Pan-European University, Faculty of Informatics

Interim Research Report

Requirements on a low-code language

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 summarises the current progress of the task 1.1 of the research project LowcodeOCP resulting in set of requirements respectively informal definition of basic entities of a low-code language based on object-centric processes.

Contents

Chapter 1

Definitions

In this chapter we provide informal definitions for basic objects, entities and modules of a low-code language, object-centric processes written in that language, and low-code platform that executes those processes.

1.1 Low-code platform

Platform is an information system that enables to create (develop) Process-oriented applications in a Builder, generate source code of the Process-oriented applications in a low-code language for object-centric processes and to deploy and run generated Process-oriented applications in Runtime environment where Users (with possibly different credentials) interact with Process-oriented applications according to Roles.

1.1.1 Process-oriented applications

Process-oriented application, shortly just an application, consists of a set of Resources and Dependencies. For sake of simplicity, we require that in a Runtime environment of a Low-code platform there is a default System application.

Examples: An application for insurance claim processing

1.1.2 Runtime environment

Environment for the execution of Object-centric processes in Process-oriented applications. It is a collection of processes on which events can be executed.

1.1.3 Resources and Dependencies

Resources and dependencies are an abstract notion to denote different types of objects that can be used in an application or by an application. Most object can serve both as

resources and dependencies. By definition of objects we will refer whether it can serve as a resource and/or a dependency.

Examples of resources and/or dependencies are Object-centric processes, instances of object-centric processes also called Cases, files, Users, Roles, modules, plugins, User interface components, templates and configurations.

1.1.4 System application

A Low-code platform provides a default application that enables rapid prototyping during the first stages of development and serves as a root of the application hierarchy.

1.1.5 Application hierarchy

Application hierarchy provides a tree structure for a logical division of applications and their contexts.

1.1.6 Builder

Builder is a special part of the platform that enables developers to create and edit Object-centric processes, manage Application hierarchy, upload and change Resources and Dependencies, design and configure user interface. Graphical modelling, drag-and-drop features, and WYSIWYG editors are used wherever possible to simplify and speed up the development process.

1.1.7 User interface components

A Low-code platform provides a component register that defines available user interface components identified by their Id. The platform provides a default set of UI components ,but also allows to add custom UI components. Custom UI components can distinguish Process-oriented applications from competition and also improve user experience. Example: number field can use components different from the default one based on what its value represents - currency field if it is a monetary value like price of a merchandise, or slider if it is a light intensity.

1.2 Object-centric processes

Object-centric process consisting of a set of Data attributes together with their life-cycle in form of a Workflow with explicit states and a set of Tasks. Access and change of the data attributes For each state, it should specify a subset of tasks that can be executed.

For each task it should specify permissions for roles to determine which events can or cannot be trigerred by users having the role.

1.2.1 Process lifecycle and versioning

Multiple versions of the process can exist in the application at the same time to support long running processes and changes in real world that occur during their lifecycle that forces the creation of new version of the process. There is also need to limit the ability to create instances of old processes, so every process needs to have state called *activated*, that enables or disables the creation of instances of that event. Activation of process can be executed directly after the new version of process is created or scheduled to happen at certain time in the future.

1.2.2 Workflow

As a workflow model, Petriflow uses place/transition Petri nets enriched by reset arcs, inhibitor arcs and read arcs. Transitions of Petri nets represent tasks of workflow models. In addition to constant values Petriflow allows to use dynamic expressions with integer return type as arc weight.

1.2.3 Cases

Instances of Object-centric processes are called cases. Each case contains a copy of the parent process, including workflow and data variables. Dynamic values of attributes of process elements are evaluated during the instantiation of the case. Static values of attributes of process elements are cloned during the instantiation of the case.

1.2.4 Tasks

Task represents an activity that Users (human or machine) performs. Tasks are associated with subset of data attributes of the process by Data references also called dataRefs. By performing a task by a user we mean that the user triggers Get task data event or Set task data event of its data references. Task can be disabled or enabled based on state of the workflow. If a task was disabled and is becoming enabled an Enable task event of the task is triggered by the system. If a task was enabled and is becoming disabled an Disable task event of the task is triggered by the system. If a task is enabled it means that it can be executed by a user. Execution of task starts by triggering Assign task event that assign a user to given task and ends by triggering Finish task event. Whether a user can assign a task to himself or a different user is determined by Role references. By triggering a View task event for disabled, enabled or assigned task the task is displayed for the user

with role given by role reference. Similarly role references are also used for finish events. If a task is assigned to a user it also can be cancelled by triggering Cancel task event.

1.2.5 Forms

Forms with different Layouts, Data references, and User interface components can be defined in the Object-centric processes. Each task can reference one form with it. A form can be referenced and reused in many tasks. Forms serve as interface for human interaction with associated data fields of the task.

Layouts

Layout defines the size, position, and alignment of data fields. Petriflow uses two types of layout, *flex* and *grid*. Flex is a one directional layout that places its items in a row or column to fill available space. Grid is a two dimensional layout that places its items in a grid with a given row and column size. The layout item can be either a data field, an empty space, or another layout of any type. A combination of flex and grid allows to design complex layouts for any use case.

1.2.6 Identifier

Each element in the Object-centric processes needs to be identified by a unique value - identifier. The identifier must be unique in respect to the elements of the process and all its ancestors through inheritance. The identifier must be a valid identifier in the platform programming language. This simplifies Actions and allows to directly reference objects and entities of the current context, e.g. data variables of the case.

1.2.7 Expression

Expressions are one-line anonymous functions that are evaluated during the runtime when needed. Expressions can be used to initialize data variable with dynamic values, serve as dynamic weight of arcs in the Workflow, and in many other situations.

Example: initialize the value of the data variable `due_date` to 30 days in the future from the current date.

1.2.8 Function

Similarly to object-oriented programming, functions are named, and can have arbitrary number of arguments and a value. Functions have a scope that determines in which context they can be called. Functions with scope `process` can be called in all events of

cases of that process. Functions with scope `application` can be called in all events of cases belonging to that application.

1.2.9 Data variables

Data variables represent all attributes of a process instance called case during its life-cycle. Data variables associated to workflow tasks define data fields and create task forms.

1.2.10 Data attributes

Each data variable has a set of attributes that further specify its purpose in the process. The core attributes that are always present in the set are: title, description, placeholder, initial value, list of validations, and default component. Attributes options and allowedNets are only available for certain fields.

1.2.11 Data references

Data variables are associated with Forms using data references. Data references has both its own Id and the Id of the referred data variable to allow multiple references to the same variable in one form. Data references allow to specify if an user needs to provide a value for the data variable using the *required* attribute. The *behaviour* attribute defines if the graphical input field is shown to the user and whether it is editable.

1.2.12 Validation

Validations are used to ensure only valid data are persisted into the database. Validation can be run on server or client side or both. Custom validations can be created and used in processes. If a validation fails (returns false) associated error message is logged (server side) and/or showed to the user (client side). Validations can have 0..N arguments to support validity based on a set of dynamic values, for example a date field value is valid if it is within 30 day range from the value of another date field.

1.2.13 Actions

Actions are anonymous functions that define reactions to all types of . In actions, events can be triggered and external functions can be called. Since events can change the state of the process, actions need to be attached to execution phase before the event, i.e. pre-phase, or after the event, i.e. post-phase.

Actions can have different context available to them depending on the type and phase of event it is attached to. For example action of Create case in the pre-phase does not have access to the data variables of the case because they have not been created yet.

1.2.14 Properties

In many real-life use cases, additional information need to be stored with the process or parts of it. Properties can be used to store them in the form of a key-value store (string:string).

1.3 Users and Roles

User management and role-based access control are integral part of the Low-code platform. Users and roles can be defined in the platform itself or they can be integrated from existing systems.

1.3.1 Users

All entities interacting with a Low-code platform are collectively called users. From the point of view of a process, it does not matter if a task is executed by human being, by robot, or by another application via integrations.

1.3.2 Groups

Groups are sets of Users used to represent organisational structure and simplify permission management. In many cases an user group has a set of Roles associated with it. This means that every user in that group also has all the associated roles.

1.3.3 Roles

Roles are sets of Users. They can be associated with sets of Events by Role references also called roleRefs. Role references allow (positive permission) or forbid (negative permission) the triggering of an associated event by its role users. Roles can be created on different levels: case, process, and application.

Case Roles

Case roles are lists of users specific for a given instance and therefore can be associated only with events of the instance. Positive permission example: only the thesis supervisor can assign student to the thesis. Negative permission example: the thesis supervisor cannot be the thesis opponent.

Process Roles

Process roles can be associated with process events and give permissions for each instance of that process. Positive permission example: only loan officers can approve mortgage

applications. Negative permission example: interns cannot assign tasks to themselves.

Application Roles

Application roles provide permissions for different management and development views of the application. Positive permission example: only administrators can assign process roles to users. Negative permission example: anonymous users cannot view internal applications.

1.3.4 Role references

Role references associates Roles with Events. For each event they define with true/false value if users of that role are allowed (true) or forbidden (false) to trigger the event.

1.4 Events

All interactions in the runtime environment are represented by events of the object type. Events of a given object class are defined by transitions of an underlying place/transition net.

1.4.1 Process events

Process events are defined by the lifecycle of Object-centric processes.

Create process

Create process event is triggered when a new process is created or uploaded in the application or if a new version of existing process is created or uploaded. Example: create process event enables to run migration scripts on existing instances of previous version of the process.

Delete process

Delete process event is triggered when an existing version of a process is deleted. Deleting process also deletes all cases of that process and triggers Delete case on those instances.

View process

The view event allows users to view detailed information of the given process and allows it to be searched.

Activate process

Activate event is triggered when the process is activated and new instances are enabled to be created.

Deactivate process

Deactivate event is triggered when the process is deactivated and new instances are forbidden to be created.

1.4.2 Case events

Case events are defined by the lifecycle of instances of Object-centric processes and are triggered by creating, deleting, and searching of process instances.

Create case

Create event is triggered when a new case of a Object-centric processes is instantiated.

Delete case

Delete event is triggered when an existing case of a Object-centric processes is deleted.

View case

View event is triggered when a case is being viewed. View event allows to restrict which users are allowed to view the case. This also means that the case will not appear in **??** searches. View event also provides immediate data fields of the case.

1.4.3 Task events

Task events are defined by the lifecycle of tasks of instances of Object-centric processes.

Assign task

Assign event marks the start of execution of a task. Assign event assigns the user to the task and change the state of the workflow by consuming tokens from input places.

Cancel task

Cancel event stops the execution of the task. Cancel event removes the assigned user and rollback the change of workflow state by returning as many tokens to input places as assign event consumed. Any changes made to data variables during the execution are not rollbacked by default.

Finish task

Finish event concludes the execution of the task. First it runs validations and checks required data fields for values and if any of them fails the finish event stops. After that the finish event removes the assigned user and change the state of the workflow by producing tokens to output places.

Enable task

Enable event is only triggered by the change of state of the workflow when the task becomes enabled. Enable event is important in cases where you need to react when part of the workflow enters a certain state from any previous state, e.g. on OR-join.

Disable task

Disable event is only triggered by the change of state of the workflow when the task becomes disabled.

View task

View event is triggered when a task is being viewed. View event allows to restrict which users are allowed to view the task. This also means that the task will not appear in **??** searches. View event also provides immediate data fields of the task.

Set task data

Set data event is triggered when attributes of one or multiple data fields are changed. This also triggers set event on affected referenced data fields.

Get task data

Get data event is triggered when the content of the task (data form) is read. This also triggers get event on all referenced data fields.

1.4.4 Data events

Data events are connected with data variables of the process and are triggered by reading values and attributes of data variables and by changing values or attributes of data variables. Difference between Set task data and Set data field, and Get task data and Get data field is that task events are only triggered on data fields of that task and data events are triggered always. This allows developers to react on changes made in specific task and for example change visibility of fields in part of the task form.

Set data field

Set data event is triggered when the value or attributes of the data variable are changed.

Get data field

Get data event is triggered when the value and attributes of the data variable is read.

1.5 Summary and outlook

In analogy to business requirement documents that provide a blue print for implementation in software projects, this report contains requirements on a low-code language formulated as an informal definition of basic entities and objects of a low-code platform, object-centric processes based on a low-code language, events, users, groups and roles. These requirements should serve as a blue print for formal mathematical definition of a next generation low-code language based on object-centric processes providing a precise formal semantics.

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.