


RESEARCH ARTICLE

Integration Flows Modeling in the Context of Architectural Views

TOMASZ GÓRSKI , (Senior Member, IEEE)

Institute of Computer Science, University of Gdańsk, 80-308 Gdańsk, Poland

e-mail: tomasz.gorski@ug.edu.pl


ABSTRACT In an increasing number of software applications, the execution of their functions depends on communication with external systems. Messaging enables the integration of information technology (IT) systems in a loosely-coupled manner. The paper aims to show the *Integrated services* architectural view and two methods of modeling messaging flows at the service and business levels. Service flows allow for modeling the communication between systems/services at the level of sending messages. In contrast, business flows model the entire course of interaction among cooperating applications. The methods employ an Integration flows diagram, which is a specialized version of the Unified Modeling Language (UML) Activity diagram. In addition, the business process task from the *Integrated Processes* view defines the context for the business flow. Moreover, the *Use Cases* view identifies the integration scope within requirements. All those views belong to the *I+5* architectural views model. The paper exerts extensibility mechanisms declared in the *UML Profile for Messaging Patterns*. Besides, the profile has been augmented with stereotypes for messaging patterns for the Apache Camel framework, stakeholders involved in communication, and up-to-date stereotypes for structural components. The methods were used in the integration design of sending orders and confirmations between the business applications of the brokerage house and the stock exchange.

INDEX TERMS Interoperability, messaging, *I+5* architectural views model, unified modeling language extensibility mechanisms, service-oriented architecture, microservices.

I. INTRODUCTION

Messaging is an integration style for cooperating business applications, which might be designed in distinct architectures and written in various programming languages. Enterprise service buses (ESB) and message queues are communication components that ensure infrastructure for delivering messages while providing transformation and routing services. Kannisto et al. [1] felicitously describe the differences between those two components. A message queue declares a communication protocol and defines message formats. In consequence, cooperating IT systems have to send messages in a standardized fashion. As a result, it enforces adding specific adapters for diverse systems. Enterprise service buses typically come with adapters that support a variety of protocols and data formats. Messaging integration style can be realized in both architectures:

Service-Oriented Architecture (SOA) and microservices. The architectures offer flexible integration and service reusability due to their modular composition. An in-depth analysis of the role of interfaces in service-oriented systems was done by Bhadoria et al. [2]. Standard ways of messaging were described by Hohpe and Wolf [3] and called Enterprise Integration Patterns (EIPs). In the recent review, Aziz et al. [4] identified the following directions of messaging development: microservices, cloud environments, and the Internet of Things (IoT). The latest research concentrates on the Publish-Subscribe messaging pattern. In that area, Martinez et al. [5] introduce a new version of the pattern, which uses Kubernetes containers for microservices orchestration. Moreover, Zhong et al. [6] present the pattern's specialization in which subscribers receive paired objects instead of separate ones. Furthermore, Livaja et al. [7] propose an approach that couples incoming geospatial objects with the appropriate subscriptions. While Daraghmi et al. [8] discuss modification of the Saga pattern for distributed transactions

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Ali Babar .

synchronization among microservices. The topic is still relevant and developing.

The ability to model messaging patterns in a commonly understood manner is of crucial importance. A software architecture description is commonly realized using Unified Modeling Language. Ozkaya et al. [9] surveyed specialists on the usage of UML diagrams. The results show that the UML Activity diagram is commonly adopted for data flow modeling (65% of inquired professionals). In this paper, the author enhances the UML Activity diagram and adopts it for modeling integration flows. Integration flows are constructed using messaging patterns. Therefore, it is important to create and maintain a set of reusable model elements representing individual messaging patterns. Thanks to this, it is possible to model integration flows in a clearly understood way. The Unified Modeling Language is designed to be conceptually simple. However, its semantics can be extended. Extensibility mechanisms can be grouped into the following categories: stereotypes, tagged values, and constraints. Moreover, UML offers profiles as a means for grouping new semantic constructs [10]. Two profiles have recently been proposed in the area of interoperability. Firstly, Petrasch et al. [11] introduced stereotypes for data integration patterns. The patterns and corresponding UML stereotypes concentrate on data integration tasks modeling. Secondly, Górski [12] showed the profile that comprises stereotypes for messaging patterns. The profile consists of stereotypes declared for the Enterprise Integration Patterns, ZeroMQ open-source messaging framework, and academic patterns resulting from recent research work. Designing message flows requires an architectural description that takes into account the aspect of communication between systems. The *I+5* model contains *Integrated services* architectural view dedicated to the modeling of integration flows [13].

The contribution encompasses modeling methods of two types of messaging flows. These are service and business flows. The paper concentrates on the following architectural views: *Integrated services*, *Use cases*, and *Integrated processes*. It should be underlined that the first view is strictly devoted to communication modeling among IT systems. The paper shows the usage of *Integration flows diagram*, which is an enhanced version of the UML activity diagram. Partitions were employed to set responsibilities for actions in message flows. Modeling of message flows allows for service identification. UML Component diagrams were used for presenting interacting components with identified services. Besides, it was also shown how the integration context is defined in the area of the business process that is modeled in the *Integrated processes* view. The author also introduces UML stereotypes for structural components, messaging patterns for the Apache Camel framework, and external systems. The latter was used in the *Use cases* view for emphasizing communication between collaborating systems.

Figure 1 depicts architectural views involved in the design of message flows. The business process model placed

in *Integrated processes* view identifies the business task, which requires communication between cooperating systems. It results in modeling the business flow in the *Integrated services* view. In addition, the *Use Cases* view identifies the IT system functions involved in the integration.

The remaining part of the paper has the following structure. Section II presents related work. Section III introduces UML stereotypes for cooperating components and messaging patterns. Section IV presents modeling approaches for service and business flows on the example of integration design between the brokerage house application and the stock exchange system. Section V contains discussion and perceived limitations. Section VI summarizes the paper and sets directions for further research.

II. RELATED WORK

The subject of describing the architecture of information systems has been developed for many years. However, the aspect of communication between systems is not so often considered. Recently, Kirpitsas et al. [14] underlined the importance of Kruchten's *4+1* architectural views model [15]. However, the authors pointed to the architectural views model *I+5* as an example of the significant progress that has been made in the description of software architecture [13]. Similarly, Suljkanovi et al. [16] emphasized that it is the *I+5* model that consists of views that allow for a diverse architectural description. The model comprises two views that are crucial for modeling message flows: *Integrated services* and *Integrated processes*. As far as communication is concerned, Zhang and Kianfar [17] in the design of Vehicle-to-Infrastructure communication system used *Physical* and *Communications* views from the Architecture Reference for Cooperative and Intelligent Transportation [18]. However, they refer to the *I+5* model while discussing the importance of interoperability design between various subsystems. As far as healthcare is concerned, Marbough et al. [19] in building a regulatory framework for mHealth put emphasis on designing the exchange of information between stakeholders. But, they also employ business process modeling. The importance of business process modeling in the process of designing IT systems still grows. Lately, González Moyano et al. [20] described the uses of business process modeling in the currently most proliferated agile software development projects. In the model, the *Integrated processes* view has been singled out for modeling business processes that involve cooperating organizations/companies. Generally, process-oriented organizations may benefit from implementing an integration platform with an enterprise service bus or message queue. There are usually many complex business processes in such companies that span various systems. Thus, they exploit services from multiple systems. Górski and Woźniak [21] show that SOA and microservices may help in optimizing the execution of business processes into three areas: Resource Allocation, Service Composition, and Service Scheduling. The most popular are genetic algorithms that are used to

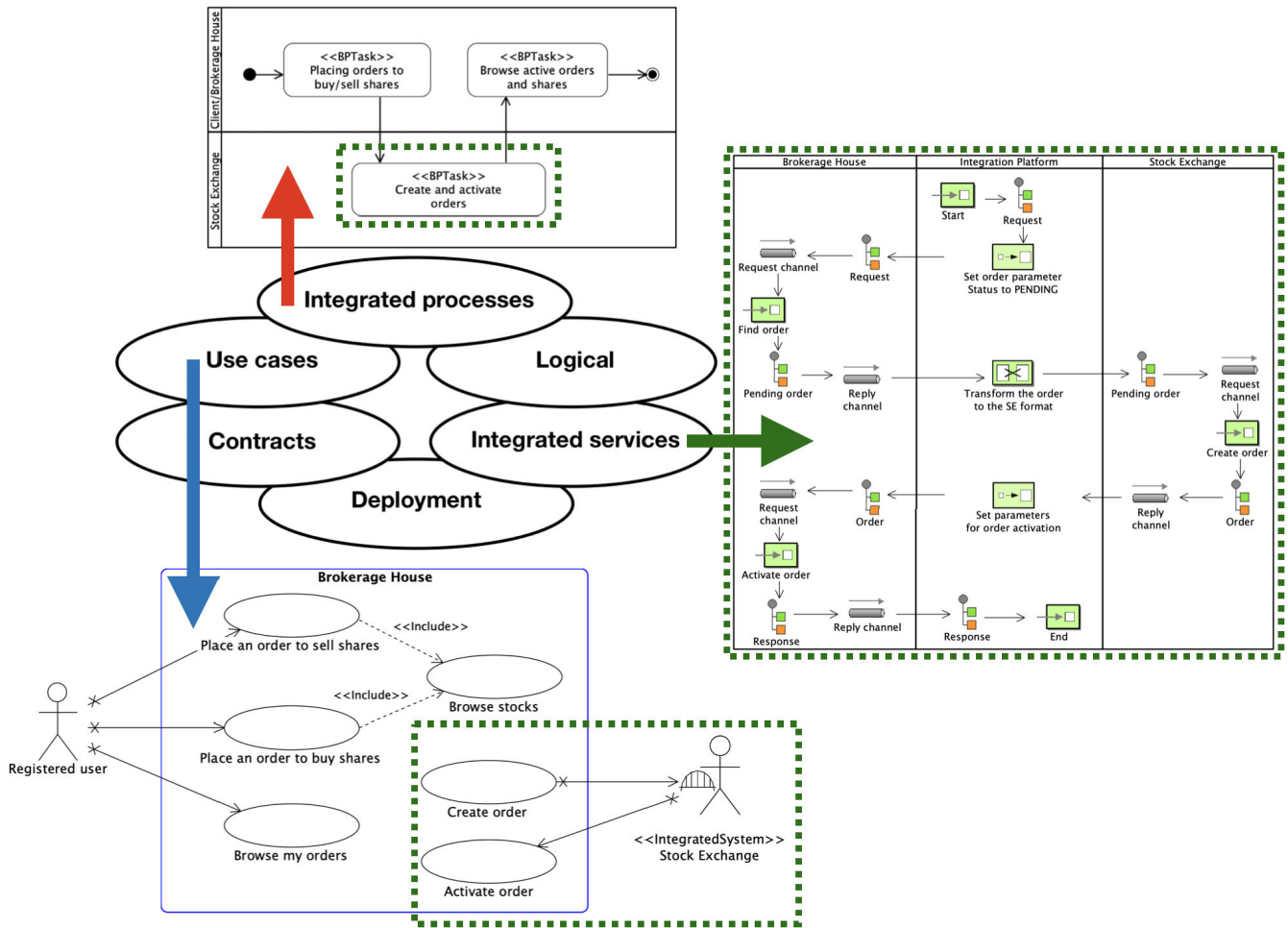


FIGURE 1. Integrated services view with integration flow in the context of the Integrated processes and Use cases architectural views.

optimize business processes during run-time. Further advances in adaptive service composition that fulfill user requirements were shown by Wang et al. [22]. Another important aspect, the automated discovery of cross-organizational collaborative business processes was shown by Montarnal et al. [23]. The business justification of the designed functions or messaging flows is crucial for the implementation of the system so as to realize actual user needs. So, message flows should be placed in the context of a business process to clearly indicate business tasks that require integration with external IT systems. Therefore, message flows are also called integration flows. Both terms will be used interchangeably hereafter.

In the I+5 model, the Integrated services view is dedicated to modeling the communication among cooperating parties [13]. Therefore, the article focuses on the presentation of this architectural view. Additionally, the integration context determines the Use cases view. In this view, the <<IntegratedSystem>> stereotype has been declared for the external system being integrated. Integration flows enable communication between systems using various data formats

or communication protocols. Researchers describe the architecture of their solutions using various architectural views and Unified Modeling Language. For example, Akhilesh et al. [24] present the design of the automated penetration testing framework for IoT devices using: UML Deployment diagram in the Deployment view, UML Component diagram in the Logical view, and the UML Activity diagram in the Use cases view. But, the Deployment view is the most frequently used. Besides, authors apply their own UML stereotypes. Han et al. [25] show the design of an access control framework using Software Guard Extensions (SGX). They do not formally construct a new UML profile but propose specific stereotypes for nodes and services, e.g.: <<SGXServerNode>>, and <<distributedStorageService>>. It should be noted that the I+5 model has been recommended by Ahmed et al. [26] to describe the software architecture and system integration. They proposed an incentive trust model based on blockchain with a privacy-preserving threshold ring signature scheme for vehicular ad hoc networks (VANETs). They focused on Logical and Deployment views to design the information exchange between systems. For architectural description

they have used four UML diagram types: class, sequence, communication, and deployment. They have also proposed specific UML stereotypes for VANETs, e.g.: `<<RSUNode>>` for a roadside unit and `<<VehicleNode>>` for a vehicle node. Researchers introduce stereotypes as semantically new modeling elements and construct new UML profiles. For example, Thramboulidis and Christoulakis [27] present a UML profile that facilitates the transformation of interfaces from the Internet of Things (IoT) to the Representational State Transfer Application Programming Interface. Besides, Marouane et al. [28] introduce the profile with concepts related to real-time databases and integrate Object Constraint Language (OCL) to enforce the variation points consistency. As was mentioned in the section I, Petrasch and Petrasch [11] presented Data Integration Patterns for the interoperability of IT systems. However, the proposed patterns concentrate on the data, and UML was applied to model data integration. Besides, Plazas et al. [29] have proposed the STS4IoT UML profile. The profile simplifies the definition of the Internet of Things applications and their integration into other information systems, such as stream data warehouses. As can be seen, UML profiles find various uses. But there is one UML profile for modeling messaging patterns [12]. That profile has been augmented with specific stereotypes for structural components, messaging patterns for the Apache Camel framework, and external systems. The paper concentrates on the communication between IT systems. Thus the emphasis was put on the design of integration flows within the dedicated *Integrated services* architectural view.

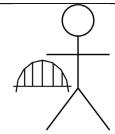
III. STEREOTYPES

The profile uses UML stereotypes for providing semantically new modeling elements. The main part of the profile constitutes a set of messaging patterns. Introducing a new modeling element requires indicating the standard UML model element type that is extended. The basic UML classifier for messaging patterns is UML Action. Besides, the UML Component base type has been applied for denoting communication components. There is also one stereotype, which stands for an external system. For that stereotype, the UML Actor base type has been chosen. The profile was prepared in the commonly used Visual Paradigm tool. The file *UMLProfile4MessagingPatterns.vpp* of the profile is exposed in the GitHub repository [30]. The profile declares the corresponding stereotype for each of included messaging patterns.

A. STAKEHOLDERS INVOLVED

For the description of integration, it is important to be able to identify the parties involved and the communication components. In this context, new stereotypes were proposed for model elements in two architectural views *Use cases* and *Integrated services*. In the *Use Cases* view, a stereotype is needed to distinguish the external IT system with which cooperation is to be designed using the integration platform. The stereotype `<<IntegratedSystem>>` has been previously declared [13]. However, the standard UML actor icon

TABLE 1. The stereotype in the Use cases view.

Stereotype	Base type	Icon
<code><<IntegratedSystem>></code>	UML Actor	

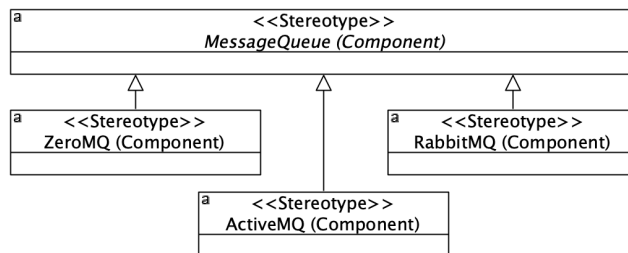


FIGURE 2. UML Profile diagram with stereotypes inheritance hierarchy for message queues.

remained the same. From the legibility point of view of the UML Use case diagram, it is important to highlight the actor representing the integrated external system. Therefore, a new icon for this stereotype has been proposed.

Table 1 show `<<IntegratedSystem>>` stereotype details and its icon.

Communication component types reside in the *Integrated services* view. The modeling method employs two types of communication components: message queues and enterprise services buses. Stereotypes for those both types of components have been included in the profile. Stereotypes for communication components use the UML Component element type as a base UML classifier. The abstract `<<MessageQueue>>` stereotype was declared for the generic message queue. Three concrete ones denote message queue environments: `<<ZeroMQ>>`, `<<RabbitMQ>>`, and `<<ActiveMQ>>`.

Figure 2 presents an inheritance tree of stereotypes for message queues.

Besides, the abstract `<<EnterpriseServiceBus>>` stereotype was declared for generic ESB. Eight concrete stereotypes are specializations of that abstract one and stand for factual environments, e.g., `<<OracleServiceBus>>`, `<<MuleESB>>`, `<<RedHatFuse>>`, `<<TalendESB>>`, and `<<WSO2EnterpriseServiceBus>>`.

Table 2 presents stereotypes for actual ESB frameworks that are comprised in the profile.

Figure 3 presents the UML Profile diagram with the hierarchy of stereotypes for enterprise service buses. The definition of stereotypes also uses a tagged value to store the full name of the framework.

Especially in this area, the profile is open to expansion. This is important due to the fact that the availability and timeliness of individual integration platforms change quite frequently over time.

TABLE 2. Stereotypes for concrete Enterprise Service Buses.

Stereotype	Base type
<<WSO2EnterpriseServiceBus>>	UML Component
<<WorkdayBusinessProcessFramework>>	UML Component
<<OracleServiceBus>>	UML Component
<<TalendESB>>	UML Component
<<RedHatFuse>>	UML Component
<<MuleESB>>	UML Component
<<AzureServiceBus>>	UML Component
<<IBMDDataPowerGateway>>	UML Component

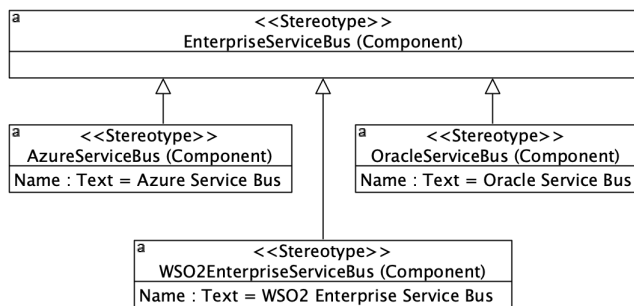


FIGURE 3. UML Profile diagram with selected stereotypes in inheritance hierarchy for enterprise service buses.

B. ENTERPRISE INTEGRATION PATTERNS

Messaging makes applications loosely coupled by communicating asynchronously. It makes communication more reliable because the two applications do not have to be running at the same time. Standard ways of messaging have been described by Hohpe and Wolf [3] and called Enterprise Integration Patterns.

The profile declares the corresponding stereotype for each of them. The stereotypes for selected Enterprise Integration Patterns (EIP) are described below:

- <<MessageChannel>> – the means of transmitting messages. The source system puts the message to the message channel and the target system gets the message.
- <<Message>> – the format for the data allowed for transmitting.
- <<PublishSubscribeChannel>> – one input channel splits into multiple output channels. When an event occurs, a copy of the message is delivered to each of the subscribers' channels.
- <<RequestReply>> – sends a pair of messages, one to the target application and the second back to the source application, in separate channels.
- <<MessageTranslator>> – translates one data format into another.
- <<ContentEnricher>> – enhances a message with additional data.
- <<Resequencer>> – collects and re-orders messages so that they can be published to the output channel in a specified order.

TABLE 3. Stereotypes for selected EIP messaging patterns.

Stereotype	Base type	Icon
<<MessageChannel>>	UML Action	
<<Message>>	UML Action	
<<PublishSubscribeChannel>>	UML Action	
<<RequestReply>>	UML Action	
<<MessageTranslator>>	UML Action	
<<ContentEnricher>>	UML Action	
<<ClaimCheck>>	UML Action	
<<MessageRouter>>	UML Action	
<<Normalizer>>	UML Action	
<<GuaranteedDelivery>>	UML Action	
<<DeadLetterChannel>>	UML Action	
<<MessageEndpoint>>	UML Action	

- <<MessageRouter>> – receives a Message from the Message Channel and sends it to one of the alternative channels due to conditions fulfillment.
- <<ContentFilter>> – removes unimportant data items from a message leaving only important ones.
- <<ClaimCheck>> – involves storing message data in a persistent store and passing proof to subsequent components, which may use that proof to retrieve the stored information.
- <<DeadLetterChannel>> – a place for storing messages that could not be delivered to intended receivers.
- <<GuaranteedDelivery>> – uses a built-in data store to persist messages.
- <<Normalizer>> – routes each message type through a custom Message Translator so that the resulting messages match a common format.
- <<MessageEndpoint>> – the entry/exit point of a messaging system used for sending/receiving messages.

Table 3 shows stereotypes and icons for selected Enterprise Integration Patterns.

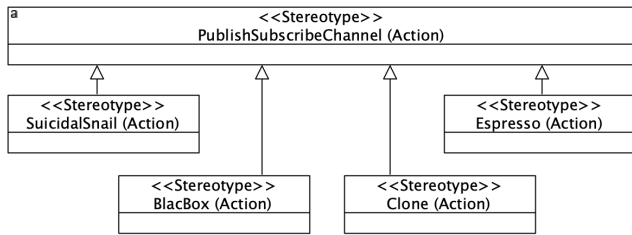


FIGURE 4. UML Profile diagram with ZeroMQ stereotypes inheritance hierarchy for Publish-Subscribe Channel specializations.

Almost all stereotypes have dedicated icons assigned. Those missing have not been provided for by the authors of the EIP, e.g.: Canonical Data Model, Scatter-Gather, and Format Indicator.

C. ZeroMQ FRAMEWORK PATTERNS

However, integration solutions are constantly evolving and new or improved mechanisms for solving existing message exchange problems are emerging. ZeroMQ framework brings a lot of freshness to this area. The framework defines specialized versions of Publish-Subscribe Channel [31] and Request-Reply [32] patterns. The original EIP icons for the Publish-Subscribe Channel and Request-Reply patterns have been modified and a black-red-white color schema has been adopted for icons proposed for stereotypes related to ZeroMQ framework patterns.

The following stereotypes have been introduced for ZeroMQ patterns that specialize Publish-Subscribe Channel:

- <<SuicidalSnail>> – detects slow subscribers in order to maintain defined maximum latency.
- <<Espresso>> – monitors the network and prints all incoming messages instantly.
- <<BlackBox>> – detects high-speed subscribers and uses multithreading for sending and reading messages in separate threads.
- <<Clone>> – maintains a common state among a group of clients using a key-value store.

Figure 4 presents an inheritance tree of stereotypes for ZeroMQ specializations of the Publish-Subscribe pattern.

Besides, the following stereotypes have been introduced for ZeroMQ patterns that specialize Request-Reply:

- <<LazyPirate>> – rather than blocking a receive, the pattern involves polling the socket to receive the reply.
- <<Majordomo>> – clients’ requests are augmented with service names. Besides, servers are asked to register for required services.
- <<BinaryStar>> – sets high-availability pair of servers in the primary-secondary configuration. Connections from client applications are accepted by the primary server. The secondary server is idle but monitors the primary one. The secondary server turns to the primary if detects the original primary server stops working.
- <<Titanic>> – ensures that messages are not lost by storing them in a message queue.

TABLE 4. Stereotypes for ZeroMQ specializations of Publish-Subscribe Channel and Request-Reply patterns.

Stereotype	Base type	Icon
<<SuicidalSnail>>	UML Action	
<<Espresso>>	UML Action	
<<BlackBox>>	UML Action	
<<Clone>>	UML Action	
<<LazyPirate>>	UML Action	
<<Majordomo>>	UML Action	
<<BinaryStar>>	UML Action	
<<Titanic>>	UML Action	
<<Freelance>>	UML Action	

- <<Freelance>> – configures a pool of name servers. The reliability is elevated at a higher level because if one server stops working, clients are connected to another server in the pool.

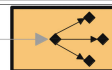


Table 4 shows selected stereotypes and icons for ZeroMQ specializations of the Publish-Subscribe Channel and Request-Reply patterns.

D. APACHE CAMEL FRAMEWORK PATTERNS

Apache Camel framework mainly implements Enterprise Integration Patterns. Virtually the full range of seventy-seven of these patterns is implemented in this environment. Each of these patterns is described in detail on the website of the framework [33]. When describing Camel’s own implementation of the pattern, the website explicitly provides information on the original EIP pattern. However, Apache Camel also extends its range of possibilities with additional own patterns. The following patterns have been identified as Apache Camel’s own designs because in descriptions there is no reference to EIP ones:

- <<LoadBalancer>> – allows for delegating messages to one of the endpoints using various load-balancing policies.
- <<ResumableConsumer>> – the resume strategy allows skipping reading and processing data that has already been consumed.

TABLE 5. Stereotypes for Apache Camel new messaging patterns.

Stereotype	Base type	Icon
<<LoadBalancer>>	UML Action	
<<ResumableConsumer>>	UML Action	
<<ChangeDataCapture>>	UML Action	

- <<ChangeDataCapture>> – allows tracking changes in databases and then lets applications listen to change events, and react accordingly.

Table 5 shows stereotypes and icons for Apache Camel patterns. An orange background has been adopted for icons related to the patterns for that framework.

IV. MODELING OF INTEGRATION FLOWS

The article deals with placing buy and sell orders for shares in a brokerage house (BH). Orders placed in the BH are transferred to the Stock Exchange (SE). This is where the communication between the systems of these two institutions comes in. The SE system manages buy/sell orders. There must be two converse sell and buy orders for the shares of the same entity for the sale and purchase transaction to be concluded in the SE system. Additionally, the buyer must consent to pay the price demanded by the seller.

Modeling of service and business integration flows are used in the architectural description of cooperating information technology systems. Both structural and dynamic sides of communication were shown in the *Integrated services* architectural view. UML Component diagrams were employed to model the structure of components taking part in message exchange: integration component and IT systems components. An integration component can be implemented as a message queue or enterprise service bus. Whereas, integration flows are modeled using Integration flows diagrams as a special form of UML Activity diagrams. These flows can represent a single message transfer. In such a case, it is the service flow.

For the sake of clarity in further considerations, the author introduces the following definition of the service flow (Theorem 1).

Theorem 1 (Service Flow): A service flow is a sequence of message processing steps that originate in the source system, run in the integration component, and reach the target system in order to invoke a specific service.

Moreover, integration flows are modeled in the context of the business process task. For the sake of clarity in further considerations, the author introduces the following definition of the business flow (Theorem 2).

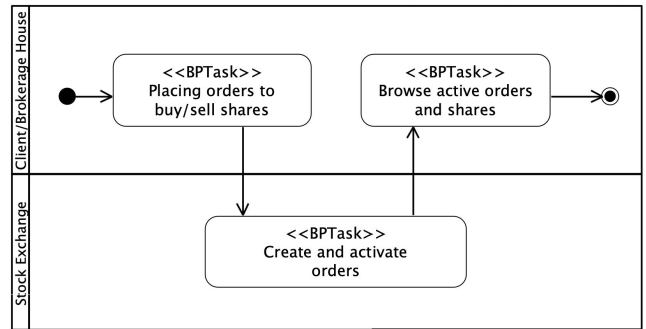


FIGURE 5. The business process of Placing orders to buy/sell shares.

Theorem 2 (Business flow):

A business flow is a sequence of service flows that are run on the integration platform to reach the goal of a business task.

In addition, the following steps can be distinguished in the modeling method of integration flows:

- business process modeling – it is done using UML Activity diagrams. Often business modeling is realized using the Business Process Model and Notation [34]. In the method, UML is employed to keep the same notation for business processes and system modeling.
- identification of business tasks that involve integration,
- identification of use cases that participate in integration,
- modeling all needed service flows and eliciting services,
- composition of business flow from service flows for the realization of a business task,
- modeling components with services.

A. INTEGRATED PROCESSES VIEW

Figure 5 presents the UML Activity diagram with the business process for placing orders to sell/buy shares.

The design of *Create and activate orders* business process task requires two-way communication and thus symmetric flow, which comprises at least two asymmetric flows.

B. USE CASES VIEW

The context of business processes is shown in the *Integrated processes* view of the 1+5 model. Activities in the business process may be of different natures. Some can be automated in the form of services or integration flows. Others may involve interactions with the system user. The *Use cases* view defines the system functions important to its users, i.e. use cases. The designs of these functions, i.e. use case realizations, are presented in the *Logical* view. In the context of the example of sending buy/sell orders, it is important to underline the beginning of the business process. In order to be able to send orders from a brokerage house to the stock exchange, someone has to place them. For this purpose, an application of a brokerage house is usually used. It must offer certain functions, including placing buy or sell orders. In the Use case diagram, we should identify all necessary use cases.

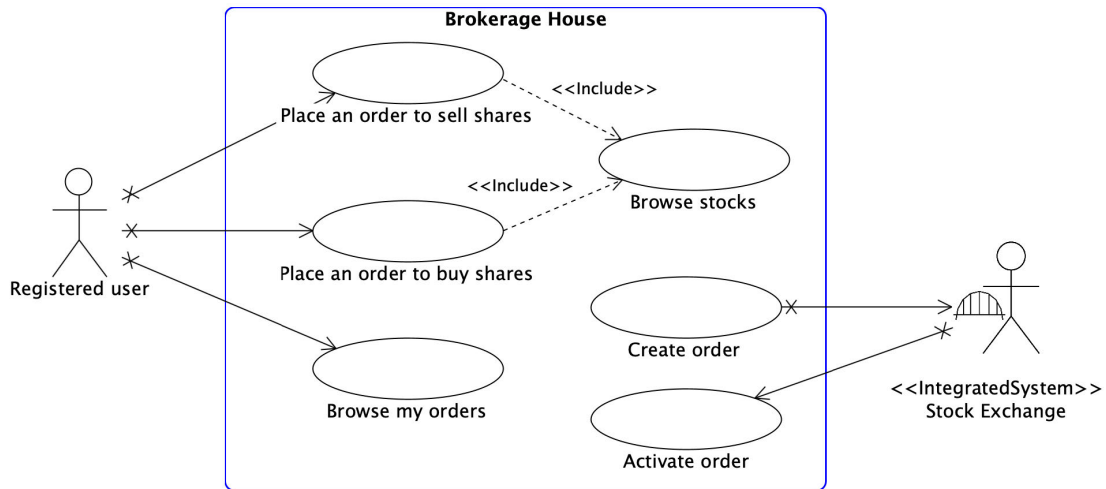


FIGURE 6. UML Use case diagram of the Brokerage House application.

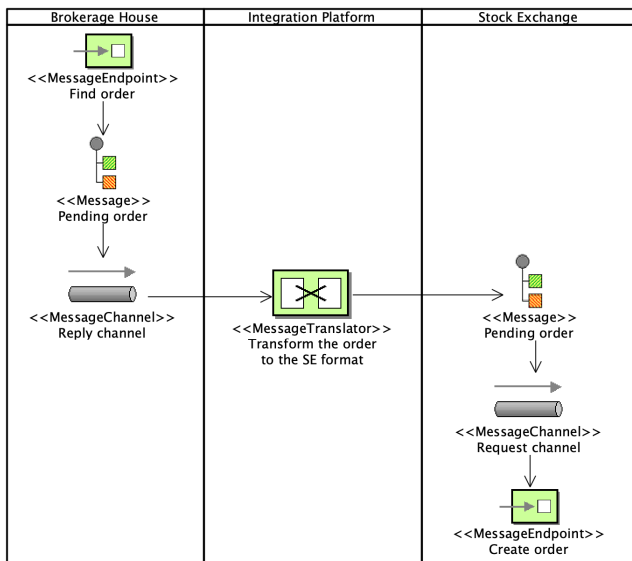


FIGURE 7. Integration flow diagram for creating the order.

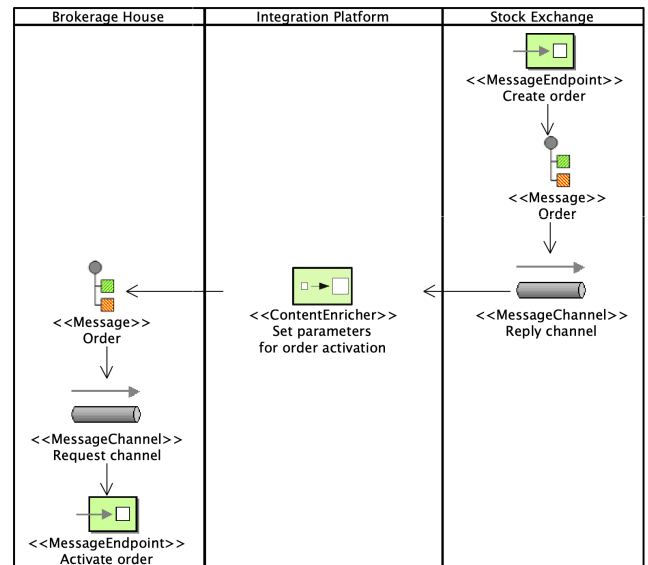


FIGURE 8. Integration flow diagram for activating the order.

Figure 6 presents the fragment of the UML Use case diagram for the Brokerage House application. Among others, the diagram shows the *Create order* and *Activate order* use cases that communicate with an external system. Marking the *Stock Exchange* actor with the `<<IntegratedSystem>>` stereotype determines the need for integration.

It has been shown to emphasize that in order to visualize the design of a complex integration system, it is necessary to look at its architecture from various views.

C. INTEGRATED SERVICES VIEW

For those two use cases (*Create order* and *Activate order*), the integration flows should be modeled in the *Integrated services* view. The dynamic aspect of communication is realized by integration flows.

1) SERVICE FLOW

In Integration flow diagrams components are represented as partitions and services are used as endpoints. By placing actions in partitions, we allocate responsibility to components. Figure 7 depicts the first integration flow. The flow is responsible for sending the order from the BH application to the SE system.

Control flows (in Figure 5) cross the brokerage house organization border two times. So, within the communication, we should identify at least two service flows. The first one flows from BH to SE - within *Create order* use case. The second one conversely, sends messages from SE to BH - within *Activate order* use case.

Figure 8 presents the second integration flow that activates the order in the BH application. But, we are not able to get a broader context for flow use or any dependencies.

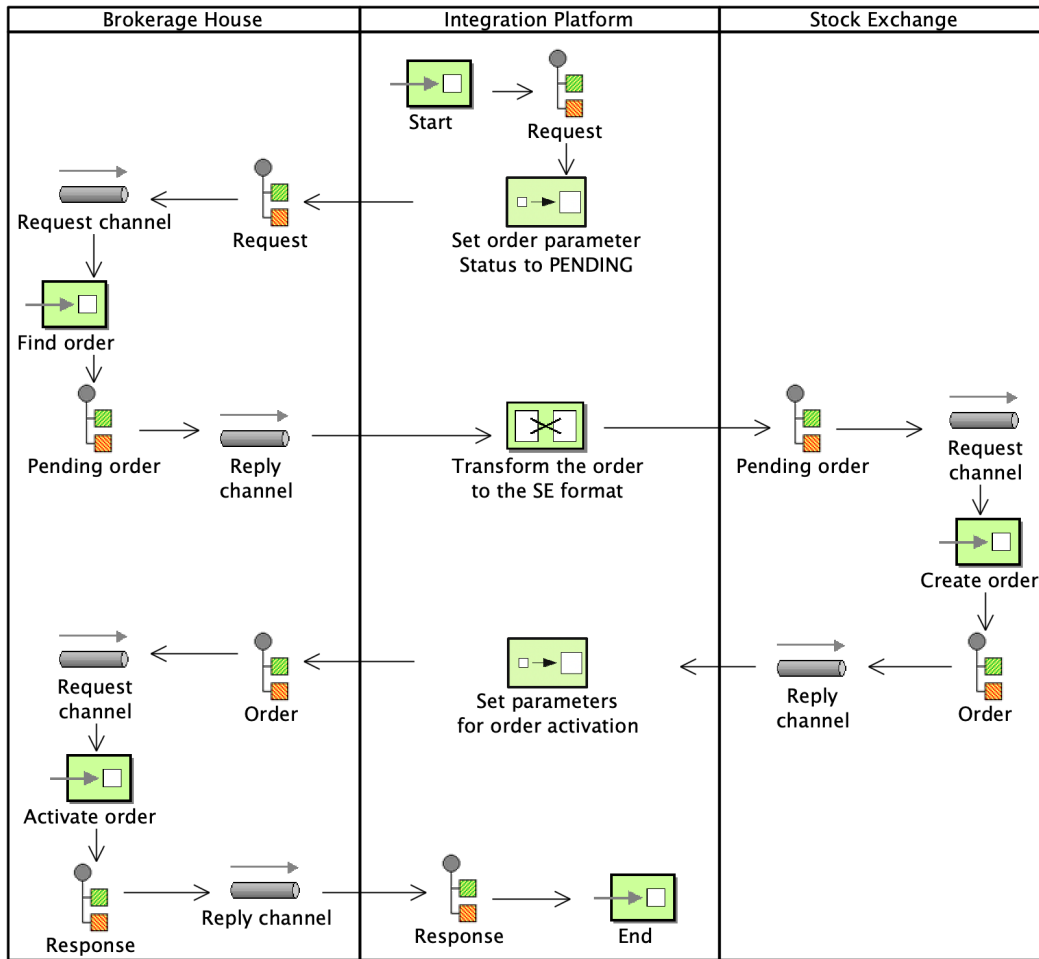


FIGURE 9. Integration flow diagram for communication in the context of *Create and activate orders* business process task.

2) BUSINESS FLOW

In the example of placing share buy/sell orders, there are four service flows combined into a logical chain of consecutive service invocations. Figure 9 shows the Integration flow diagram for communication in the context of *Create and activate orders* business process task and both use cases: *Create order* and *Activate order*.

The complete business integration flow includes starting the order reading flow initiated on the message bus, downloading orders from the brokerage house and sending them to the stock exchange, obtaining confirmation from the brokerage house that the orders have been created on the stock exchange, and getting a response from the brokerage house on the message bus. As has been shown, the integration design involves various architectural views. A single message transfer must be placed in the complete integration flow, which in turn should be situated in the context of a specific functional requirement. The prepared profile facilitates modeling and enables unambiguous visualization of message patterns.

3) COMPONENTS AND SERVICES

The structural aspect is modeled using a UML Component diagram and encompasses the systems components and message queue or service bus components. Two important pieces of information are presented in the UML Component diagram: the components that interact and the services provided by each component. Services are modeled using interfaces realized by components. The most important are the interfaces provided by the components representing IT systems. Because in these components, the functions necessary to perform the integration flow are actually implemented. The message queue or service bus component performs flow actions but forwards calls for execution to component services (they are hidden behind interfaces).

Figure 10 depicts the UML Component diagram, which contains services identified during analysis of *Create and activate orders* business process task. Both systems cooperate using the Integration Platform component.

Only by having modeled a full business flow, we can identify all the necessary services in the two components involved

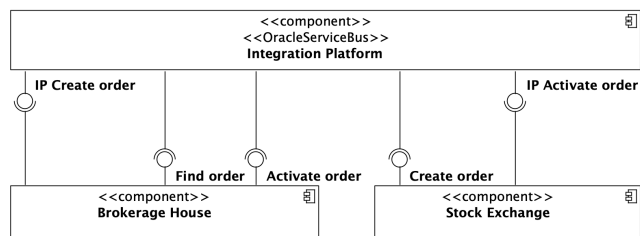


FIGURE 10. UML Component diagram for BH and SE systems cooperation.

in the communication. As a result of business flow modeling, two additional service flows, and one the *Find order* service were identified.

V. DISCUSSION AND LIMITATIONS

Several changes have been made to the integration flow modeling method. The first is adding an icon for the `<<IntegratedSystem>>` stereotype representing an integrated external system. A various level of modeling competencies of the UML Use case diagram has been observed among researchers and practitioners. As a rule, actors that represent humans usually are drawn on the left. Besides, actors that represent external systems usually are drawn on the right. Inside reside use cases as functions of the described system. This rule is not followed by everyone. In order to make the integration areas even more visible, a dedicated icon for the `<<IntegratedSystem>>` stereotype has been introduced. In the presented example (section IV), the *Stock Exchange* actor representing the integrated system is drawn on the right side of the use cases. On the left side, there is the *Registered user* actor representing the user of the system. The second change refers to UML Component diagrams for components taking part in integration flows. The `<<Consumer>>` and `<<Provider>>` stereotypes of the SoaML language have been abandoned in the UML component diagram. In a situation where the cooperating components provide their own services and use others, it is no longer justified to mark all of them with two more stereotypes. In addition, the use of interfaces in UML Component diagrams with the definition of the realization and usage relations brings in enough relevant information. Furthermore, it is recommended to draw interfaces close to the components that deliver (realize) them. As a third change, it was decided to introduce partitions on Integration flow diagrams. This change made it possible to clearly assign responsibilities for activities performed in the course of the integration flow. This is of particular importance in complex integration projects involving several individual flows. The author has also intended to prepare the profile in a commonly used and affordable modeling tool. That is why the Visual Paradigm Standard version has been chosen. The fourth element is the inclusion of the business process aspect. Only a broader perspective provides the context for the use of the single integration flow. Business flow covers the context of the business process task placed within the business process. It shows the whole interaction among participating services. Generally, UML proved to be the right choice for constructing

the set of modeling means and the method for integration flows design.

Outside the scope of patterns included in the profile, there are two important areas: IoT and blockchain. In terms of the Internet of Things, the analysis of Covert Channels seems to be important. Velinov et al. [35] introduce this topic. However, in the current version of the profile, protocols used in IoT applications are included [36]. The article shows that the area of communication between systems/services is constantly being developed and more and more flexible patterns are emerging. An interesting pattern is the Free-lance introduced in the ZeroMQ framework. It makes the communication mechanism independent of a single intermediary element. Therefore, it eliminates a single point of failure. These types of patterns can become the basis for communication between networks of distributed blockchain nodes. Such exchange of messages may also be organized in a distributed manner. Decentralized platforms for the stock exchange that are based on blockchain technology attract researchers' attention [37]. It seems interesting to verify and potentially enhance the proposed method in the case of the integration design of the blockchain-based distributed stock exchange. The area of model-driven engineering is also outside the scope of the article. Open Platform Communications Unified Architecture (OPC UA) is a platform-independent standard that facilitates information exchange between clients and servers using messages. Existing work by Pauker et al. [38] addresses an automatic transformation of UML class diagrams to OPC UA information models. In the case of the method, it would be interesting to investigate the feasibility of designing the transformation of the UML activity diagram. The work of Lee et al. [39] can be used because they define the mapping between UML and OPC UA constructs. The current work focuses on the modeling method. However, as part of further work, it would be worth designing transformations that automate the implementation of integration flows for selected message queues and enterprise service buses. This seems to be an interesting task. Because integration environments use a variety of integration flow implementation technologies, ranging from Java applications to Business Process Execution Language [40] flows. At the level of UML models, it is planned to use tagged values and the Object Constraint Language [41].

VI. CONCLUSION AND FURTHER WORK

The paper introduces business and service integration flows and methods of their modeling. It should be underlined that integration design involves various architectural views of the *1+5* model. Integration design is realized in the *Integrated services* view. Whereas, the business process context is shown in the *Integrated processes* view. Architecture description also requires using diverse diagrams. Business processes are modeled using UML Activity diagrams. While service and business integration are presented in Integration flow diagrams. The structural aspect of communication is shown in UML Component diagrams. A business flow realizes a

business process task and its implementation should be rather done by enterprise service bus. Whereas, service flows are well-suited for single service invocation and thus may be recommended in a microservices architecture. As it was presented, integration flows can be diverse and quite complex. However, one should strive to make communication between systems or services as simple as possible. It should be underlined that the proposed modeling methods are generic and not limited to a single technology or architecture. Besides, integration flows are modeled using one notation, the Unified Modeling Language. The same notation is used to model the other aspects of the integration solution in the individual architectural views of the 1+5 model. Activities in a business process may be of different natures. Some can be automated in the form of services or integration flows. Others may involve interactions with the system user. The *Use cases* view defines the system functions important to its users. Therefore, it was also included in the architectural description of the integration.

As further work, it is planned to include the *Contracts* view of the model in the integration design. This is especially important when designing solutions for which Service Level Agreements have been established. Adapting the method for designing communication for a distributed stock exchange, which employs blockchain technology is also considered. Designing the method for generating executable integration flows for complex messaging patterns may be also a promising research direction.

REFERENCES

- [1] P. Kannisto, D. Hästbacka, T. Gutiérrez, O. Suominen, M. Vilkkö, and P. Craamer, "Plant-wide interoperability and decoupled, data-driven process control with message bus communication," *J. Ind. Inf. Integr.*, vol. 26, Mar. 2022, Art. no. 100253, doi: [10.1016/j.jii.2021.100253](https://doi.org/10.1016/j.jii.2021.100253).
- [2] R. S. Bhadoria, N. S. Chaudhari, and V. G. T. N. Vidanagama, "Analyzing the role of interfaces in enterprise service bus: A middleware epitome for service-oriented systems," *Comput. Standards Interfaces*, vol. 55, pp. 146–155, Jan. 2018, doi: [10.1016/j.csi.2017.08.001](https://doi.org/10.1016/j.csi.2017.08.001).
- [3] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA: Addison-Wesley, 2004.
- [4] O. Aziz, M. S. Farooq, A. Abid, R. Saher, and N. Aslam, "Research trends in enterprise service bus (ESB) applications: A systematic mapping study," *IEEE Access*, vol. 8, pp. 31180–31197, 2020, doi: [10.1109/ACCESS.2020.2972195](https://doi.org/10.1109/ACCESS.2020.2972195).
- [5] H. F. Martinez, O. H. Mondragon, H. A. Rubio, and J. Marquez, "Computational and communication infrastructure challenges for resilient cloud services," *Computers*, vol. 11, no. 8, p. 118, Jul. 2022, doi: [10.3390/computers11080118](https://doi.org/10.3390/computers11080118).
- [6] Y. Zhong, S. Zhu, Y. Wang, J. Li, X. Zhang, and J. S. Shang, "Pairwise location-aware publish/subscribe for geo-textual data streams," *IEEE Access*, vol. 8, pp. 211704–211713, 2020, doi: [10.1109/ACCESS.2020.3038921](https://doi.org/10.1109/ACCESS.2020.3038921).
- [7] I. Livaja, K. Pripuzić, S. Sovilj, and M. Vuković, "A distributed geospatial publish/subscribe system on apache spark," *Future Gener. Comput. Syst.*, vol. 132, pp. 282–298, Jul. 2022, doi: [10.1016/j.future.2022.02.013](https://doi.org/10.1016/j.future.2022.02.013).
- [8] E. Daraghmi, C.-P. Zhang, and S.-M. Yuan, "Enhancing saga pattern for distributed transactions within a microservices architecture," *Appl. Sci.*, vol. 12, no. 12, p. 6242, Jun. 2022, doi: [10.3390/app12126242](https://doi.org/10.3390/app12126242).
- [9] M. Ozkaya and F. Erata, "A survey on the practical use of UML for different software architecture viewpoints," *Inf. Softw. Technol.*, vol. 121, May 2020, Art. no. 106275, doi: [10.1016/j.infsof.2020.106275](https://doi.org/10.1016/j.infsof.2020.106275).
- [10] T. Pender, "Customizing UML using profiles," in *UML Bible*. Indianapolis, IN, USA: Wiley, 2003, ch. 21, pp. 687–723.
- [11] R. J. Petrasch and R. R. Petrasch, "Data integration and interoperability: Towards a model-driven and pattern-oriented approach," *Modelling*, vol. 3, no. 1, pp. 105–126, Feb. 2022, doi: [10.3390/modelling3010008](https://doi.org/10.3390/modelling3010008).
- [12] T. Górski, "UML profile for messaging patterns in service-oriented architecture, microservices, and Internet of Things," *Appl. Sci.*, vol. 12, no. 24, p. 12790, Dec. 2022, doi: [10.3390/app122412790](https://doi.org/10.3390/app122412790).
- [13] T. Górski, "The 1+5 architectural views model in designing blockchain and IT system integration solutions," *Symmetry*, vol. 13, no. 11, p. 2000, Oct. 2021, doi: [10.3390/sym13112000](https://doi.org/10.3390/sym13112000).
- [14] I. K. Kirpitsas and T. P. Pachidis, "Evolution towards hybrid software development methods and information systems audit challenges," *Software*, vol. 1, no. 3, pp. 316–363, Aug. 2022, doi: [10.3390/software1030015](https://doi.org/10.3390/software1030015).
- [15] P. B. Kruchten, "The 4+1 view model of architecture," *IEEE Softw.*, vol. 12, no. 6, pp. 42–50, Nov. 1995, doi: [10.1109/52.469759](https://doi.org/10.1109/52.469759).
- [16] A. Suljkanović, B. Milosavljević, V. Indić, and I. Dejanović, "Developing microservice-based applications using the silvera domain-specific language," *Appl. Sci.*, vol. 12, no. 13, p. 6679, Jul. 2022, doi: [10.3390/app12136679](https://doi.org/10.3390/app12136679).
- [17] K. Zhang and J. Kianfar, "An automatic incident detection method for a vehicle-to-infrastructure communication environment: Case study of interstate 64 in Missouri," *Sensors*, vol. 22, no. 23, p. 9197, Nov. 2022, doi: [10.3390/s22239197](https://doi.org/10.3390/s22239197).
- [18] United States Department of Transportation. (Mar. 2023). *Architecture Reference for Cooperative and Intelligent Transportation, ARC-IT Version 9.1*. [Online]. Available: <https://www.arc-it.net/>
- [19] D. Marboub, M. C. E. Simsekler, K. Salah, R. Jayaraman, and S. Ellahham, "A blockchain-based regulatory framework for mHealth," *Data*, vol. 7, no. 12, p. 177, Dec. 2022, doi: [10.3390/data7120177](https://doi.org/10.3390/data7120177).
- [20] C. G. Moyano, L. Pufahl, I. Weber, and J. Mendling, "Uses of business process modeling in agile software development projects," *Inf. Softw. Technol.*, vol. 152, Dec. 2022, Art. no. 107028, doi: [10.1016/j.infsof.2022.107028](https://doi.org/10.1016/j.infsof.2022.107028).
- [21] T. Górski and A. P. Woźniak, "Optimization of business process execution in services architecture: A systematic literature review," *IEEE Access*, vol. 9, pp. 111833–111852, 2021, doi: [10.1109/ACCESS.2021.3102668](https://doi.org/10.1109/ACCESS.2021.3102668).
- [22] H. Wang, X. Hu, Q. Yu, M. Gu, W. Zhao, J. Yan, and T. Hong, "Integrating reinforcement learning and skyline computing for adaptive service composition," *Inf. Sci.*, vol. 519, pp. 141–160, May 2020, doi: [10.1016/j.ins.2020.01.039](https://doi.org/10.1016/j.ins.2020.01.039).
- [23] A. Montarnal, W. Mu, F. Benaben, J. Lamothe, M. Lauras, and N. Salatge, "Automated deduction of cross-organizational collaborative business processes," *Inf. Sci.*, vol. 453, pp. 30–49, Jul. 2018, doi: [10.1016/j.ins.2018.03.041](https://doi.org/10.1016/j.ins.2018.03.041).
- [24] R. Akhilesh, O. Bills, N. Chilamkurti, and M. J. M. Chowdhury, "Automated penetration testing framework for smart-home-based IoT devices," *Future Internet*, vol. 14, no. 10, p. 276, Sep. 2022, doi: [10.3390/fi14100276](https://doi.org/10.3390/fi14100276).
- [25] J. Han, Y. Zhang, J. Liu, Z. Li, M. Xian, H. Wang, F. Mao, and Y. Chen, "A blockchain-based and SGX-enabled access control framework for IoT," *Electronics*, vol. 11, no. 17, p. 2710, Aug. 2022, doi: [10.3390/electronics11172710](https://doi.org/10.3390/electronics11172710).
- [26] W. Ahmed, W. Di, and D. Mukathe, "A blockchain-enabled incentive trust management with threshold ring signature scheme for traffic event validation in VANETs," *Sensors*, vol. 22, no. 17, p. 6715, Sep. 2022, doi: [10.3390/s22176715](https://doi.org/10.3390/s22176715).
- [27] K. Thramboulidis and F. Christoulakis, "UML4IoT—A UML-based approach to exploit IoT in cyber-physical manufacturing systems," *Comput. Ind.*, vol. 82, pp. 259–272, Oct. 2016, doi: [10.1016/j.compind.2016.05.010](https://doi.org/10.1016/j.compind.2016.05.010).
- [28] H. Marouane, C. Duvallet, A. Makni, R. Bouaziz, and B. Sadeg, "An UML profile for representing real-time design patterns," *J. King Saud Univ.*, *Comput. Inf. Sci.*, vol. 30, no. 4, pp. 478–497, Oct. 2018, doi: [10.1016/j.jksuci.2017.06.005](https://doi.org/10.1016/j.jksuci.2017.06.005).
- [29] J. E. Plazas, S. Bimonte, M. Schneider, C. de Vaulx, P. Battistoni, M. Sebillio, and J. C. Corrales, "Sense, transform & send for the Internet of Things (STS4IoT): UML profile for data-centric IoT applications," *Data Knowl. Eng.*, vol. 139, May 2022, Art. no. 101971, doi: [10.1016/j.datak.2021.101971](https://doi.org/10.1016/j.datak.2021.101971).
- [30] GitHub Repository. *UML Profile for Messaging Patterns*. Accessed: Mar. 7, 2023. [Online]. Available: <https://github.com/drGorski/UMLProfile4MessagingPatterns>
- [31] ZeroMQ Guide. *Publish-Subscribe Messaging Patterns*. Accessed: Mar. 7, 2023. [Online]. Available: <https://zguide.zeromq.org/docs/chapter5/>

- [32] ZeroMQ Guide. *Request-Reply Messaging Patterns*. Accessed: Mar. 7, 2023. [Online]. Available: <https://zguide.zeromq.org/docs/chapter4/>
- [33] *Apache Camel—Enterprise Integration Patterns*. Accessed: Mar. 7, 2023. [Online]. Available: <https://camel.apache.org/components/3.20.x/eips/enterprise-integration-patterns.html>
- [34] Object Management Group. *Business Process Model and Notation, Specification 2.0.2*. Accessed: Mar. 7, 2023. [Online]. Available: <https://www.omg.org/spec/BPMN/>
- [35] A. Velinov, A. Mileva, S. Wendzel, and W. Mazurczyk, “Covert channels in the MQTT-based Internet of Things,” *IEEE Access*, vol. 7, pp. 161899–161915, 2019, doi: [10.1109/ACCESS.2019.2951425](https://doi.org/10.1109/ACCESS.2019.2951425).
- [36] E. Al-Masri, K. R. Kalyanam, J. Batts, J. Kim, S. Singh, T. Vo, and C. Yan, “Investigating messaging protocols for the Internet of Things (IoT),” *IEEE Access*, vol. 8, pp. 94880–94911, 2020, doi: [10.1109/ACCESS.2020.2993363](https://doi.org/10.1109/ACCESS.2020.2993363).
- [37] H. Al-Shaibani, N. Lasla, and M. Abdallah, “Consortium blockchain-based decentralized stock exchange platform,” *IEEE Access*, vol. 8, pp. 123711–123725, 2020, doi: [10.1109/ACCESS.2020.3005663](https://doi.org/10.1109/ACCESS.2020.3005663).
- [38] F. Pauker, S. Wolny, S. M. Fallah, and M. Wimmer, “UML2OPC-UA Transforming UML class diagrams to OPC UA information models,” *Proc. CIRP*, vol. 67, pp. 128–133, Jan. 2018, doi: [10.1016/j.procir.2017.12.188](https://doi.org/10.1016/j.procir.2017.12.188).
- [39] B. Lee, D.-K. Kim, H. Yang, and S. Oh, “Model transformation between OPC UA and UML,” *Comput. Standards Interfaces*, vol. 50, pp. 236–250, Feb. 2017, doi: [10.1016/j.csi.2016.09.004](https://doi.org/10.1016/j.csi.2016.09.004).
- [40] OASIS. *Web Services Business Process Execution Language, Version 2.0*. Accessed: Mar. 7, 2023. [Online]. Available: <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf>
- [41] Object Management Group. *Object Constraint Language, Version 2.4*. Accessed: Mar. 7, 2023. [Online]. Available: <https://www.omg.org/spec/OCL/2.4/PDF>



TOMASZ GÓRSKI (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from the Military University of Technology (MUT), Warsaw, Poland, in 1997 and 2000, respectively. He was with the Computer Science Center of the General Staff of the Polish Armed Forces for ten years and ended his military service as a major. After getting the Ph.D. degree, he worked on public-sector (Polish Border Guard, PKP CARGO) and commercial integration projects (Zone Vision). From 2005 to 2020, he ran the consulting company RightSolution, an IBM Authorized Training Provider for the Rational brand. From 2004 to 2017, he led the IT Systems Engineering Department, MUT, and built the Center for Advanced Studies in Systems Engineering. From 2017 to 2022, he steered the IT Systems Department, Polish Naval Academy. He is currently with the Institute of Computer Science, University of Gdansk. He reviews papers for *Applied Sciences*, *IEEE Access*, *Information and Software Technology*, and *Journal of Industrial Information Integration*. He teaches object-oriented programming. His research interests include software architecture, software design and test patterns, blockchain, model-driven development, and continuous delivery. He is the author of the 1+5 Architectural Views Model in Designing Blockchain and IT System Integration Solutions, Use Case API Design Pattern, UML Profile for Distributed Ledger Deployment, UML Profile for Messaging Patterns, Smart Contract Design Pattern, and k+1 Symmetric Test Pattern. He is a member of the IEEE Computer and IEEE Information Theory Societies.

• • •