Practical Experience with Petriflow: Enriched Process Models Serving as Implementation

Gabriel Juhás [1], Milan Mladoniczky [2], and Ľuboš Petrovič [2]

Abstract: In this paper we discuss experiences gained by using low-code language Petriflow, based on extended Petri nets enriched by data variables and forms. We illustrate on several real-life use cases how the Petriflow models of business processes can directly be used as implementation when deployed in Petriflow interpreter.

Keywords: Low-code, Petri nets, Petriflow

1 Introduction to low-code language Petriflow

In the paper, we will discuss how Petriflow, a low-code language based on enriched process models, can be used for process-driven application development.

Petriflow is a high-level modelling language for process-driven application development [Ju21b]. Petriflow follows the programming paradigm called process-driven programming (PDP) [Ju19]. A comparison of Petriflow's concept with other well-known programming paradigms is essential to understanding the meaning of the PDP paradigm. Petriflow language combines the advantages of object-oriented programming (OOP), business process modelling (BPM) [ADO00; DPW04], event-driven programming (EDP) and relational databases (RDB).

Object-oriented programming languages brought a distinguishing feature – encapsulation. This was a key step for the business world that needed new languages to solve the problem of the growing complexity of applications. OOP was a concept, which got rid of the problems that developers had with procedural and imperative programming. The concept of classes that contain data and methods strongly supports the modularity of applications.

While binding methods with the data in classes was one of the main features of OOP that helped to create more modular programs, Petriflow is binding a workflow process to a class to describe the life cycle of objects of the class.

1 Pan-European University, Faculty of Informatics, Tematínska 10, 85105 Bratislava, Slovak Republic, juhas@netgrif.com, https://orcid.org/0000-0001-8302-5112
2 NETGRIF, s.r.o., Slávičie údolie 106, 811 02, Bratislava , mladoniczky@netgrif.com, https://orcid.org/0000-0002-1385-2304; petrovic@netgrif.com, https://orcid.org/0009-0005-6142-6827
This work is licensed under Creative Commons Attribution 4.0 International License http://creativecommons.org/licenses/by/4.0/, https://doi.org/10.18420/modellierung2024-ws-024

The main building blocks of object-oriented programs are classes and their objects. In comparison, the main building blocks of process-driven programs in the Petriflow language are processes and their instances called cases. A class is a blueprint of an object and a Petriflow process is a blueprint of a Petriflow process instance or case. Simply, a Petriflow process is a class enriched by a workflow process that defines the life cycle of the objects of that class. More accurately, a Petriflow process consists of data variables, tasks and actions, roles, and a workflow process. In the same way as in classes in OOP, data variables in Petriflow processes represent all attributes of a Petriflow process instance. The change of the value of a data variable can be triggered by a so-called set-event. Reading a value of a data variable can be triggered by a so-called get-event.

Tasks are the active parts of the Petriflow processes. Data variables can be associated with workflow tasks to define data fields and create task forms. A data field, which is an association of a data variable to a task, is given as a rich relation, that states:

- whether a get-event and/or a set-event can trigger the data variable, i.e. whether its value is readable and/or editable,
- whether the value of data variable is required,
- what are valid values of the data variable within the data field.

Tasks have a simple life cycle: a task can be enabled, disabled or executed. Change of the state of a task can be triggered as follows:

- if a task is enabled, its change to the state executed can be triggered by a so-called assign-event
- if a task is enabled, its readable data fields are accessible for reading by get-events
- if a task is executed, its readable data fields are accessible for reading by get-events
- if a task is executed, its editable data fields can be changed to valid values by set-events
- if a task is executed and all its required data fields have valid values, its change to the state enabled or disabled can be triggered by a so-called finish-event.

Using the principles of event-driven programming, each data variable and each task is associated with an event listener: whenever an event triggers a change in the value of a data variable or whenever an event triggers a change in the state of a task, then a reaction can be defined by pieces of code called actions in the event listener. Whenever an event is occurring, the actions in its event listener are executed. In actions, as a part of the code, events for tasks and events for data variables can be emitted. In this way, events and their reactions can create chains. Roles [BDM11] or lists of users can be associated with events of tasks, defining for each task which users are authorized to emit events on that task. Similarly to data fields, an association of users with events is a rich relation. For example, a user authorized to emit an assign-event of a task can emit the assign-event. By emitting the assign-event, this user has to choose one of the users that are authorized to possibly emit the

finish-event of this task and only this user is then authorized to emit set-events of editable data fields of this task and to emit the finish-event of this task. In other words, by emitting an assign-event, the authorized user is assigning that task to a user (possibly himself), that is authorized to perform the task, i.e. to fill editable data fields and finish the task.

As a workflow process, Petriflow language uses place/transition Petri nets [De05; DJ01; DR15] enriched by reset arcs [Aa09], inhibitor arcs [AJ15] and read arcs [Vo02] to define the life cycle of the Petriflow process. Places of the Petri net represent the control variables. Transitions of Petri net represent tasks of the workflow process. A task is enabled, whenever the corresponding transition in the underlying Petri net is enabled. Assign-event occurring on this task consumes tokens from the input places of the corresponding transition and moves the state of the task to be executed. The finish-event on the task being executed produces tokens to the output places of the corresponding transition. In this way, the workflow process defines when a task is enabled, executed or disabled. The life cycle of the Petriflow process is given as flow of assign/get/set/finish events on tasks and data variables respecting the restrictions on events given by the underlying Petri net.

When compared with relational databases, Petriflow processes correspond to tables, while instances (cases) of the Petriflow processes correspond to single records (rows) of a table. In a similar way to foreign keys in RDB and in a similar way to attributes of objects containing references to other objects in OOP, data variables of Petriflow processes can store the references to instances of Petriflow processes and references to the task and list of tasks of Petriflow process instances. In this way, one can easily share forms associated with one task as sub-forms within other tasks and implement a single source of truth architecture.

Process models in Petriflow language employ principles of object-oriented programming, relational databases and event-driven programming combining them with user authorization and the concept of the life cycle of objects using workflow processes borrowed from business process management. All this should bring a higher level of programming, with information about the data layer, application (process) layer and presentation layer (forms) covered by a single Petriflow object with the aim to make the development of complex applications more structured, closer to the business user, faster, without the necessity to deal with the implementation details of the middleware. Netgrif application builder (NAB) is then a platform for developing process-driven applications. NAB produces process models in Petriflow language – a combination of XML and Groovy. This code can directly be interpreted in Netgrif application engine – Petriflow interpreter written in Java using the framework Spring Boot and storing the data of Petriflow process instances in MongoDB. Similarly to SQL build over relational databases, Petriflow language also provides a powerful query language that enables to create filters over process instances and their tasks [JJP23; Ju22].

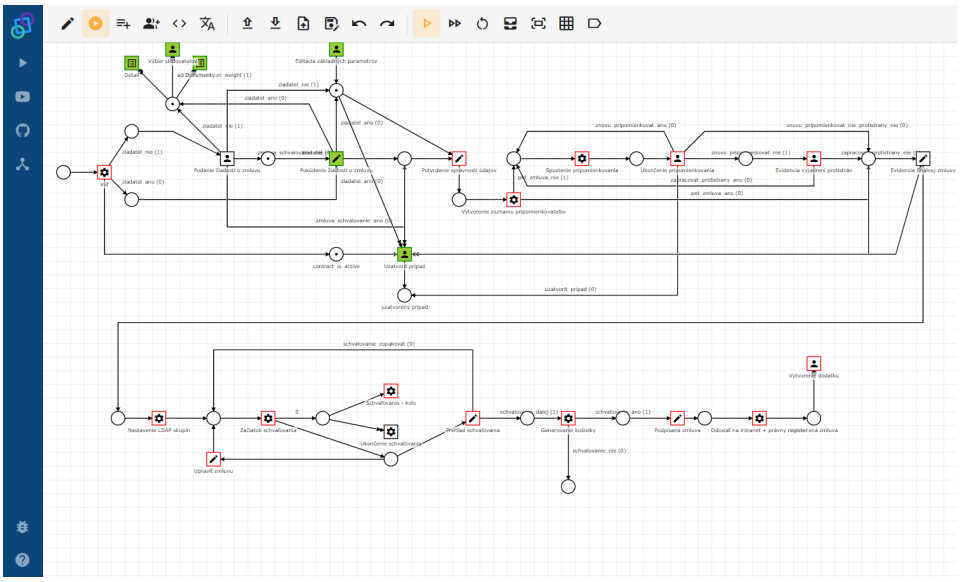

Fig. 1: Application Builder with general contracts processing

2 Use cases

There are many implementations of process automation using low-code language Petriflow. We have chosen four case studies, one from utility, one from insurance, one from healthcare and one from financial leasing, to illustrate how enriched process models in low-code language Petriflow can serve as an implementation of process-based information systems.

2.1 Contract management in a utility company

First, we briefly describe a case study for a well-established supplier of electricity and gas in Slovakia. The company supplies energy to more than 700,000 customers. Using Petriflow process models described above, we implemented a project thanks to which households can sign a contract for the supply of electricity or gas online. It is enough to fill in the data in the form application and then electronically sign the contract documentation. The entire process takes place online without the need for personal contact, or sending physical mail, and it is also environmentally friendly. The contract documentation is available to the customer immediately after signing it in his/her email. It is followed by the processing of the received request on the supplier side. The integration of new online processes into the existing infrastructure also required the deployment of the Netgrif Application Engine as BPM platform that automatically controls the process flow and moves the request between individual systems. The BPM platform communicates with a form application, a signature solution, a document repository, and a system for processing received requests.

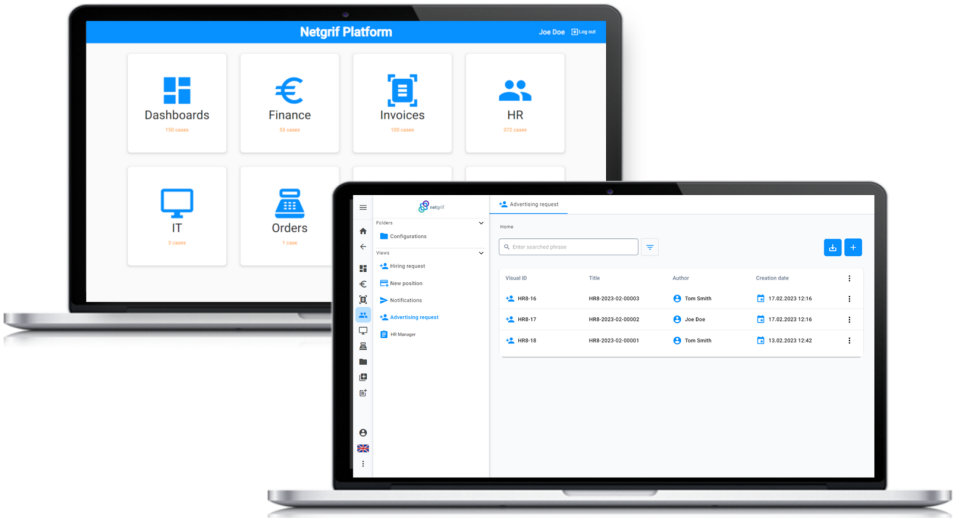

Fig. 2: eTask application

2.2 Process automation platform in an insurance company

Second, we describe a successful use case for the insurance company Union. Union is a leading insurance company in Slovakia, it is a member of Achmea group based in Netherlands. The Union company has more than 1 million customers and is active in life insurance, company insurance, travel insurance and individual health insurance. The first time Petriflow processes running in Netgrif low-code application development platform were used in Union was in 2018. Union wanted to digitize non-life and life insurance documents. Originally, these documents were all scanned manually. The aim of Union was was to replace certain internal processes and systems, some of which were very clumsy, inflexible, difficult to use, not customizable and hard to maintain. The aim was to transform this into a well-functioning digital process and Union used the Netgrif platform to a Proof of Concept. Netgrif Application Builder was used to develop the application for the various forms and shifted this – over time – to an e-flow system, an internal enterprise workflow for Union, which is now the backbone of the Union organization. Later, Union used Netgrif platform to automate the processing of the health care application forms, reducing the manual workload. During the period of 2018-2024 Union has been able to centralize more and more business requests via the workflows that were defined in Petriflow language and are now running in Netgrif platform, including claim processing and multichannel customer communication processing, internally called the Backbone. From the beginning, Union has had a desire to run the project autonomously. They started to build internal skills and capabilities in the Petriflow language, so they could run Netgrif as a platform internally, without the need to involve external developers. Today, Union continues to work on developing their Petriflow-based solution further to encompass more and more processes

and workflows, both internal processes and customer-facing processes including customers forms, all with the aim of saving valuable time and resources.

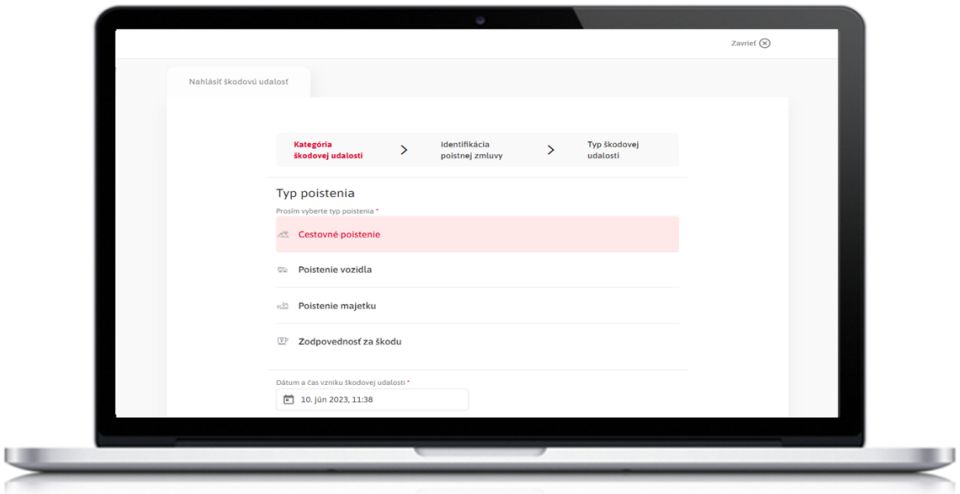

Fig. 3: eForm application

2.3 Process automation platform in a healthcare provider

Third, we describe an agenda system for all non-medical processes in AGEL. AGEL is the most successful healthcare provider in Central Europe. In the Czech Republic alone, AGEL operates 14 hospitals, health centers, a network of pharmacies, laboratories, and distribution companies, along with other specialized medical facilities with more than 13 thousand employees. The wide variety of hospitals that are operated by AGEL has led to a very diverse application landscape. All these hospitals have acquired a number of software products to manage their internal processes. Many of the solutions that were in use, were essentially doing the same thing. It was clear that AGEL could work much more efficiently if they aggregated all processes in one environment. So, AGEL decided to consolidate, streamline and centralize this by implementing one central platform for all non-medical agendas within the AGEL group. Eventually, using Petriflow processes deployed in Netgrif platform, AGEL were able to shut down a number of existing systems and replace them with Netgrif eTask application for employees. Agendas already implemented in Petriflow and running in Netgrif platform include interconnected contract processing, order processing and incoming invoice processing with integrated parametrized multiple series-parallel approval, as well as HR agenda. In the past, employees worked in multiple systems. As a result, AGEL employees had to learn to work in different environments which led to high onboarding and training costs. Also, from an IT perspective, AGEL had to support and maintain several applications. By aggregating all these processes in one environment, AGEL have already been able to significantly increase efficiency. The aim of AGEL is to

continue to migrate applications to the Netgrif environment until all non-medical agendas are consolidated. Similarly to Union, AGEL builds their own team of Petriflow developers.

2.4 Integration dealer portal in a leasing company

The last chosen use case describes the Integrated dealer portal for Tatra-Leasing. Tatra-Leasing is a subsidiary of Tatra Banka, a member of Raiffeisen Group. Tatra-Leasing is using Petriflow processes running in the Netrgif platform for the Integrated Dealer Portal (IDP). IDP establishes a common ground for the leasing company, Tatra, the dealers that Tatra partners with and their respective brand importers, such as Ford Automotive. Tatra developed a Petriflow-based app, a portal, which allowed dealers to keep an eye on the stocks, at all times. Besides, they could order cars via the application, get orders approved, communicate directly with the importer, and access all relevant data directly via this application. The first thing the development team did was: speak to the people within the company who were working on stock financing on a daily basis and – based on these interviews - describe the process in detail. Based on this description, Petriflow developers were able to start developing the application, which was designed around our work process. The resulting Petriflow application became the bridge between dealers, importers and Tatra. In early 2022, the application also had to be moved to the cloud. The bank decided that as part of the merger project, the IDP application would be migrated from an on-prem solution to the AWS cloud. The Integrated Dealer Portal has been fully functional in AWS cloud for a while now. It is the preferred way of communication for the Tatra business partners. It also makes the interaction with both importers and dealers much easier. Besides, significant cost savings can be made since certain steps in the process are automated. Also, the application has led to a strong improvement in the data quality: there is a single data source and a single point of truth now. The implementation includes integration between Tatras core system and the IDP.

3 Typical architecture of Petriflow applications running in the Netgrif platform

The typical architecture of Petriflow applications running in the Netgrif platform is depicted in Figure 4. The core of the Netgrif platform is the Netgrif application engine, which actually is a Java-Spring Boot-based BPM server interpreting workflows in extended Petri nets automatically storing data in the persistence layer using Mongo DB indexed via Elasticsearch. There are two Angular web clients, namely eForm for anonymous users (typically customers) and eTask for logged users (typically employees, but also registered customers). Netgrif application engine can be integrated with other information systems using web services, both REST and SOAP, or DB connectors.

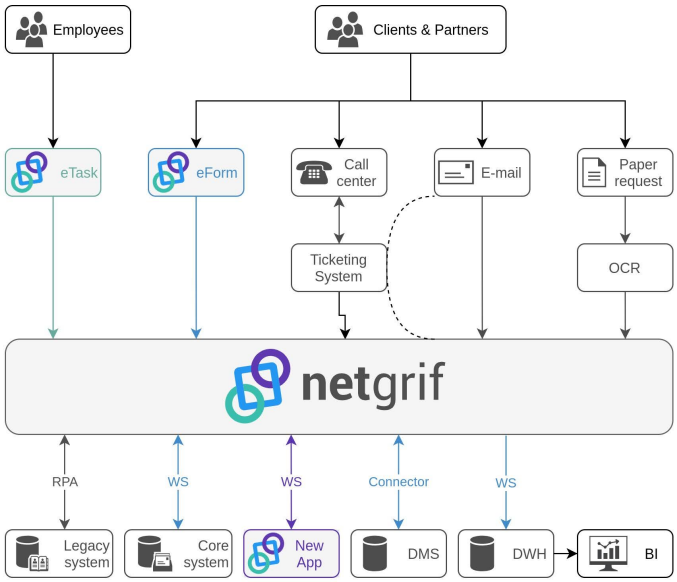

Fig. 4: Architecture of Petriflow applications

4 Experience with enriched process models in Petriflow serving as implementation

The observation obtained from practical experiences with developing applications using the low-code language Petriflow and running them in the Netgrif low-code platform, including the cases mentioned above, can be summarized as follows.

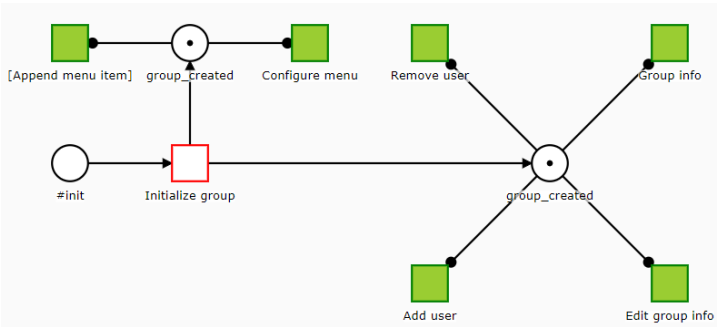

Fig. 5: Read arc

Petri nets can be successfully used to model implementable workflow processes, but they need to be extended. Explicit modelling of state via markings is an advantage of Petri nets in comparison to models without explicit states, such as BPMN. As suggested in

[AH02; AJ15], to keep Petri net models enough simple, transitions should model durable activities, and respective tasks instead of events. Thus, assign event of a task results in the consummation of tokens and finish event of task results in production of tokens. Triggering an event in reaction to another event resulting in the synchronization of events enables to interconnecting of different cases quite effectively. Extensions such as read arcs and inhibitor arcs are suitable. Typically, read arcs are used to model the parallel execution of tasks in a given marking, see Figure 5.

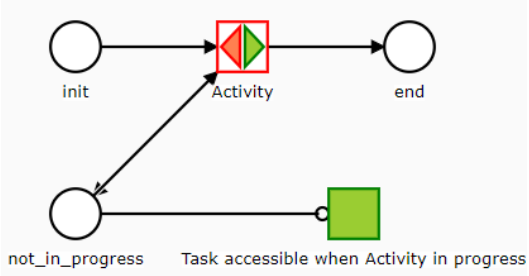

Fig. 6: Inhibitor arc

Inhibitor arcs can be used to check whether a task is assigned to a user as it is illustrated in Figure 6. Reset arcs can model resetting from a set of states, typically, when one need to set marking back, see Figure 7.

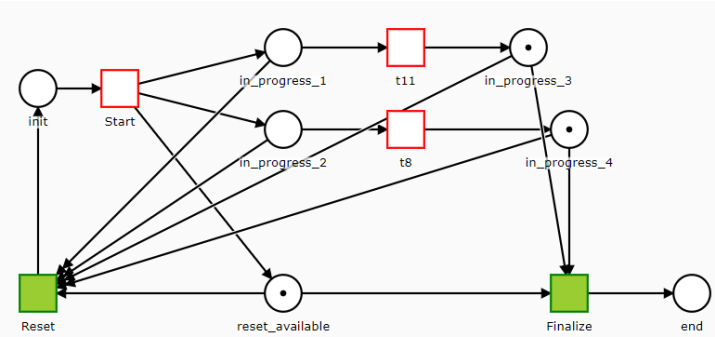

Fig. 7: Reset arc

Extension of the concept of self-modifying nets [Va78] to so-called arcs with variable weights can successfully be used to model or-splits and or-join as it is illustrated in Figure 8. This means, that actual nonnegative values of integer data attributes are used as weights of arcs. The main power of the Petriflow processes is that they truly represent object-centric processes and their interconnection. Here the fact that a Petriflow process consists of

- data attributes representing data layer (including a data type for a reference to (a list of) cases (a caseRef) and a data type for a reference to (a list of) tasks (a taskRef)).

- process consisting of a workflow of tasks given by extended Petri net representing the life-cycle of data attributes
- forms representing a user interface that can be associated to tasks

enables to create rich relationships between processes and their cases.

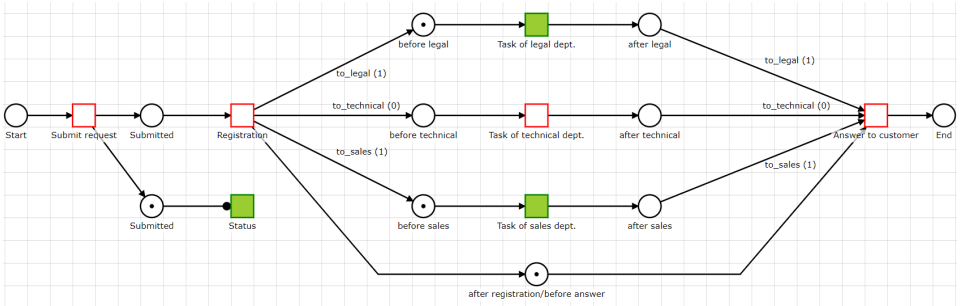

Fig. 8: Arcs with variable weights modelling or-split and or-join

Let us illustrate such one-to-many rich relationship on a simple example from one of the mentioned use cases. Imagine that you have a Petriflow process for an order and a Petriflow process for an incoming invoice, with one-to-many relationship between an order case and its incoming invoice cases. Using data attribute caseRef in incoming invoices one can pair the invoices to orders on data layer. Imagine that the order process has a task for a bulk approval of all incoming invoices, that have caseRef to the given case of order process. Imagine also, that the invoice process has a task for approval of the given invoice case. Thus, assigning the event of a bulk approval task synchronizes assigning events of invoice approval task in each case of invoice process with caseRef to the order process. Now, one would like to see the forms associated with approval tasks of single invoices as sub-forms of the form associated to the bulk approval of the case of the order process. This can be easily modelled by using a taskRef attribute in the order case, with the setting its value to the list of approval tasks of all invoice cases that have the caseRef to the given order case as it is illustrated in Figure 9. Now, the finish event of the bulk approval task of the order will synchronize the finish event of the approval task of each invoice case in the taskRef.

Our experience with the application of Petriflow-based object-centric processes to different use cases is that the resulting models can be successfully used as implementation in the Netgrif low-code platform for use cases of practically any complexity. Petrilow is an open source low-code language, the Community version Netgrif platform is available online for free and its source code can be downloaded from GitHub. Both Netgrif Application Builder (builder.netgrif.com) for modelling Petriflow processes [Ju21b], and a cloud demo version of Netgrif Application Engine (etask.netgrif.cloud) for deploying and running Petriflow processes can also be used online for free [Ju21a].

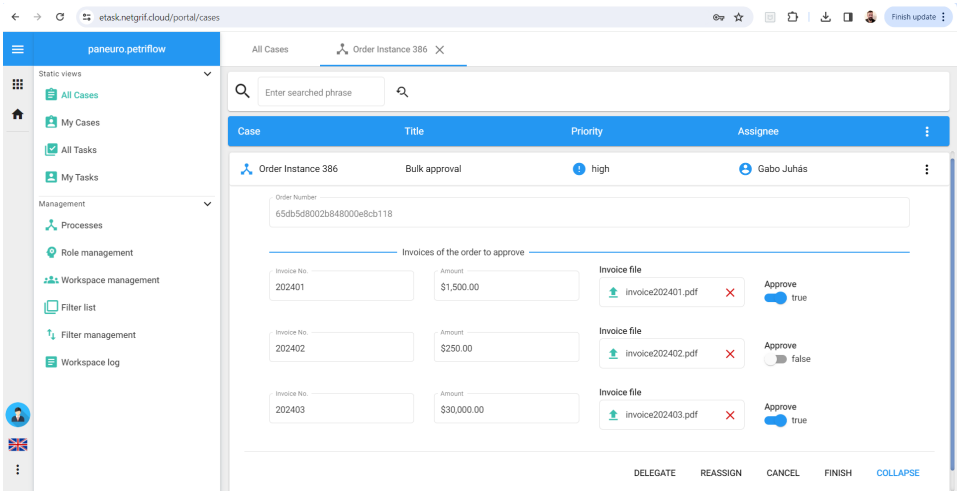

Fig. 9: Form of Bulk approval task of an order containing forms of Approval task of associated orders as subforms - example of one-to many relationship on presentation layer

5 Summary

In this paper we summarized our experience from modelling business processes with low-code language Petriflow. Our experience is that a suitable extension of Petri nets can be used not only as an implementation of the modelled business processes. Another important experience that we gained is that by modelling business objects vertically, i.e. modelling data layer, application layer and presentation layer of a business object in one artefact and then interconnecting such vertically defined objects one obtain all the advantages of the paradigm of object-centric processes, when compared with traditional approach, where data layer (data objects and their relationships) is modelled separately, application layer (workflow processes) is modelled separately and presentation layer (screens) is modelled separately and then the communication between the three layers is established.

References

[Aa09] van der Aalst, W. M. P.; van Hee, K. M.; ter Hofstede, A. H. M.; Sidorova, N.; Verbeek, H. M. W.; Voorhoeve, M.; Wynn, M. T.: Soundness of Workflow Nets with Reset Arcs. Trans. Petri Nets Other Model. Concurr. 3/, pp. 50–70, 2009.

[ADO00] van der Aalst, W. M. P.; Desel, J.; Oberweis, A., eds.: Business Process Management, Models, Techniques, and Empirical Studies, vol. 1806, Lecture Notes in Computer Science, Springer, 2000.

[AH02] van der Aalst, W. M. P.; van Hee, K. M.: Workflow Management: Models, Methods, and Systems. MIT Press, 2002.

[AJ15] Alqarni, M. A.; Janicki, R.: On Interval Process Semantics of Petri Nets with Inhibitor Arcs. In: Petri Nets. Vol. 9115. Lecture Notes in Computer Science, Springer, pp. 77–97, 2015.

[BDM11] Bergenthum, R.; Desel, J.; Mauser, S.: Workflow Nets with Roles. In: EMISA. Vol. P-190. LNI, GI, pp. 65–78, 2011.

[De05] Desel, J.: Process Modeling Using Petri Nets. In: Process-Aware Information Systems. Wiley, pp. 147–177, 2005.

[DJ01] Desel, J.; Juhás, G.: "What Is a Petri Net?" In: Unifying Petri Nets. Vol. 2128. Lecture Notes in Computer Science, Springer, pp. 1–25, 2001.

[DPW04] Desel, J.; Pernici, B.; Weske, M., eds.: Business Process Management: Second International Conference, BPM 2004, Potsdam, Germany, June 17-18, 2004. Proceedings, vol. 3080, Lecture Notes in Computer Science, Springer, 2004.

[DR15] Desel, J.; Reisig, W.: The concepts of Petri nets. Softw. Syst. Model. 14/2, pp. 669–683, 2015.

[JJP23] Juhás, G.; Juhásová, A.; Petrovič, L.: Low-Code Languages in IT Education: Integrating Theory and Practice. In: Prof. ICETA 2023 - 21st Year of International Conference on Emerging eLearning Technologies and Applications. IEEE, pp. 249–257, 2023.

[Ju19] Juhásová, A.; Juhás, G.; Molnár, L.; Ondrišová, M.; Mažári, J.; Mladoniczky, M.: IT induced innovations: Digital transformation and process automation. In: Proc. ICETA 2019 - 17th IEEE International Conference on Emerging eLearning Technologies and Applications. IEEE, pp. 322–329, 2019.

[Ju21a] Juhás, G.; Kováčik, T.; Kovář, J.; Kranec, M.; Petrovič, Ľ.: Netgrif Application Engine. In: BPM (PhD/Demos). Vol. 2973. CEUR Workshop Proceedings, CEUR-WS.org, pp. 166–170, 2021.

[Ju21b] Juhás, G.; Kováčik, T.; Kovář, J.; Kranec, M.; Petrovič, Ľ.: Petriflow language and Netgrif Application Builder. CEUR Workshop Proceedings 2973/, pp. 171–175, 2021.

[Ju22] Juhás, G.; Molnár, L.; Juhásová, A.; Ondrišová, M.; Mladoniczky, M.; Kováčik, T.: Low-code platforms and languages: the future of software development. In: Proc. 20th Anniversary of IEEE International Conference on Emerging eLearning Technologies and Applications, ICETA 2022. IEEE, pp. 286–293, 2022.

[Va78] Valk, R.: Self-Modifying Nets, a Natural Extension of Petri Nets. In: ICALP. Vol. 62. Lecture Notes in Computer Science, Springer, pp. 464–476, 1978.

[Vo02] Vogler, W.: Partial order semantics and read arcs. Theor. Comput. Sci. 286/1, pp. 33–63, 2002.