph.

```
<exclusiveGateway id=" decision" />
<sequenceFlow id=" s2" sourceRef=" decision"
targetRef=" externalSystemCall" >
    <conditionExpression xsi:type=" tFormalExpression" >
        <![CDATA[
        ${approved}
        ]]>
    </conditionExpression>
</sequenceFlow>
<sequenceFlow id=" s3" sourceRef=" decision" targetRef=" sendRejectionMail" >
    <conditionExpression xsi:type=" tFormalExpression" >
        <![CDATA[
        ${!approved}
        ]]>
    </conditionExpression>
</sequenceFlow>
```

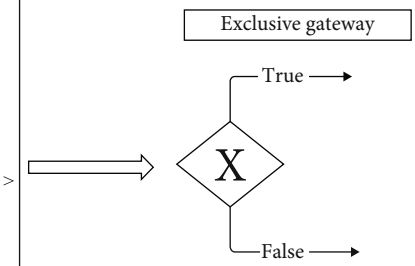

Figure 6: Parse a gateway with a Boolean branching condition.

```
<sequenceFlow id=" d1" sourceRef=" decision"
targetRef=" externalSystemCall" >
    <conditionExpression xsi:type=" tFormalExpression" >
        <![CDATA[
        ${temperature>80}
        ]]>
    </conditionExpression>
</sequenceFlow>
<sequenceFlowid=" d2" sourceRef=" decision" targetRef=" sendRejectionMail" >
    <conditionExpression xsi:type=" tFormalExpression" >
        <![CDATA[
        ${temperature<40}
        ]]>
    </conditionExpression>
</sequenceFlow>
<sequenceFlowid=" d3" sourceRef=" decision" targetRef=" sendRejectionMail" >
    <conditionExpression xsi:type=" tFormalExpression" >
        <![CDATA[
        ${temperature=50}
        ]]>
    </conditionExpression>
</sequenceFlow>
```

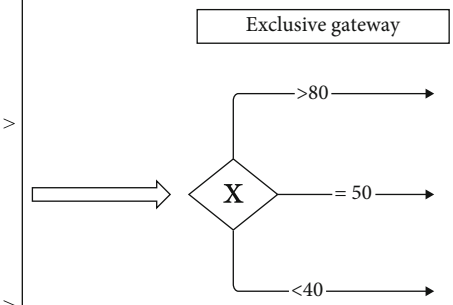

Figure 7: Parse a gateway with a numerical branching condition.

Finally, information about each path from the root node to the child node in the variable multifork tree, respectively (line 11), is {initial variables, approved: true, temperature: 81}, {initial variables, approved: true, temperature: 29}, {the initial variable, approved: true, temperature: 50}, and {initial variables, approved: false}. Call engine API (line 13) to launch 4 instances that, respectively, carry four variables' sequence to go through all the paths of the business process; other running code is fixed and can be automatically generated by using templates (line 18). By analyzing the log files generated after the process runs (line 19), we can easily find which activities read and write data that can be variables, forms, etc. As shown in Figure 9, the activityId is the unique identification number of the activity, Revision is the number of times the variable was written to clarify how many times it was written, while the textValue is what was written. Finally, the read and write of data are stored in the data flow HashMap for the drawing program to generate a data flow graph.

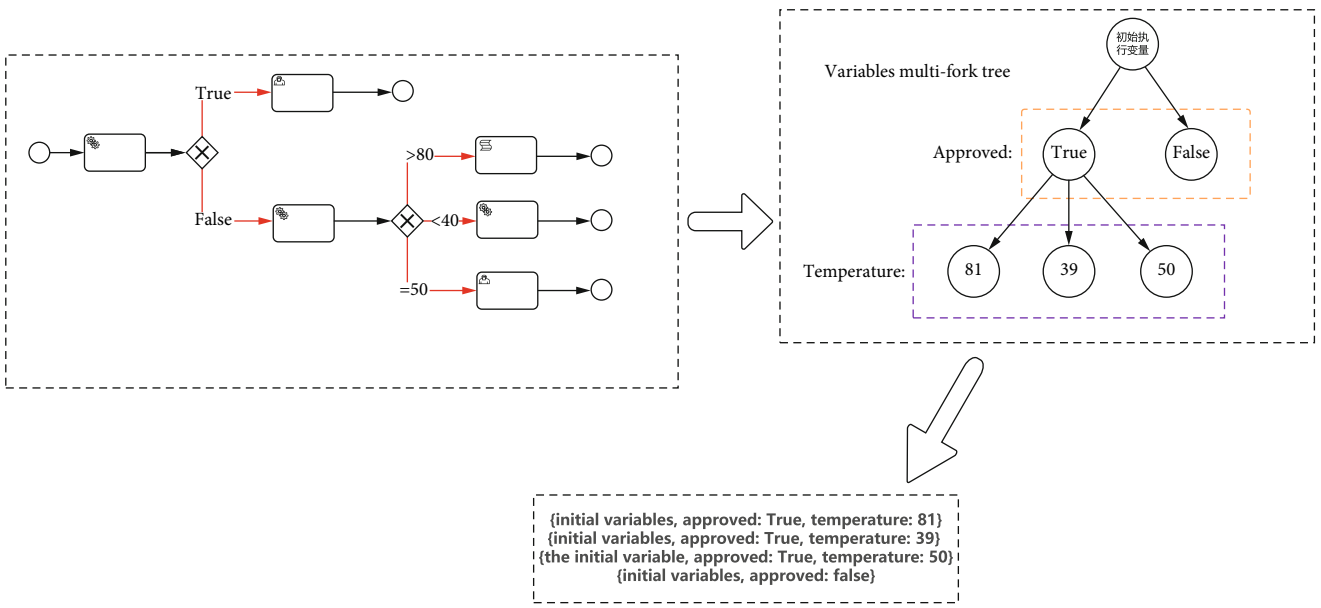

Figure 8: Traverse the path information from the root node to each leaf node of a variable multifork tree.

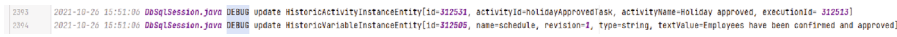

Figure 9: The data flow in running logs.

Third step: automatic plotting program: this drawing program uses Pydotplus—a Python drawing library. Pydotplus accepts the business process sequence graph and data flow HashMap mentioned in the preceding section as input and then automatically generates a complete business process model data flow diagram using the front-end interface.

4. Evaluation

To assess the effectiveness of our system in business processes, we test 5 real-world business process cases in which two cases are presented in detail. We evaluate the system in terms of two sets of measures:

(i) Accuracy. What is the fraction of correctly analyzed data flow in the complete data flow (precision rate)?

(ii) Performance. Can we cope with the real-world business process? How long does it take when submitting a BPM File to generate a data flow diagram need?

4.1. Evaluation Process Cases. As shown in Table 1, we have carefully listed the amount of data flow of five real-world business processes for subsequent analysis and evaluation.

The first case to verify the accuracy of the algorithm is the example mentioned in the official Flowable manual (business process for employees to apply for leave) as shown in Figure 10. As the example is too simple to have an obvious data flow, we add some read/write operations of data flow without changing the real business logic. The user needs to submit three initial variables including the employee's name, number of days leave requested, and reason for requesting leave, when the process runs. The manager needs to look at the information submitted by the employee to decide whether to agree or disagree. Eventually, the data flow diagram is shown in Figure 11.

The second case is the insurance business process as shown in Figure 12. The process is complex, with multiple swim lanes, script tasks, business rules tasks, and subprocesses. These elements do not affect the use of the algorithm proposed in this paper. Finally, the generated data flow diagram is shown in Figure 13.

4.2. Accuracy

4.2.1. Experimental Setup. To evaluate the precision of the proposed system, we compare the attributes of all aspects of the data flow that are automatically produced by the system with the values judged by the human in the above table. Then, we calculate five formulas to assess the accuracy of the system.

$$\text{PathNumberPrecisionRate} = \frac{\text{SystemToCalculatePath}}{\text{ActualPath}}, \tag{2}$$

$$\text{DataNumberPrecisionRate} = \frac{\text{SystemToCalculateData}}{\text{ActualData}}, \tag{3}$$

$$\text{Data} - \text{read} - \text{flowPrecisionRate} = \frac{\text{SystemToCalculateDataRead}}{\text{ActualDataRead}}, \tag{4}$$

Table 1: The evaluation index of 5 real business process.

Business process name	Path number	Data number	Data-read-flow number	Data-write-flow number	Data flow number
Employees apply for vacation	2	5	7	7	14
Customer insurance	5	9	9	12	21
The ship trajectory	8	11	12	13	25
Personalized intelligent medical care	12	8	8	11	19
Transportation of dangerous chemicals	13	16	14	18	32

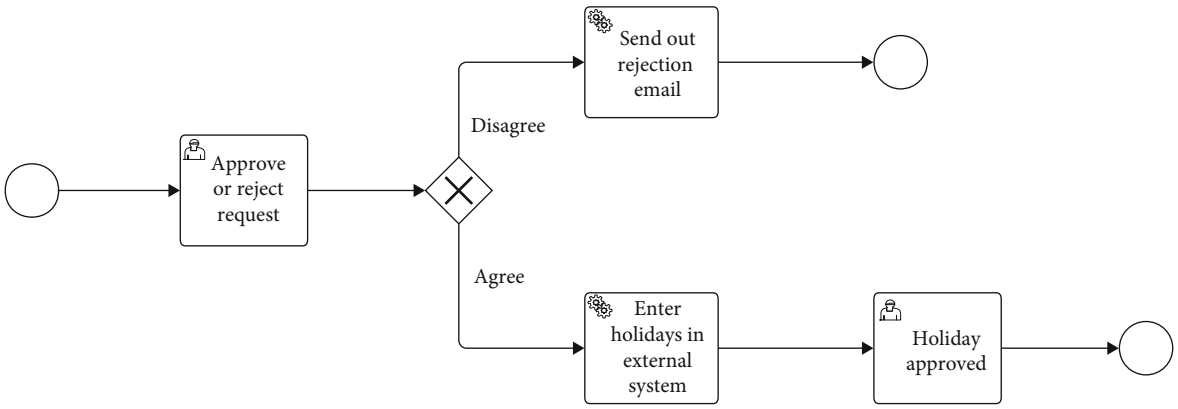

Figure 10: Employee leave process.

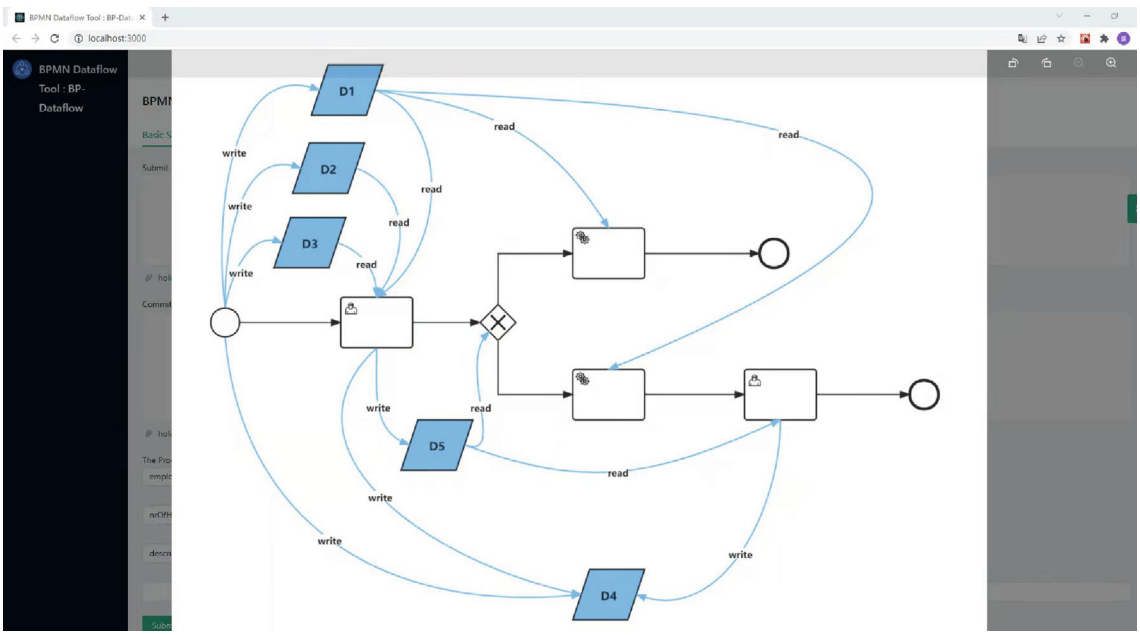

Figure 11: Data flow diagram of the employee leave process.

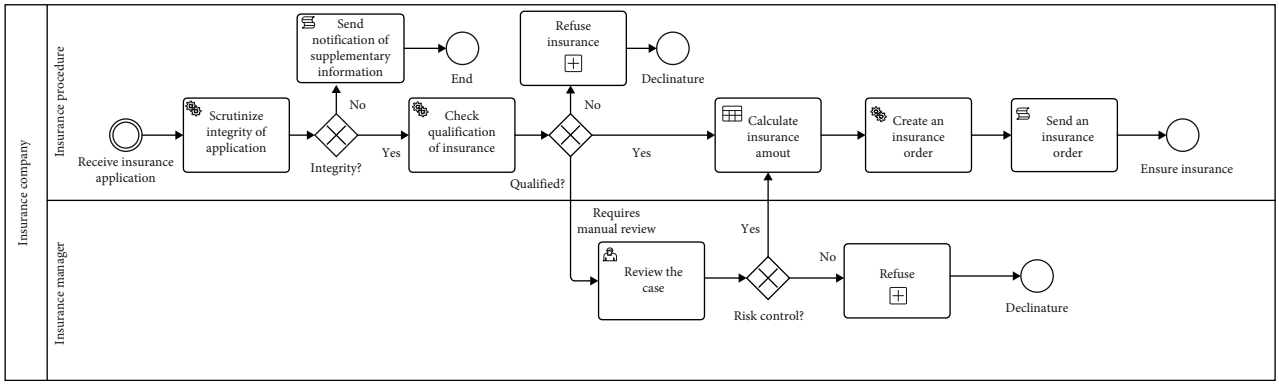

Figure 12: Insurance business process.

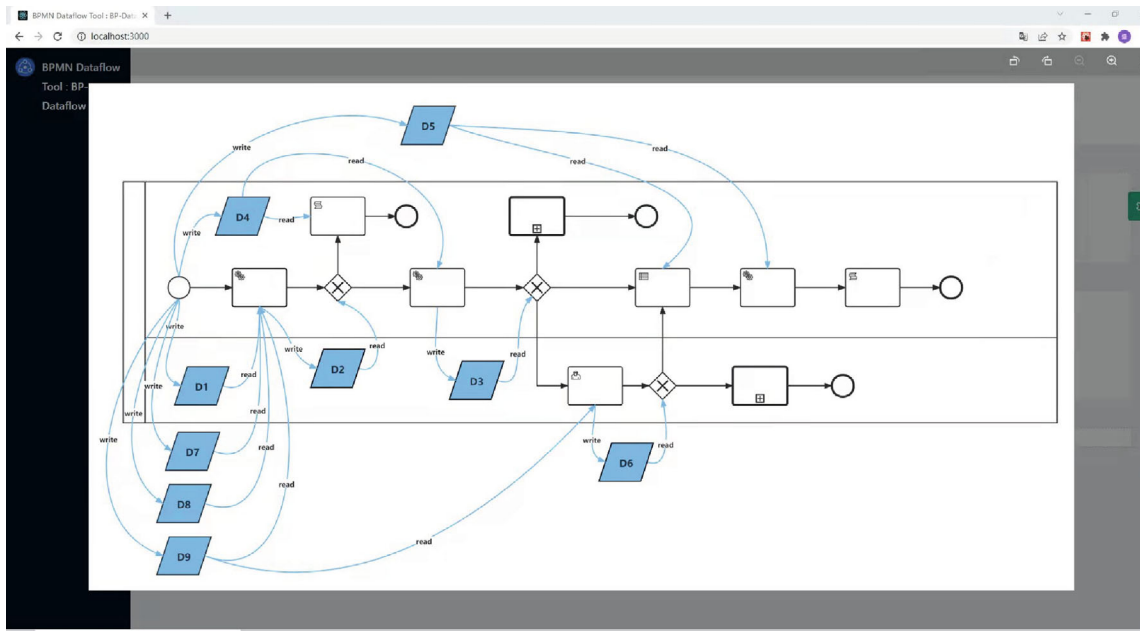

Figure 13: Data flow diagram of insurance business process.

$$\text{Data} - \text{write} - \text{flowPrecisionRate} = \frac{\text{SystemToCalculateDataWrite}}{\text{ActualDataWrite}}, \tag{5}$$

$$\text{DataflowPrecisionRate} = \frac{\text{SystemToCalculateDataflow}}{\text{ActualDataflow}}. \tag{6}$$

The algorithm proposed in this paper guarantees full path coverage, which is a necessary condition for data-flow integrity. As shown in Equation (2), it is used to measure the accuracy of path coverage. The other four equations are designed to evaluate the accuracy of the data flow in more detail.

4.2.2. Findings. As shown in Table 2, the test results for each of the five business process examples are the same as the real value with 100 percent accuracy. Based on these numbers, we can conclude that the number of paths and data are correct, which makes the precision of the system 100%.

4.3. Performance

4.3.1. Experimental Setup. We intend to evaluate the performance of the proposed system through total runtime that includes generating a data flow diagram from user submission to the system.

4.3.2. Findings. We first focus on the time spent on running the business process. It is important to understand that the complexity of business processes is a combination of the multipath and data flow complexity of the business processes. Furthermore, running is directly proportional to the complexity of the business process. From the results, we can see that as business processes become more complex, they take longer to run.

Table 2: Statistics of various indicators and running time of the data flow.

Business process name	Calculate path number	Actual path number	Calculate data number	Actual data number	Calculate data-read-flow number	Actual data-read-flow number	Calculate data-write-flow number	Actual data-write-flow number	Calculate data flow number	Actual data flow number	Run times (s)
Employees apply for vacation	2	2	5	5	7	7	7	7	14	14	4.37
Customer insurance	5	5	9	9	9	9	12	21	21	21	5.21
The ship trajectory	8	8	11	11	12	12	13	13	25	25	6.22
Personalized intelligent medical care	12	12	8	8	8	8	11	11	19	19	5.77
Transportation of dangerous chemicals	13	13	16	16	14	14	18	18	32	32	8.32

Based on the analysis, we can conclude that by comparing with piecing together a complete data flow diagram from the data flow analyzed by each independent task, it is more accurate and efficient to execute the business process by automatically writing the executable program to get the data flow diagram.

5. Conclusion and Future Work

Previous work on the data flow is limited to static code analysis, which is not accurate and time-consuming. Based on previous work, this paper proposes a method to recover the data flow in the business process by combining static code analysis technology and dynamic running log analysis method, which not only has high accuracy and short time, but also, the tool developed according to this method is simple and efficient. This hybrid architectural approach is based on the Flowable engine and platform. Of course, with some modifications, we can run BPM files that support the Camunda engine. Essentially, their underlying API is the same, but the namespace in the XML is different. In terms of evaluation, the paper uses 5 real-world business processes, and the results are promising.

In summary, BPM files are composed of basic and common BPM elements that are analyzed in this paper. However, some tasks and events defined by the entire BPMN2.0 specification have not been analyzed here, such as intermediate events, boundary events, and tasks such as Web tasks, shell tasks, and listeners. This is also the direction of our future work, as we continue to refine the architecture to fully adapt to all specifications defined by BPMN2.0.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Key R&D Plan (No. 2018YFB1402500), the Key Program of the National Natural Science Foundation of China (No. 61832004), and the International Cooperation and Exchange Program of the National Natural Science Foundation of China (No. 62061136006).

References

[1] X. Kechagioglou, R. Lemmens, and V. Retsios, "Sharing geoprocessing workflows with Business Process Model and Notation (BPMN)," in *Proceedings of the 2019 2nd International Conference on Geoinformatics and Data Analysis*pp. 56–60, Prague, Czech Republic, 2019.

[2] N. Chadli, M. I. Kabbaj, and Z. Bakkoury, "Detection of data-flow anomalies in business process an overview of modeling approaches," in *Proceedings of the 12th International Conference on Intelligent Systems: Theories and Applications*pp. 1–6, Rabat, Morocco, 2018.

[3] N. Trčka, W. M. Van der Aalst, and N. Sidorova, "Data-flow anti-patterns: discovering data-flow errors in workflows," in *International Conference on Advanced Information Systems Engineering*, pp. 425–439, Springer, Berlin, Heidelberg, 2009.

[4] K. Schneid, H. Kuchen, S. Thöne, and S. Di Bernardo, "Uncovering data-flow anomalies in bpmn-based process-driven applications," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pp. 1504–1512, New York, United State, 2021.

[5] M. L. Chaim, K. Baral, J. Offutt, M. Concilio, and R. P. Araujo, "Efficiently finding data flow subsumptions," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pp. 94–104, Porto de Galinhas, Brazil, 2021.

[6] C. Ramon-Cortes, F. Lordan, J. Ejarque, and R. M. Badia, "A programming model for hybrid workflows: combining task-based workflows and dataflows all-in-one," *Future Generation Computer Systems*, vol. 113, no. 7, pp. 281–297, 2020.

[7] X. Guo, S. X. Sun, and D. Vogel, "A dataflow perspective for business process integration," *ACM Transactions on Management Information Systems (TMIS)*, vol. 5, no. 4, pp. 1–33, 2015.

[8] K. Schneid, S. Di Bernardo, H. Kuchen, and S. Thöne, "Data-Flow analysis of BPMN-based process-driven applications: detecting anomalies across model and code," *ERCIS Working Paper*, vol. 2021, no. 38, 2021.

[9] S. Ji, B. Li, and P. Zhang, "XCFG based data flow analysis of business processes," in *2019 5th International Conference on Information Management (ICIM)*, pp. 71–76, Cambridge, UK, 2019.

[10] W. Amme, A. Martens, and S. Moser, "Advanced verification of distributed WS-BPEL business processes incorporating CSSA-based data flow analysis," *International Journal of Business Process Integration and Management*, vol. 4, no. 1, pp. 47–59, 2009.

[11] X. Tao and X. Fang, "Detecting data inconsistency based on workflow nets with tables," *IEEE Access*, vol. 9, pp. 81740–81749, 2021.

[12] C. Liu, Q. Zeng, H. Duan et al., "Petri net based data-flow error detection and correction strategy for business processes," *IEEE Access*, vol. 8, pp. 43265–43276, 2020.

[13] D. Xiang, G. Liu, C. Yan, and C. Jiang, "Detecting data-flow errors based on Petri nets with data operations," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 1, pp. 251–260, 2018.

[14] Y. Zhai, H. Su, and S. Zhan, "A data flow optimization based approach for BPEL processes partition," in *IEEE International Conference on e-Business Engineering*pp. 410–413, Hong Kong, China, 2007.

[15] M. I. Kabbaj, A. Bétari, Z. Bakkoury, and A. Rharbi, "Towards an active help on detecting data flow errors in business process models," *International Journal of Computer Science and Applications*, vol. 12, no. 1, pp. 16–25, 2015.