Baltic J. Modern Computing, Vol. 11 (2023), No. 1, 15-33
https://doi.org/10.22364/bjmc.2023.11.1.02

Towards Topological Functioning Model as a Source Model for Event-Driven Solutions

Sai Teja DEHARAM[1], Gundars ALKSNIS[2]

[1] Self-employed, Nellore, Andhra Pradesh, India
[2] Riga Technical University, Riga, Latvia

tejasai6601@gmail.com, gundars.alksnis@rtu.lv

Abstract. Activities an organization performs can be viewed as a sequence of events and responses to them. Event-driven solutions have emerged for event management to help capture events and respond by triggering other events. Nevertheless, designing solutions which conforms with a problem domain is not trivial task. Therefore, holistic understanding of the problem domain is essential. To understand the problem domain, we use formalism of Topological Functioning Model which analyses an organization as a system from computation independent viewpoint and helps to represent holistically both structural and functional aspects of an organization. By applying case study analysis research method, the article discusses how the use of Topological Functioning Model can help to derive an event-driven solution which conforms with the problem domain of an organization.

Keywords: Event-Driven Solution, Event-Driven Architecture, Problem Domain Analysis, Topological Functioning Model.

1. Introduction

Organizational and enterprise systems can be viewed from an event-driven and a real-time perspective. An information technology system that supports the business processes and automation of an organization can be perceived as a sequence of events and responses to them. An event is a change of state of an object in the system. When an event occurs, it triggers other components or events in the system. As modern IT systems are increasingly using real-time analysis, the notations, and methods for producing effective event-driven solutions are in demand (Stopford, 2018; Tiempo Development, 2020).

Although numerous methods were developed for software design using various modeling languages, the Unified Modeling Language (UML) is considered as one of finest to design, document, visualize and specify artifacts of a software intensive system (Platt and Thompson, 2015). However, UML lacks in representing a problem domain from a computation independent viewpoint (Osis and Donins, 2017). This was one of the reasons for the development of Topological UML – an extension of UML with the formalism of Topological Functioning Model. The Topological Functioning Model (TFM) helps to analyze the problem domain (e.g., an organization, an enterprise, a system in general) holistically from the computation independent viewpoint. It specifies

the functionality of a system in the form of topological space, where functional features of the system are shown in the form of a directed graph with cause-and-effect relations among these functional features (Osis and Asnina, 2010). The goal of the present research is to determine the scope and the extent of event-driven solution generation from TFM. By applying case study analysis research method, the article discusses how the knowledge about system's functioning represented with TFM can help to derive an event-driven solution which conforms with the problem domain of an organization (enterprise) thus fostering its digital transformation.

This article is an extended version of the (Deharam and Alksnis, 2022) with different case study, extended related works and validation. It is organized as follows. Section 2 recaps the principles of TFM. Section 3 highlights the principles of event-driven solutions. Section 4 introduces a case study in a library problem domain and discusses proposed mappings between elements of TFM and elements of event-driven solution. Section 5 discusses the validation results while Section 6 discusses related works of TFM transformations to and from other models. Finally, Section 7 concludes the article summarizing main contributions.

2. Topological functioning model

Understanding of the problem domain and developing conforming solution is a primary goal for any software intensive system developer (Osis and Donins, 2017). However, often accurate analysis of the problem domain is not performed carefully enough during software development. The goal of the analysis is to describe the solution complying with to the problem domain, while the design specified, for example, by UML models show how the artifacts of the solution should be implemented. As identified in (Osis and Donins, 2017) and (Osis and Asnina, 2010), UML diagrams are not tailored for analyzing the problem domain from computation independent viewpoint (i.e., the system As-Is). To overcome this, a profile to the UML was introduced in which Topological Functioning Model was integrated and it was called Topological UML (Osis and Donins, 2017).

Originally, TFM was introduced in 1969 by Jānis Osis with the goal to represent system's functionality in a holistic manner by emphasizing topological (connectedness, closure, neighborhood, and continuous mapping) and functional (cause-and-effect relations, cycle structure, inputs, and outputs) characteristics of the system. Over the years, due to its fundamental principles, TFM also has been applied to complex systems diagnostics, mechatronics and embedded systems, model-driven software engineering and problem domain modeling. Few sources where the research in those directions is discussed are (Osis and Donins, 2017; Osis and Asnina, 2010; Osis, 2003). In the rest of this section, we recap the principles behind TFM formalism.

TFM of a system formally is defined in the form of a topological space (Osis and Donins, 2017, Section 4.3.1.1):

$$G = (X, \Theta) \tag{1}$$

In the equation (1), X represents a finite set of functional features X_{id}, whereas Θ represents a topology that satisfies topological characteristics between functional features of a problem domain (e.g., an enterprise, an organization, a system in general). A functional feature represents system's functional characteristic, e.g., a process, a

function, a task, an action, or an activity. Visually, TFM of the system can be represented as a graph where nodes represent the functional features, while arcs correspond to the cause-and-effect relations among them.

The topological space Z represents functioning of the system which is split into a set N to represent the system's inner functional features and a set M to represent external functional features that the system interacts with, or which affect system's environment (Osis and Donins, 2017, Section 4.3.1.1):

$$Z = N \cup M \tag{2}$$

To construct the topological space Z, the information is obtained from various documents, interviews, use cases, ontology and other sources that are related to the problem domain. The separation of the system from the topological space Z is done by performing a closure operation which is done on a set of inner functional features N (Osis and Donins, 2017, Section 4.3.1.1):

$$X = [N] = \bigcup_{\rho=1}^{n} X_{\rho} \tag{3}$$

In the equation (3), X_{ρ} is the adherence point of set N and capacity of X is the number of adherence points of N. An adherence point of the set N is a point, each neighborhood of which includes at least one point from the set N.

Every functional feature X_{id} is defined as a tuple containing the necessary elements needed during construction of TFM (Osis and Donins, 2017, Section 4.3.1.2):

$$X_{id} = \langle Id, A, Op, R, O, Cl, St, PreCond, PostCond, E, Es, S \rangle \tag{4}$$

where:

- Id – Unique identifier
- A – Action of the object O
- Op – Operation that provides functionality defined by action A
- R – Result of action A
- O – Object that receives the result or that is used in action A
- Cl – Class who represents the object O in static viewpoint of the system
- St – State of the object O after the action A
- PreCond – Set of preconditions
- PostCond – Set of postconditions
- E – Entity responsible for performing action A
- Es – Indicates if execution of action A can be automated by software
- S – Subordination (i.e., inner or external functional feature)

However, it is not required that all these fields are used in a particular case.

3. Event-driven solutions

An event is considered as a change of state of an object or in a system. In the surrounding world everything can be related to events and their consequences (Stopford, 2018; Tiempo Development, 2020). To gain competitive value enterprises are transforming their IT systems to event-driven and real-time. We are all surrounded by

events. Everything we do or plan is connected to events around us. For example, in an elevator one presses destination floor number, and the elevator takes to the selected floor. Pressing the desired floor number is an event, and the rest happens without intervention or knowledge about functioning of the elevator. When an event occurs, it brings changes or triggers other components or events in the system to react accordingly. Events are things/processes/steps that trigger a change in the system (Solace PubSub+, 2022).

Events can be processed in several styles; however, the question is how to understand exactly which processing style should be used. Event-driven architecture systems are often a combination of more than one processing styles, so that the solution performs accordingly to the requirements and restrictions set by the problem domain (i.e., it should conform to it). Event processing styles are following (Stopford, 2018; Tiempo Development, 2020):

Simple event processing: A simple event occurs when a significant change occurs in the system. Simple event processing starts the action further down the application stream which makes it ideal for real-time workflow. This processing style is cost effective and reduces latency.

Complex event processing: Complex event processing uses more advanced patterns and techniques to identify and to evaluate large volumes of data occurring from different sources and to respond accordingly. This processing style is used in security systems.

Stream event processing: Stream event processing helps in making in-time decisions. When an event occurs, it brings a significant change in the system, this processing style processes a continuous stream of events and performs fast computations against high-speed streaming data.

Event-driven architecture is among the popular and a go-to architectural approaches used by software developers. From real-time analysis, business process monitoring and online shopping etc., the event-driven architecture has been successfully used in effective IT solutions. (Narkhede, 2018) As it is a software architecture, event-driven architecture has mainly four different flow layers with each of them having respective flow components (Tiempo Development, 2020):

Event channels: Event channels can be compared to a bridge which connects one point to the other. An event channel sends events in the required format to the event producers, event consumers and event subscribers.

Event processing: Events can be processed in several styles. Event processing actions are carried out by evaluating the event against the processing rules. Actions might be storing the event, generating an event.

Event-driven downstream activity: Any event will cause a sequence of downstream activities which are responses to the event. The occurred event can be a push notification by the event processing engines, pull notifications by subscribers, etc.

Event-driven architecture flow components are (Tiempo Development, 2020):

1 Event notification: In this flow, events are only meant to notify that some state change has occurred. In this flow, senders usually have no intention of expecting responses from the consumers. There will be clear boundaries between senders and consumers.

2 Event carried state transfer: In this flow, events eventually carry a whole state instead of just notifying about the state change.

3 Event sourcing: All events happening in the system will be recorded in this flow. Responding to the events accordingly, the state of the system can be re-designed. In this flow, one can recreate historic states.

4 Event producer: As the name suggests, it is the original source of an event. It can be anything for example a business process, electronic object, etc.

5 Event consumer: Event consumers are the subscribers who consume or who will subscribe to the events.

6 Event queue: When an event is sent to a queue, a flow of events begins, and this queue is used to send the events further to an event mediator.

7 Event handler: As the name suggests, all the occurrence of the events will be handled by event handler.

8 Event store: A database which contains all the events stored for the future use if necessary

9 Publish/subscribe messaging: Publish/subscribe messaging technique uses a topic between client and server interaction as an intermediator instead of a queue (see Fig. 1). It is flexible messaging pattern which allows system components to communicate or interact asynchronously. For example, take an online shopping website where a consumer from their online shopping mobile application orders any product. Here the consumer's information is captured by the service and the order placement event is published. Once this event is published, product manufacturer and delivery partner subscribe to the event and interacts with each other until the product is delivered. All the services are linked by an event broker that delivers events from publishers to subscribers and vice versa.

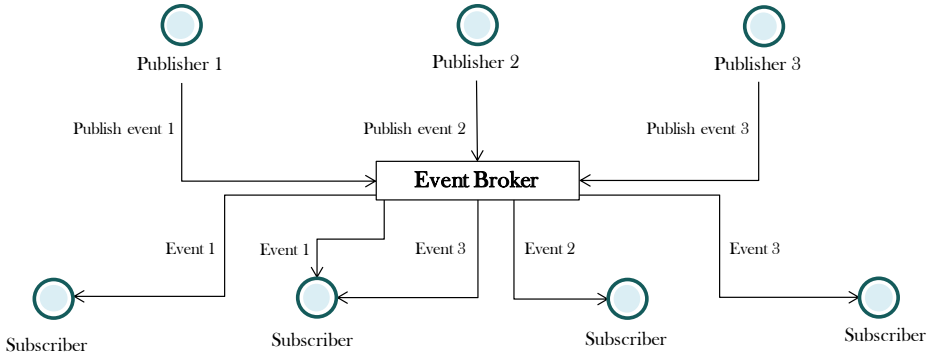

Figure 1. Example of a publish/subscribe message pattern in the event-driven architecture.

In this technique consumers have access to multiple subscriptions to topics and can access the information from their subscribed topics. Subscribers who want to subscribe to the information are classified by the producer. This is a good way of finding out the information being loosely coupled. It is loosely coupled because one can have a subscription once the application is developed or later without changing the course of action or things happening from the producer's side.

We have selected the Solace PubSub+ Platform to demonstrate possible event-driven solution design, how events flow within the solution, how they are visualized, what events need to be published, what events should be subscribed to and how events are managed (Solace PubSub+, 2022). This platform can be used to manage event streams and technologies to design and visualize event-driven solutions. An event mesh which supports their respective cloud then is built. After mapping events to applications, one can analyze the IT architecture, the events which are most used, and which are rarely used, and model them. One can import and export application domain files that define

business areas of the system into the platform. The JSON format is used for writing payload schema along with the source file and can be understood and easily written by software developer or non-technical user.

4. Case study

We have already applied the case study analysis research method to analyze the transformation from TFM to event-driven solution in the context of state traffic department (Deharam and Alksnis, 2022). Here we are using the case study analysis research method in the context of a library problem domain which has been used as example domain in several related research, e.g. (Osis et al., 2008; Nazaruka and Osis, 2018).

4.1. The As-Is problem domain description of a library

Let's assume the following current (As-Is) situation description:

> "When an unregistered person arrives at the library, the librarian creates a new reader account and a reader card. The librarian gives out the card to the reader. When the reader completes the request for a book, they give it to the librarian. If the book copy is available in a book fund, the librarian checks out the requested book from a book fund to the reader. When the reader returns the book copy, the librarian takes it back and returns the book to the book fund. They impose the fine if the term of the loan is exceeded or the book copy is damaged. When the reader pays the fine, the librarian closes the fine. If the book copy is badly damaged, the librarian completes the statement of utilization, and sends the book copy to the utilizer."

According to this description and functional features adopted from (Osis et al., 2008), Fig. 2 shows corresponding As-Is Topological Functioning Model.

4.2. The To-Be problem domain description of a library

As we have seen in the As-Is description, how takes place the procedure starting from the registration of the person until completing the statement of utilization, keeping this procedure in mind, the To-Be description is designed in such a way that most of the process activities is done digitally. So that it is more comfortable free for a reader. In the As-Is case, we see that a person must visit a library in-person to get registered, to get a reader account and to submit a book request form. In the To-Be case, one can register, get a reader account created and submit a book request form online:

> "A person visiting a library website is asked if they are already registered or a new reader. If the person is already a registered reader, they can use their login credentials and enter the library website. If the person is not registered, to become a library reader they are asked to register. On a click, the person opens the registration page, where they fill in their details in the registration form, which also includes a checkbox option that one should check to receive notifications like updates about new arrivals, featured author talks, information about due dates and payments.

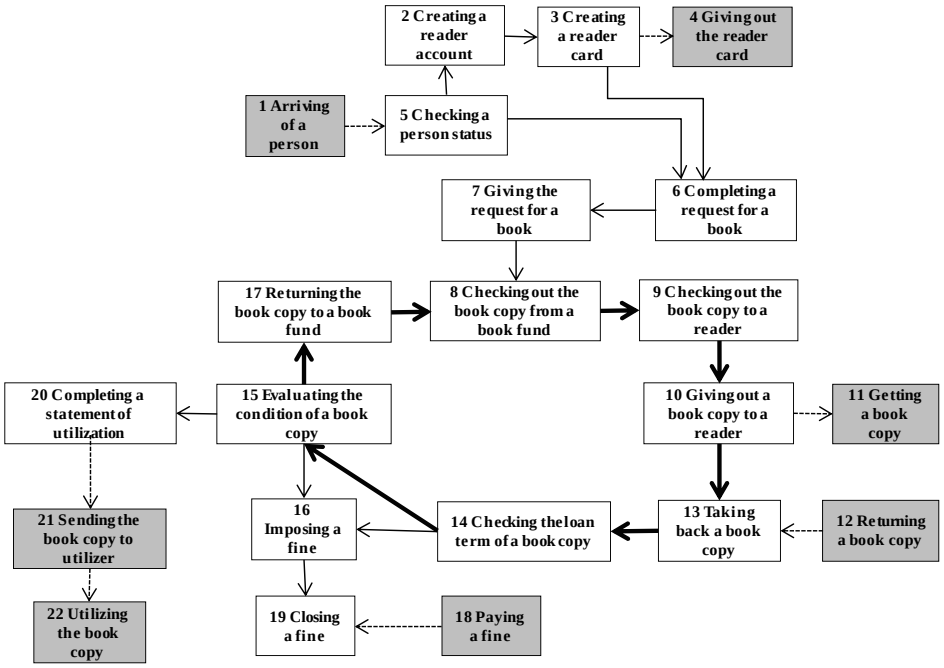

Figure 2. The As-Is TFM of the library problem domain. Nodes represent functional features while arrows represent cause-and-effect relations. The loop consisting of bold arrows represents the main functioning cycle of the system, while dashed arrows relate input or output functional features (greyed) with the functional features of the system.

Once the unregistered person submits the form, the person becomes registered, the library system automatically creates a reader account and a reader ID (scannable QR code) available for downloading and future use. Once the reader account is activated, the reader can browse for the books available in the book fund. On the library website readers also see different panes like new arrivals, featured author talks, and information about book fairs. The reader can also filter their favorite categories (science, detective comics, action & adventure, etc.). The reader also can select and check the number of copies left for an interested book. If no book copy is available, there is an option where the reader can choose to get an alert notification once the book copy becomes available. If the reader agrees to it, they will be put in the book request queue for the requested book.

If the book is available and the reader chooses to loan it, they fill in the book request form available online. Before filling in the book request form, the library system checks reader account for pending fines imposed on the reader. If there is any, the reader is not allowed to loan the book copy until they clear their dues. Once the reader submits the book request form, they are automatically redirected to a page where they can schedule a pick-up time. This must be done in two days from submitting the book request form. Once the reader arrives at the library, the librarian asks for the reader ID. As the librarian scans the reader ID, all the details of the requests are shown, and the librarian gives the book copy from the book fund to the reader. The librarian then updates the status in the library

system that the book copy has been checked out and the term for the book loan is also mentioned. For every reader, five days before the term, the library system sends an automatic notification about the due date and at the same time an automatic notification also is sent to the immediate reader in the book request queue whether they are still interested in that book copy. There is a retention time of 24 hours to decide. If the next interested reader is still interested, that reader is asked to fill in the book request form and is given a list of dates to pick-up the requested book copy.

Once the reader returns the book copy to the librarian, they check for the due date of the book loan and evaluates the condition of the book. If there is any damage or torn pages, the librarian calculates and imposes the fine to the reader. When the reader pays the fine, the librarian closes the fine. If the book copy is badly damaged, the librarian updates the statement of utilization, and sends the book copy to the utilizer."

Based on this description, the Table 1 lists identified functional features while Fig. 3 shows the corresponding To-Be Topological Functioning Model.

Table 1. The functional features of the To-Be Library System.

Id	Functional feature	Action	Result	Object	Precondition	Executor	Sub.*
1	Opening a library website	Open		Person		Person	I
2	Clicking on the registration page	Click	Registration page	Registration page	The person is not registered	Person	I
3	Filling in the registration form	Fill in	Registration form	Registration	The person is not registered	Person	I
4	Checking the notification checkbox	Checked	Notifications	Registration form	The person has not checked during registration	Person	I
5	Creating reader account	Create	Reader account	Reader account	Registered successfully	Library System	I
6	Creating reader ID for the reader	Create	Reader ID	Reader ID	The reader account is created	Library System	I
7	Activating reader account	Activate	Reader account	Reader account	Registration is successful and received a reader ID	Library System	I
8	Logging in with reader credentials	Login	Reader account	Reader account	The reader account is activated	Reader	I

Id	Functional feature	Action	Result	Object	Precondition	Executor	Sub.*
9	Browsing categories	Browse	Categories of books	Book copy	The reader logs in to their account	Reader	I
10	Selecting interested book copy	Select	Book copy	Book copy	The reader is interested in a book copy	Reader	I
11	Checking for the book copy availability	Check	Book	Book request form	The reader is interested in loaning the book copy	Reader	I
12	Choosing alert notification	Choose	Notification	Alert Notification	The book copy is not available	Book Request Queue	I
13	Checking reader account for pending fines	Check	Pending fine	Pending fine	Before filling in book request form	Library System	I
14	Filling in the book request form	Fill in	Book	Book request form	The book is available	Reader	I
15	Scheduling a pick-up time	Schedule	Book copy	Book copy	The reader submitted book request form	Reader	I
16	Arriving at the library	Arrive	Reader	Reader	The date and time to pick up the book copy are booked	Reader	E
17	Asking for the reader ID	Ask	Reader ID	Reader ID	The reader arrives at the selected time	Librarian	I
18	Taking selected book copy from the book fund	Take	Book copy	Book copy	The reader ID is checked	Librarian	I
19	Checking out the book copy to the reader	Check	Book copy	Book copy	The book copy is taken	Librarian	I
20	Updating status in the library system	Update	Status	Status	The book copy is given to the reader	Librarian	I
21	Sending notification to the reader about due date	Notify	Due date	Book loan	The end date to book loan is in five days	Librarian	I

Id	Functional feature	Action	Result	Object	Precondition	Executor	Sub.*
22	Sending notification to the next interested reader in the book request queue	Notify	Reader account	Reader account	Another reader has requested for the same book	Book Request Queue	I
23	Returning the book copy	Return	Book Copy	Book copy	The book's loan term is exceeded	Reader	E
24	Taking back the book copy	Take	Book copy	Book copy	After reader returned book copy	Librarian	I
25	Evaluating the condition of the book copy	Evaluate	Book copy	Book copy	The reader returned the book copy	Librarian	I
26	Calculating fine	Calculate	Fine	Fine	The book is lost, there is any damage or torn pages, or exceeded the due date of the book loan	Librarian	I
27	Imposing fine to the reader	Impose	Fine	Fine	The fine is calculated	Librarian	I
28	Paying fine	Pay	Fine	Fine	The fine is imposed	Reader	E
29	Closing the fine	Close	Fine	Fine	The fine is paid	Library System	I
30	Returning the book copy to the book fund	Return	Book copy	Book copy	The fine is paid	Librarian	I
31	Updating the statement of utilization in the library system	Update	Statement	Statement	The fine is paid	Librarian	I
32	Sending the book copy to the utilizer	Send	Book	Book	The statement of use is completed	Librarian	I

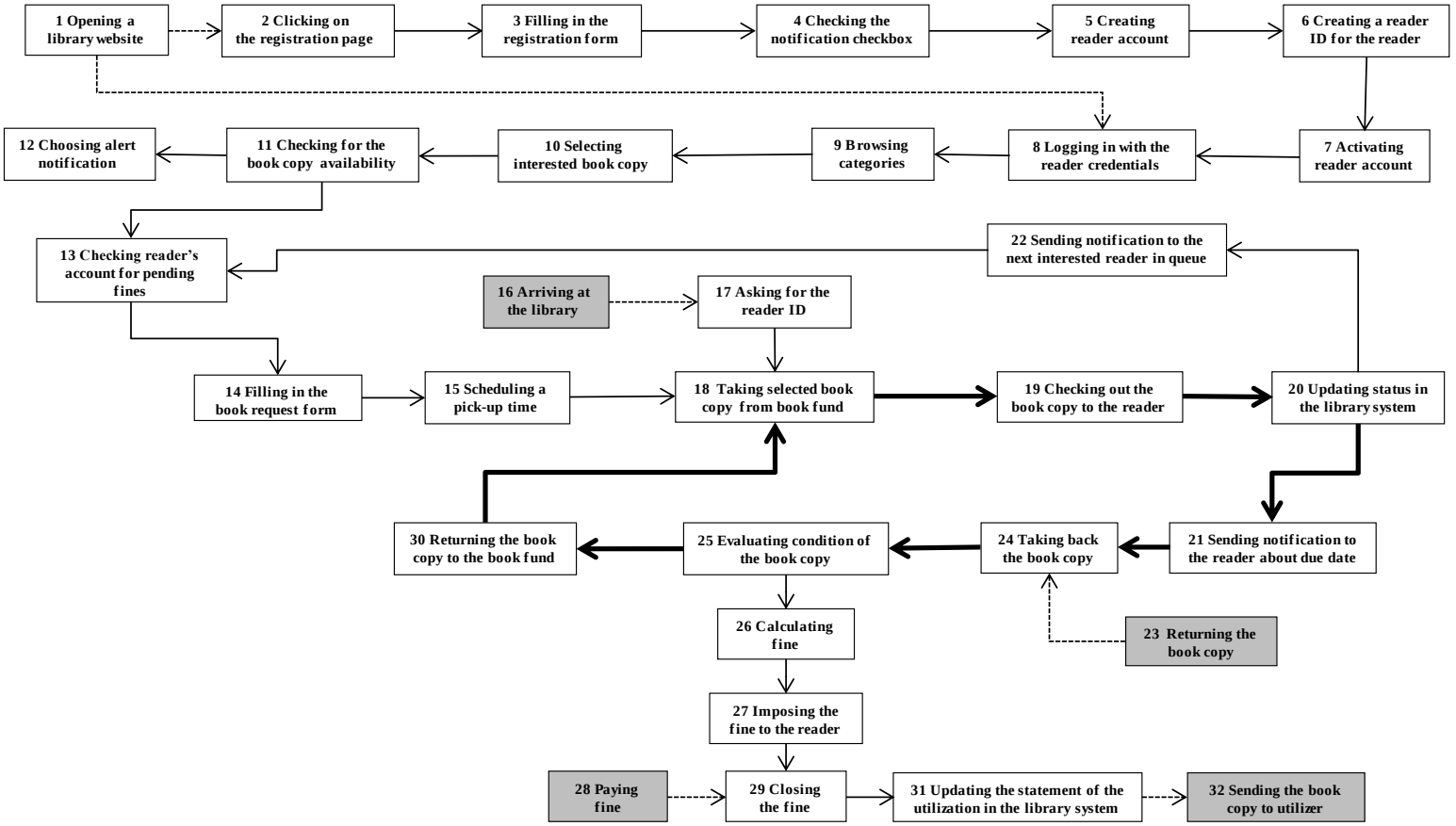

Figure 3. The To-Be Topological Functioning Model of the Library System.
Nodes represent functional features while arrows represent cause-and-effect relations.

Functional features in TFM specify functionality that exists in the problem domain, but functional requirements of software specify the functionality that must be implemented in the solution. (Nazaruka and Osis, 2018; Osis and Donins, 2017, Section 6.5)

The functional requirements which specify the functionality that should be implemented in the library's IT solution are:

FR1 – The system should perform registration of a new reader.

FR2 – The system should create a new reader ID for the registered reader.

FR3 – The system should handle all the requests by the reader while browsing website.

FR4 – The system should send alert notifications, notifications about book loan terms, information about new arrivals and other featured events to the reader.

FR5 – The system should allow to check out the book copy.

FR6 – The system should allow to impose fine to the reader and handle payments.

FR7 – The system should allow to check reader account for pending fines.

For each functional requirement Fig. 4 shows the mappings to corresponding functional features of the problem domain. Mappings are described with arrow predicates and enable validation of the completeness of functional requirements and their conformance with the constructed TFM.

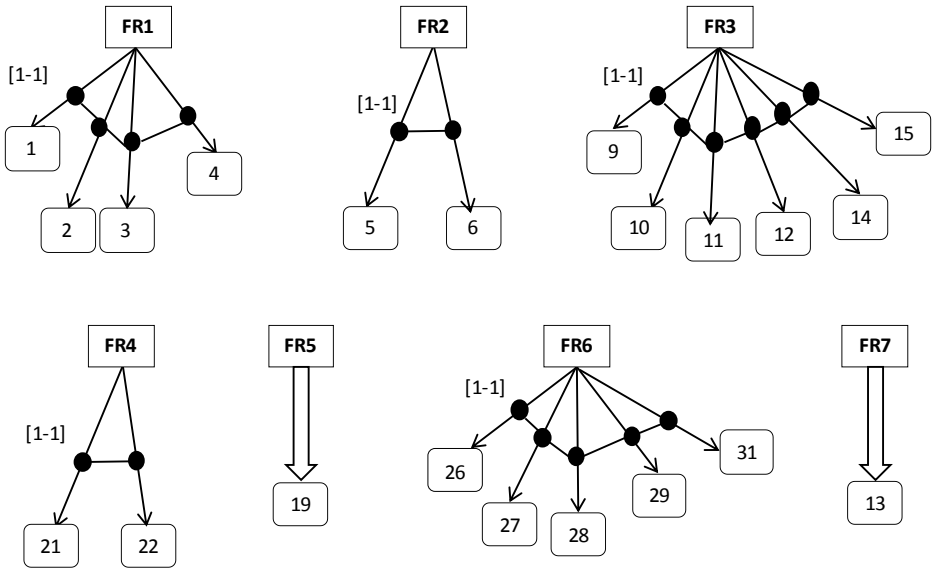

Figure 4. The mappings from functional requirements to functional features. Functional requirements FR1-FR4 and FR6 have one to many mappings, while FR5 and FR7 have one to one mapping.

4.3. Transformation from topological functioning model to event-driven solution

Table 2 summarizes proposed mappings as introduced in (Deharam and Alksnis, 2022) between elements in TFM and elements in event-driven solution. A functional feature in TFM forms an event in an event-driven solution. It is because the event, which is

referenced by the application, triggers components and events in other applications. The functional feature also triggers related functional features in TFM. The name of the functional feature forms the name of the event. The event description is the most important element as it represents the main reason (the cause) that provides necessary condition to trigger the event. The event description is formed by a physical or business functional feature specification in TFM.

Table 2. Mappings between elements of event-driven solution and TFM (Deharam and Alksnis, 2022).

TFM element	**Event-driven solution element**
Functional feature	Event
Name of the functional feature	Event name
A physical or business functional feature specification	Event description
An executor of the functional feature	Application
A cause functional feature in cause-and-effect relation where functional features with different executors take part.	Publish event (producer)
An effect functional feature in cause-and-effect relation where functional features with different executors take part.	Subscribe event (consumer)
An object or a result of a functional feature	Schema
Cause-and-effect relation between functional features	Event flow between applications
A topological space where functional features are logically joined	Logical event mesh
An indicator to which functional feature an object belongs	Event topic address

Other mappings can be substantiated with the following arguments. An application in the event-driven solution comes from an executor in functional feature. The executor in TFM performs (executes) the necessary action on the object. The application in the event-driven solution contains required events which trigger the required action to bring the necessary changes in the system. The cause-and-effect relation in TFM forms publish and subscribe event where one application subscribes or publishes events.

Event flow forms a relationship between the functional features. Logical event mesh is an event mesh where all the events are joined logically and are associated in an event-driven solution. A topological space where all the functional features in TFM are logically joined corresponds to the logical event mesh in the event-driven solution.

Event schema comes from an object of a functional feature performing the action. Event schema is a specification inside the object which flows internally to trigger the event. The event schema can be written in plain text, JSON, or binary. For example, after selecting a book copy, the reader checks for the availability and if the interested book copy is available, to loan the book copy, the reader needs to fill in a book request form. The corresponding functional features of this scenario are shown in Fig. 5 while the payload JSON schema with objects and properties involving for the book request form is listed in Fig. 6.

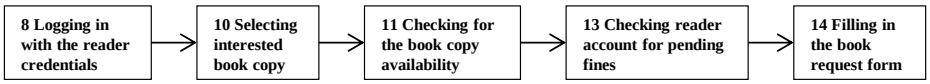

Figure 5. The functional features of TFM involved in the book request schema.

```
{
  "definitions": {},
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://example.com/root.json",
  "type": "object",
  "title": "The Book Request Schema",
   "properties": {
    "Login to reader account": {
      "$id": "#/properties/Book request",
      "type": "string",
      "title": "The Book Request Schema",
      "default": "",
      "examples": [
        "xyz"
      ],
      "pattern": "User name & Password"
    },
     "Select book copy": {
      "$id": "#/properties/Book request",
      "type": "string",
      "title": "The Book Request Schema",
      "default": "",
      "examples": [
        "xyz"
      ],
      "pattern": "Think & Grow Rich"
    },
     "Check for available book copies": {
      "$id": "#/properties/Book request",
      "type": "integer",
      "title": "The Book Request Schema",
      "default": "",
      "examples": [
        "xyz"
      ],
      "pattern": "4"
    },
  "pattern": "Think & Grow Rich"
    },
     "Check reader account for pending fine": {
      "$id": "#/properties/Book request",
      "type": "boolean",
      "title": "The Book Request Schema",
      "default": "",
      "examples": [
        "x"
      ],
      "pattern": "No"
    },
```

```
    "Fill in book request form": {
      "$id": "#/properties/Reader",
      "type": "string",
      "title": "The Book Request Schema",
      "default": "",
      "examples": [
        "Reader info"
      ],
      "pattern": "ID & Book copy"
    }
  }
}
```

Figure 6. The payload schema related to the book request form.

5. Validation of transformations

After applying the transformations to TFM shown in Fig. 3, the resulting event-driven solution with applications, events, and event flows is shown in Fig. 7. In the Solace PubSub+ Platform larger circled nodes represent applications, while smaller green nodes represent events. Event path shows the direction of event flows from one application to another and are shown as directed arcs.

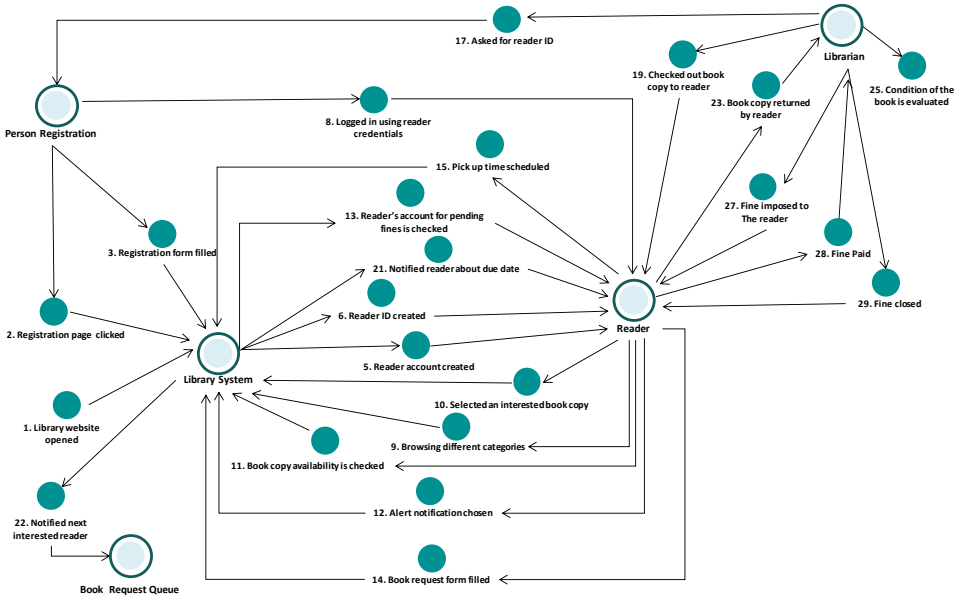

Figure 7. Event-driven solution with applications, events, and their interactions for the library system.

Fig. 8 shows the fragment of TFM how the reader, after logging in to their account, selects an interested book copy. After selecting an interested book copy, the reader

checks for the availability of book copy in the book fund. Once it is available, reader fills in the book request form available online.

In Fig. 8 the mapping A traces to the executor of the functional feature, the mapping B refers to the event in the event-driven solution tracing to the functional feature. Finally, the mapping C represents the flow of the events in the event-driven solution tracing to the cause-and-effect relation between two functional features.

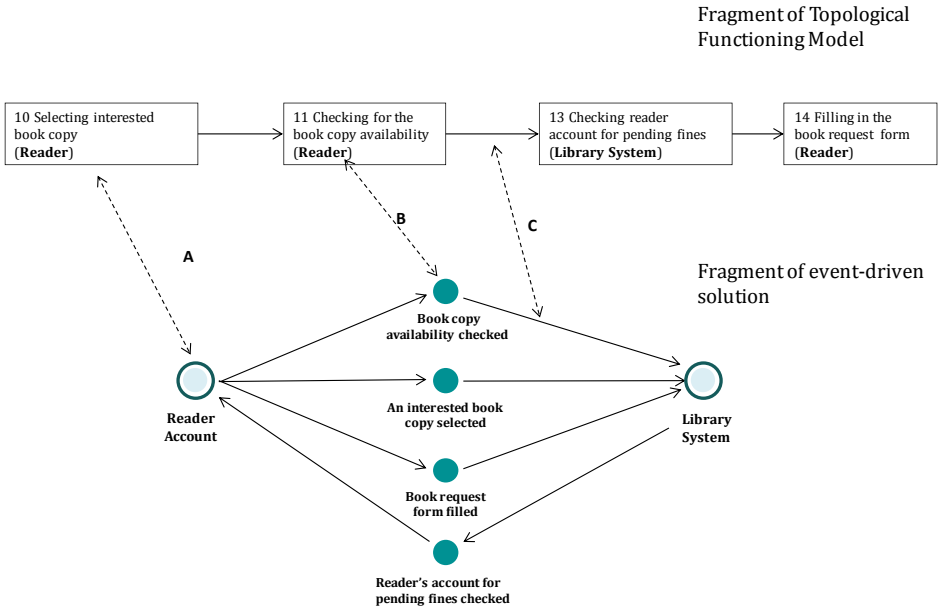

Figure 8. Mappings from the TFM elements to event-driven solution elements. In the functional features the executors are shown in bold.

As we chose the Solace PubSub+ Platform to demonstrate event-driven solution design, the Fig. 9 shows a fragment of events in the library system event-driven solution as seen in the PubSub+. In Fig. 9, the name column refers to the name of the event and shared columns refers if the event is shared across other applications in the application domain.

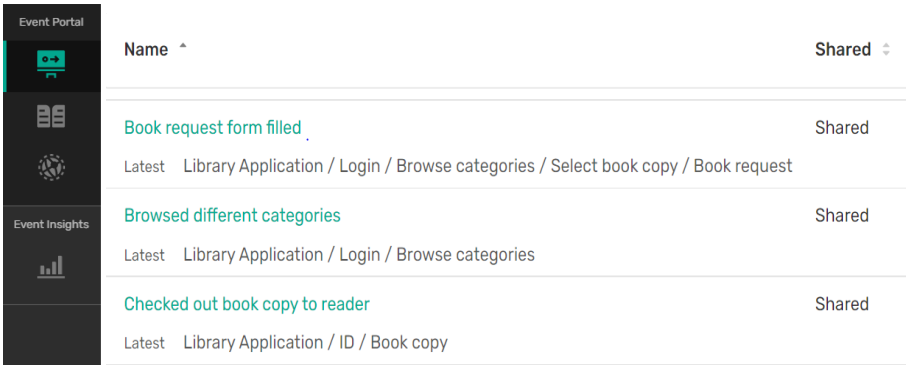

Figure 9. Fragment of list of events in the library system application domain.

Fig. 10 shows corresponding payload schemas used in the library system's event-driven solution. In Fig. 10, the name column refers to the name of the schema, schema type is JSON Schema, and the content type is JSON.

Fig. 11 shows the details of the book request form where the reader, after selecting an interested book copy, fills in the book request form (see Fig. 6).

Figure 10. Fragment of list of schemas in the library system application domain.

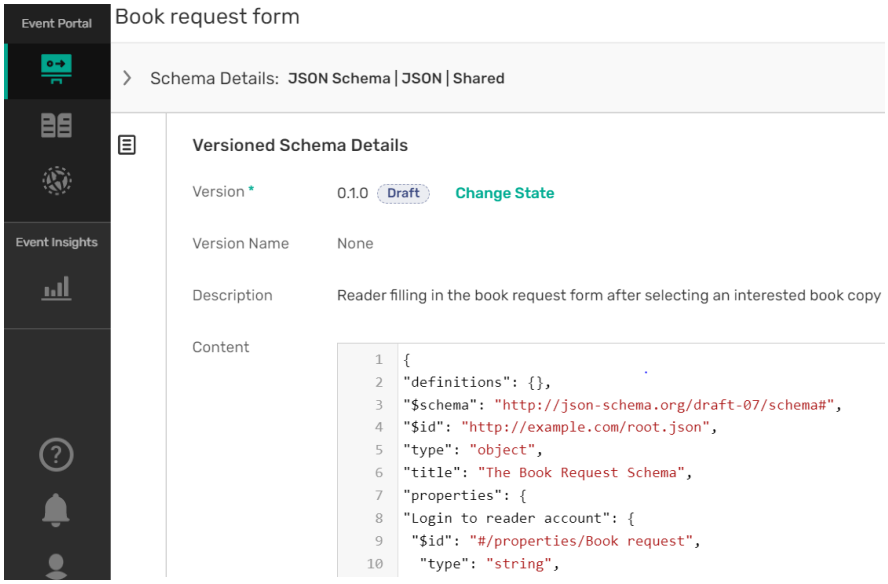

Figure 11. Details of the book request form schema in the library system application domain.

6. Related works

There are various approaches proposed in the context of event modelling and event-driven IT systems (e.g., Stopford, 2018) However, the goal of our research and thus the scope is to contribute to this knowledge area from a new perspective by investigating suitability of TFM in obtaining event-driven solution from the problem domain. Therefore, the related work to consider must involve the use of TFM as a source or target model for mapping, traceability or transformation into/from other abstractions or models. Due to a formal nature of TFM and ability to holistically represent a system both

from topological and functional aspect, there has been several research performed which involve the use of TFM as a source or destination to other models. There are two related works to mention matching these criteria.

In (Donins et al., 2012) the UML state transitions are analyzed by the functional characteristics of TFM and mappings from TFM elements to the UML state machine elements are proposed. In their approach the state of an object used in functional feature is mapped to the UML state element while the cause-and-effect relation is mapped to the transition of the state machine. Action of the functional feature is mapped to the event, entry or exit action of the state machine. Pre- and postconditions are mapped to guard conditions. Finally, logical relationship between functional features is mapped to fork and join element of the state machine. Later this has been integrated in Topological UML (Osis and Donins, 2017).

In (Nazaruka et al., 2016) the authors verify the functional completeness of a BPMN model. To do this, the BPMN model is transformed to TFM for further analysis. An example is taken and mappings from BPMN elements to TFM elements are introduced, and the completeness of the BPMN model is discussed. The mapping of BPMN elements to other formal notations is not new concept and has been mapped to such formal notations like Petri nets, Prolog, Communicating sequential processes, etc. It is concluded that those mappings were limited to deadlocks, thread correctness and data flows. By using TFM, the system's functionality and completeness can be determined holistically. They explain how the functional feature is executed after a trigger. If the trigger is successful, then its termination leads to a successful trigger of the effect functional feature. In contrast to BPMN, TFM contains just a few but fundamental modeling constructs (Nazaruka et. al., 2016). BPMN elements are elicited and TFM elements are identified, and mappings between TFM elements and BPMN elements are listed. For example, tasks, events, and data come from a functional feature tuple. All elements which have corresponding notion in TFM are mapped, and those which don't have them are ignored. For example, elements like text annotation and conversation link doesn't have corresponding mapping element in TFM. Results of transformation are observed for verification whether the elements follow TFM properties as it is checked for the presence of topological and functional properties. TFM is considered valid if it has no isolated vertices and has a functioning cycle in addition to inputs and outputs, continuous mapping, and all cause-and-effect relations are necessary and sufficient to trigger subsequent functional features (Osis and Asnina, 2010). The functioning cycle must contain functional aspects with respect to problem domain. All transformations were done manually, and functioning cycle identification were considered the hardest part as it was not evident how to identify and when to use output and input events.

7. Conclusions

As many organizations are considering event-driven solutions to support their digitalization, the article provided contribution with and approach that facilitates solution's conformance with the problem domain.

TFM was chosen as it can represent the functioning of the problem domain (e.g., an enterprise as a system) holistically by emphasizing computation independent viewpoint. TFM allows to capture and emphasize cause-and-effect relations between functional characteristics of the problem domain.

To propose an event-driven solution, the case study analysis research method was conducted for a library problem domain. Functional features were identified, and cause-and-effect relations were identified. TFM was constructed and the event-driven solution was designed in the Solace PubSub+ Platform.

Proposed mappings between TFM elements and event-driven solution elements demonstrate that they can contribute to obtaining an event-driven solution which conforms with the problem domain. The scope for future research includes development of automated transformations of proposed mappings and to investigate the possibilities of this approach in iterative development processes.

References

Deharam, S.T., Alksnis, G. (2022). Using Topological Functioning Model to Support Event-Driven Solutions, in Ivanovic et al. (Eds.), *Proceedings of the 15th International Baltic Conference*, Baltic DB&IS 2022, CCIS 1598, Springer.

Doniņš, U., Osis, J., Nazaruka, E., Jansone, A. (2012). Using Functional Characteristics to Analyze State Changes of Objects, in *Databases and Information Systems*, pp. 94-106.

Narkhede, N. (2018). *SpringOne Platform. Event Driven Systems*, VMware.

Nazaruka, E., Ovchinnikova, V., Alksnis, G., Sukovskis, U. (2016). Verification of BPMN Model Functional Completeness by using the Topological Functioning Model, in *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering*, SciTePress.

Nazaruka, Ē., Osis, J. (2018). The Topological Functioning Model as a Reference Model for Software Functional and Non-functional Requirements, in: *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering*, SciTePress.

Osis, J. (2003). Extension of Software Development Process for Mechatronic and Embedded Systems, in *Proceedings of the 32nd International Conference on Computers and Industrial Engineering*, pp. 305-310.

Osis, J., Asnina, E. (2010). Topological modeling for model-driven domain analysis and software development: Functions and Architectures, in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, pp. 15-39. Hershey, New York: IGI Global.

Osis, J., Asnina, E., Grave, A. (2008). Computation Independent Representation of the Problem Domain in MDA, in *e-Informatica Software Engineering Journal*, Volume 2, Issue 1.

Osis, J., Donins, U. (2017). *Topological UML Modeling: An Improved Approach for Do-main Modeling and Software Development*, Elsevier.

Platt, R., Thompson, N. (2015). The evolution of UML, in *Encyclopedia of Information Science and Technology*, 3rd ed., IGI Global.

Solace PubSub+ Platform (2022). https://docs.solace.com/Solace-PubSub-Platform.htm, last accessed 29 December 2022.

Stopford, B. (2018). *Designing Event-Driven Systems*, O'Reilly.

Tiempo Development. (2020). *A Business Leaders Guide to Event-Driven Architecture*, https://vdocuments.net/a-business-leaders-guide-to-event-driven-architecture-event-driven-architecture.html, last accessed 29 December 2022.

Received October 31, 2022, revised January 21, 2023, accepted January 27, 2023