Pan-European University, Faculty of Informatics

Interim Research Report

Definition of object-centric process

of the project

Requirements and formal definition of a low-code language based on object-centric processes - LowcodeOCP

Authors:
Gabriel Juhás, Milan Mladoniczky, Juraj Mažári and Tomáš Kováčik

Contact:
gabriel.juhas@paneurouni.com, milan.mladoniczky@paneurouni.com

Funded by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No.

09I03-03-V04-00493

October 1, 2025

Version: v0.1

Abstract

This interim report summarizes the current progress of the task 2.1 of the research project LowcodeOCP resulting in the formal definition of the standalone object-centric process.

Contents

Chapter 1

Introduction

In this chapter, we discuss requirements on a low-code language for object centric processes, resulting in definition of low-code language called Petriflow, namely we discuss a definition of a process and inter-process communication. We also illustrate on several real-life use cases how the Petriflow models of business processes can directly be used as implementation when deployed in a Petriflow interpreter. Briefly, Petriflow is based on extended Petri nets (where transitions represents tasks), enriched by data variables and data forms (associated with tasks).

Petriflow is a high-level modeling language for process-driven application development [Juh+21]. Petriflow follows the programming paradigm called process-driven programming (PDP) [Juh+19]. 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 modeling (BPM) [ADO00; DPW04], event-driven programming (EDP), and relational databases (RDB).

Object-oriented programming languages brought about a distinguishing feature, encapsulation. This was a key step for the business world that needed new languages to solve the problem of growing application complexity. OOP was a concept that solved 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.

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 in 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 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:

- 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 of the task state, then a reaction can be defined by pieces of code (called actions) in the event listener. Whenever an event occurs, 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 user lists can be associated with task events, defining for each task which users are authorized to trigger events on that task. Similarly to data fields, an association of users with events is a rich relation. For example, a user authorized to trigger 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 trigger the finish event of this task and only this user is then authorized to trigger set events of editable data fields of this task and to trigger 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 [DJ01; Des05; DR15] enriched by reset arcs [Aal+09], inhibitor arcs [AJ15] and read arcs [Vog02] to define the life cycle of the Petriflow process. Places of the Petri net represent the control variables. As mentioned above, 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. The 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 a Petriflow process is given by flow of assign/get/set/finish events on tasks and data variables respecting the restrictions on events given by the underlying Petri net and permissions in roles and user lists.

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 be directly 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 [Juh+22; JJP23].

Chapter 2

Formal definition of a object centric process in low-code language Petriflow

2.1 Mathematical preliminaries

We use $\mathbb{N}$ to denote the nonnegative integers and $\mathbb{N}^+$ to denote the positive integers. Given two arbitrary sets A and B, the symbol B^A denotes the set of all functions from A to B. Given a function f from A to B and a subset C of A we write $f|_C$ to denote the restriction of f to the set C. The symbol 2^A denotes the power set of a set A. Given a set A, the symbol $|A|$ denotes the cardinality of A and the symbol id_A the identity function on the set A. We write id to denote id_A whenever A is clear from the context. The set of all multisets over a set A is denoted by $\mathbb{N}^A$. The addition of multisets over a finite set A is denoted by $+$. Given two multisets m and m' over A, $m + m'$ is defined by $\forall a \in A : (m + m')(a) = m(a) + m'(a)$. Notice that $(\mathbb{N}^A, +)$ is the free commutative monoid over A. We do not distinguish between a subset $X \subseteq A$ and its characteristic multiset m_X given by $m(x) = 1$ for each $x \in X$ and $m(x') = 0$ for each $x' \in A \setminus X$. Finally, we write as usual $\sum_{a \in A} m(a)a$ to denote the multiset m over A. Given a binary relation $R \subseteq A \times A$ over a set A, the symbol R^+ denotes the transitive closure of R and R^* the reflexive and transitive closure of R.

2.2 Petriflow process definition

Let us now define formal semantics for Petriflow processes.

Definition 1 (Petriflow process)
A Petriflow object-centric process *is a triple* $OCP = (Data, Workflow, Interface)$, *where:*

- *Data is a triple* $Data = (Variables, Types, Typing)$*, such that:*
  - $Variables$ *is a finite set of data variables*
  - $Types$ *is a set of data types such that for each data type* $V \in Types$*,* V *is a set, called values of the data type* V*. We also consider a unique element null such that for each* $V \in Types$ *element* $null \notin V$*.*
  - $Typing$ *is a function* $Typing : Variables \to Types$ *associating a data type to each data variable.*
- $Workflow$ *is a six-tuple* $Workflow = (P, T, F, C_+, C_-, W)$*, where:*
  - P *is a finite set of* places*,*
  - T *is a finite set of* tasks*, satisfying* $P \cap T = \emptyset$*,*
  - $F \subseteq (P \times T) \cup (T \times P)$ *is a* flow relation*,*
  - $C_+, C_- \subseteq P \times T$ *are positive and negative context relations,*
  - $W : F \cup C_+ \cup C_- \to \mathbb{N}^+ \cup P \cup V$ *is a weight function.*
- $Interface$ *is a function* $Interface : T \to 2^V$ *associating a subset of data variables to each task. Given a task* $t \in T$ *elements of* $Interface(t)$ *are called data fields or data refs of the task* t *and* $Interface(t)$ *itself is also called a form of task* t*.*

Definition 2 (State of Petriflow OCP) *Given a set of users denoted by* $Users$ *with element* $null \notin Users$*, then a state of a Petriflow OCP as defined in Definition 1 is a function* s *with definition domain* $P \cup V \cup T$ *such that:*

- $s|_P : P \to \mathbb{N}$ *is a function that associates a nonnegative marking with each place,*
- *for each* $v \in V$: $s(v) \in Typing(v) \cup \{null\}$*, i.e. state of each data variable is a value of the type of the variable or null,*
- *for each* $t \in T$: $s(t) = (assigned, accessibility, required, consumed)$ *where*
  - $assigned \in Users \cup \{null\}$ *determines whether a task is assigned to a user, and*
  - $accessibility$ *is a function* $accesibility : Interface(t) \to \{visible, editable, hidden\}$ *determining for each data field of the task whether it is visible, editable or hidden, and*
  - $required$ *is a function* $required : Interface(t) \to \{0, 1\}$ *determining for each data field of the task whether it is required (must have a value) or can be unfilled (without any value), and*
  - $consumed$ *is a function* $consumed : \{p \in P : (p, t) \in F\} \to \mathbb{N}$

Definition 3 (Events and roles of a Petriflow OCP) *Given a Petriflow OCP as defined in Definition 1, and a set of users denoted $Users$ we define a mapping Role and events as follows:*

- *for each $t \in T$ we define:*
  - *$Role(t)$ is a subset of set $Users$*
  - *$view_t$ denotes a view event and*
  - *$Role(view_t)$ is a subset of set $Users$*
  - *$assign_t$ is a parametrized assign event with set of parameters $Role(t)$,*
  - *for each user $u \in Role(t)$, $Role(assign_t(u))$ is a subset of set $Users$ satisfying $Role(assign_t(u_1)) = Role(assign_t(u_2))$ for any $u_1, u_2 \in Role(t)$*
  - *$cancel_t$ denotes a cancel event and*
  - *$Role(cancel_t)$ is a subset of set $Users$*
  - *$finish_t$ denotes a finish event*
  - *$Role(finish_t)$ is a subset of set $Users$*
  - *for each $v \in V$ we define event get_v and a parametrized event set_v with set of parameters $Typing(v) \cup \{null\}$*

Definition 4 (Enabled task) *Given a Petriflow OCP as defined in Definition 1, we define that a task t is enabled in state s of the Petriflow OCP $\iff$:*

- *$s(t) = (assigned, accessibility, required, consumed)$ with $assigned = null$ and*
- *for each $(p,t) \in F \cup C_+$ such that $W((p,t)) \in \mathbb{N}$ there holds $s(p) \geq W((p,t))$ and*
- *for each $(p,t) \in F \cup C_+$ such that $W((p,t)) \in P \cup V$ there holds $s(W((p,t))) \in \mathbb{N} \wedge s(p) \geq s(W((p,t)))$ and*
- *for each $(p,t) \in C_-$ such that $W((p,t)) \in \mathbb{N}$ there holds $s(p) < W((p,t))$ and*
- *for each $(p,t) \in C_-$ such that $W((p,t)) \in P \cup V$ there holds $s(W((p,t))) \in \mathbb{N} \wedge s(p) < s(W((p,t)))$.*

Definition 5 (Triggering a view event) *Given a Petriflow OCP as defined in Definition 1, and a set of users denoted $Users$ with extended mapping Role and events as defined in Definition 3, we define triggering of a view event by a user $u \in Users$ in state s of the Petriflow OCP as follows:*

- *for each $t \in T$ event $view_t$ can be triggered by user u in state $s \iff$*
  - *user $u \in Role(view_t)$ and*

  - *task t is enabled in state s or*
  - *$s(t) = (assigned, accessibility, required, consumed)$ with $assigned \neq null$.*

Definition 6 (Triggering an assign event) *Given a Petriflow OCP as defined in Definition 1, and a set of users denoted $Users$ with extended mapping Role and events as defined in Definition 3, we define triggering of an assign event by a user $u \in Users$ in state s of the Petriflow OCP as follows:*

- *for each $t \in T$ event $assign_t(x)$ with user $x \in Role(t)$ as a parameter can be triggered by user u in state $s \iff$*
  - *user $u \in Role(assign_t(x))$ and*
  - *task t is enabled in state s.*

When an assign event $assign_t(x)$ can be triggered by user u in state s with $s(t) = (assigned, accessibility, required, consumed)$ then its triggering leads to a new state s' that differs from state s as follows:

- *for each $(p,t) \in F$ such that $W((p,t)) \in \mathbb{N}$ there holds $s'(p) = s(p) - W((p,t))$ and $consumed(p) = W((p,t))$ and*
- *for each $(p,t) \in F$ such that $W((p,t)) \in P \cup V$ there holds $s'(p) = s(p) - s(W((p,t)))$ and $consumed(p) = s(W((p,t)))$ and*
- *$assigned = x$.*

Definition 7 (Triggering a cancel event) *Given a Petriflow OCP as defined in Definition 1, and a set of users denoted $Users$ with extended mapping Role and events as defined in Definition 3, we define triggering of an cancel event by a user $u \in Users$ in state s of the Petriflow OCP as follows:*

- *for each $t \in T$ event $cancel_t$ can be triggered by user u in state $s \iff$*
  - *$s(t) = (assigned, accessibility, required, consumed)$ with $assigned \neq null$ and*
  - *user $u \in Role(cancel_t)$.*

When a cancel event $cancel_t$ can be triggered by user u in state s with

$$s(t) = (assigned, accessibility, required, consumed)$$

then its triggering leads to a new state s' that differs from state s as follows:

- *for each $(p,t) \in F$ there holds $s'(p) = s(p) + consumed(p)$ and*
- *$assigned = null$.*

Definition 8 (Triggering a finish event) *Given a Petriflow OCP as defined in Definition 1, and a set of users denoted $Users$ with extended mapping $Role$ and events as defined in Definition 3, we define triggering of an finish event by a user $u \in Users$ in state s of the Petriflow OCP as follows:*

- *for each $t \in T$ event $finish_t$ can be triggered by user u in state $s \iff$*
  - *$s(t) = (assigned, accessibility, required, consumed)$ with $assigned \neq null$ and*
  - *user $u \in Role(finish_t)$ and*
  - *for each $v \in Interface(t) : required(v) = 1 \implies s(v) \neq null$ and*
  - *for each $(t,p) \in F$ such that $W((t,p)) \in P \cup V$ there holds $s(W((t,p))) \in \mathbb{N}$.*

When a finish event $finish_t$ can be triggered by user u in state s with

$$s(t) = (assigned, accessibility, required, consumed)$$

then its triggering leads to a new state s' that differs from state s as follows:

- *for each $(t,p) \in F$ such that $W((t,p)) \in \mathbb{N}$ there holds $s'(p) = s(p) + W((t,p))$ and*
- *for each $(t,p) \in F$ such that $W((t,p)) \in P \cup V$ there holds $s'(p) = s(p) + s(W((t,p)))$ and*
- *$assigned = null$.*

Definition 9 (Triggering a set event) *Given a Petriflow OCP as defined in Definition 1, and a set of users denoted $Users$ with extended mapping $Role$ and events as defined in Definition 3, we define triggering of a set event by a user $u \in Users$ in state s of the Petriflow OCP as follows:*

- *for each $v \in V$ and each $value \in Typing(v) \cup \{null\}$ event $set_v(value)$ can be triggered by user u in state $s \iff$*
  - *there exists a task $t \in T$ such that*

$$s(t) = (assigned, accessibility, required, consumed)$$

with $assigned = u$ and $v \in Interface(t)$ and $accessibility(v) = editable$.

When $set_v(value)$ can be triggered by user u in state s then its triggering leads to a new state s' that differs from state s as follows:

- *s'(v) = value.*

Definition 10 (Triggering a get event) *Given a Petriflow OCP as defined in Definition 1, and a set of users denoted $Users$ with extended mapping Role and events as defined in Definition 3, we define triggering of a get event by a user $u \in Users$ in state s of the Petriflow OCP as follows:*

- *for each $v \in V$ event get_v can be triggered by user u in state $s \iff$*
  - *there exists a task $t \in T$ such that*

$$s(t) = (assigned, accessibility, required, consumed)$$

with

* *t is enabled in s or $assigned = u$ and*
* *$v \in Interface(t)$ and*
* *$accessibility(v) \neq hidden$.*

2.3 Summary and outlook

This report contains the basic definition of a singleton object-centric process in low-code language Petriflow, including definition of the process structure consisting of data, workflow and interfaces of workflow tasks, state of the object centric process, events and roles on tasks, events on data attributes and behavior based on triggering events by a user.

For better readability we plan to include informal description of the above mentioned formal artifacts of object centric processes together with some illustrative examples to make the formal definition easier to understand. We also plan to extend the definition with actions and their behavior.

Bibliography

[ADO00] Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, eds. *Business Process Management, Models, Techniques, and Empirical Studies*. Vol. 1806. Lecture Notes in Computer Science. Springer, 2000.

[Aal+09] Wil M. P. van der Aalst et al. "Soundness of Workflow Nets with Reset Arcs". In: *Trans. Petri Nets Other Model. Concurr.* 3 (2009), pp. 50–70.

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

[BDM11] Robin Bergenthum, Jörg Desel, and Sebastian Mauser. "Workflow Nets with Roles". In: *EMISA*. Vol. P-190. LNI. GI, 2011, pp. 65–78.

[Des05] Jörg Desel. "Process Modeling Using Petri Nets". In: *Process-Aware Information Systems*. Wiley, 2005, pp. 147–177.

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

[DPW04] Jörg Desel, Barbara Pernici, and Mathias Weske, 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] Jörg Desel and Wolfgang Reisig. "The concepts of Petri nets". In: *Softw. Syst. Model.* 14.2 (2015), pp. 669–683.

[JJP23] G. Juhás, A. Juhásová, and L. Petrovič. "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, 2023, pp. 249–257.

[Juh+21] G. Juhás et al. "Petriflow language and Netgrif Application Builder". In: CEUR Workshop Proceedings 2973 (2021), pp. 171–175.

[Juh+22] G. Juhás et al. “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, 2022, pp. 286–293.

[Juh+19] A. Juhásová et al. “IT induced innovations: Digital transformation and process automation”. In: *Proc. ICETA 2019 - 17th IEEE International Conference on Emerging eLearning Technologies and Applications*. IEEE, 2019, pp. 322–329.

[Vog02] Walter Vogler. “Partial order semantics and read arcs”. In: *Theor. Comput. Sci.* 286.1 (2002), pp. 33–63.

Appendix A

Supplementary Material

For newest version of this report and other supplementary material including XML scheme of the syntax of the Petriflow OCP see petriflow.org.