

A web-based Modeling Tool for object-centric Business Processes

Lisa Arnold, Manfred Reichert

Institute of Databases and Information Systems, Ulm University, Ulm, Germany

Abstract

Business processes have the potential to enhance efficiency, flexibility, productivity and revenue. They can automate routine procedures and thereby reduce costs in the process. In recent years, a plethora of frameworks have been developed that facilitate the modelling of activity-centred business processes. Nevertheless, there is a paucity of frameworks that concentrate on object-centric or data-driven business processes. Furthermore, the majority of commercially available business process tools provide local applications, with only a limited number leveraging the benefits of a web-based environment. This demonstration paper presents the implementation of a web-based modelling environment that implements the object-centric business process management approach: PHILharmonicFlows. The implementation yielded a redesigned and enhanced web-based edition of the original, locally developed prototype. Moreover, the web-based framework incorporates additional features, including sophisticated verification algorithms, measurement metrics for the monitoring component, as well as a more user-friendly graphical user interface (GUI) and functions that enable the modelling of a business process in greater detail by setting constraints.

Keywords

Business process modelling tool, object-centric BPM, web-based, business process management, development, framework, graphical user interface

1. Introduction


The advent of web-based technologies has eliminated the need for different versions for different operating systems, removed installation hurdles, outsourced computing power, and required administrators to maintain the framework without inconveniencing end users. Business process modelling tools, such as the workflow management system Camunda, have recently taken advantage of these benefits as well [1]. Nevertheless, most of them focus on traditional activity-centric approaches, concentrating on the execution order of their activities. In addition to these traditional approaches, a new paradigm of object-centric business process management has emerged in recent years, exemplified by the framework PHILharmonicFlows, which focuses on business objects and their business data as they exist in real processes [2].

In the object-centric process management paradigm, a business process is described in terms of interacting business objects that correspond to real-world entities. These business objects (i.e. *user type* and *object type*) and their relations, including their cardinalities and hierarchical structure, are manifested in the Relational Process Structure (RPS) [3]. In addition,

BI-Week'24: Business Information Week, September 09–13th, 2024, Vienna, Austria

✉ lisa.arnold@uni-ulm.de (L. Arnold); manfred.reichert@uni-ulm.de (M. Reichert)

ORCID 0000-0002-2358-2571 (L. Arnold); 0000-0003-2536-4153 (M. Reichert)

 © 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

business attributes can be defined for each business object, which specify the business process. The RPS, with its business objects and object attributes, defines the holistic data model of an object-centric business process. At runtime, any number of object instances (restricted by their cardinalities) can be created by the business objects. The runtime behaviour of these business objects is defined in terms of object lifecycles [4]. In contrast to activity-centric processes, object-centric business processes typically exhibit greater flexibility, as the objects within the processes can be processed largely independent of one another [2].

In [5], the implementation of modelling objects with their relations and their lifecycle processes for object-centric business processes is presented. The latter is constructed as a locally installed software tool utilising a distributed microservice-based software architecture (original framework for short). The web-based framework presented in this paper represents a reimplementa-tion of the original framework, which has been extended to include expressions for coordination process constraints [6], permissions, and sophisticated verification. Furthermore, it incorporates measurement metrics (e.g. weights for Kalman filter [7]) for monitoring predictions. The latter minimises complexity for the modeller, thereby facilitating the creation of correct business processes without the necessity for extensive knowledge of the sophisticated object-oriented process paradigm.

In [8], the original framework is extended by a runtime engine which automatically generates form sheets based on the structure of its lifecycle processes. The lifecycle contains states that are linked to each other. In addition, the states represent the form sheets. Each state can contain any number of steps (except the end state), which represent the input fields in the form sheets. Furthermore, the aforementioned steps are based on the business attributes that have been defined in the business objects within the data model.

The remainder of this paper is structured as follows. Section 2 provides insight into the core functionality offered by the web-based, object-centric business process framework, as well as an explanation of the functionality extensions in comparison to the original framework. Section 3 presents the development of the monitoring tool, including a description of its technical architecture, with an overview of the components, frameworks and libraries employed. Section 4 concludes the paper.

2. Object-centric Business Processes

The object-centric business process modelling tool is comprised of four distinct components: the data model (RPS and business data), their respective lifecycles, the coordination process(es) and the associated permissions.

2.1. Data Model

The data model comprises two key elements: the *Relational Process Structure* (RPS) and the business attributes. Figure 1 depicts a screenshot of the Data Model module within the PHILharmoniCFlows framework. The RPS establishes the hierarchy and the relations between the business objects, as well as the cardinalities thereof. The web-based tool has been enhanced with the incorporation of more sophisticated cardinalities. The original framework permits only one-to-many relations (i.e. 1:n), whereas the web-based tool permits many-to-many relations (i.e.

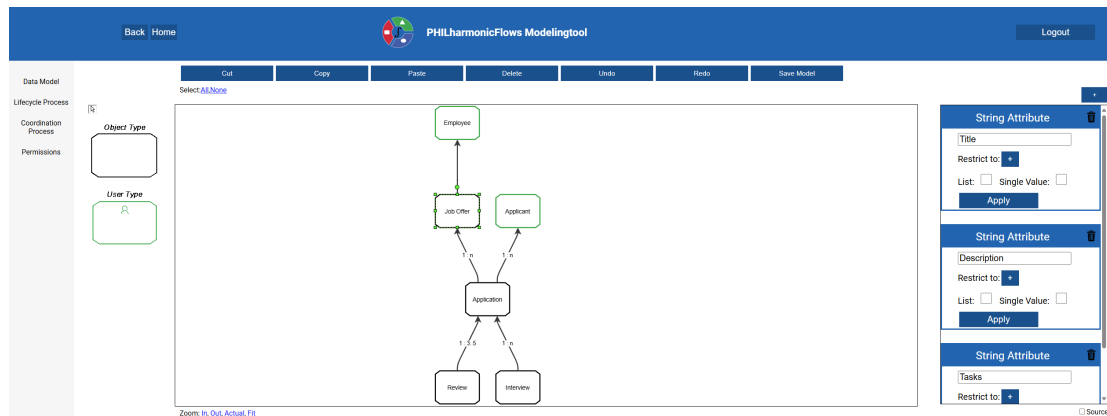


Figure 1: Data Model of a recruitment business process example with its RPS and business attributes.

n:m). Furthermore, it is crucial to emphasise that the placeholders m and n can be constrained by fixed values, both above and below (e.g. $3..5 : 2..n$).

In addition, each business object is characterised by a number of business attributes that describe the data the corresponding business objects. These business attributes are defined by a specified data type (i.e. *String*, *Number*, *Boolean*, *Date*, *File*, or *Relation*). The attributes of the data model serve as the foundation for the input fields of the auto-generated form sheets. Furthermore, the attribute types *String*, *Number*, and *Date* permit the definition of these attributes as list attributes, which permit the storage of multiple values within a single business attribute. Moreover, the web-based framework enables the setting of predefined values as input, which generates a drop-down menu at the runtime framework. In the event that invalid entries are made when creating or changing business attributes (e.g. minimum is greater than maximum restriction), error messages are generated to inform users of this issue. This verification process serves to prevent the occurrence of deadlocks at runtime and to ensure the integrity of the business process. In the original framework, there was no mechanism for verifying the specifications when creating the business attributes.

2.2. Lifecycle Processes

The runtime behaviour of each business object is manifested by its own lifecycle process. Figure 2 depicts a screenshot of the `Lifecycle` module within the `PHILharmonicFlows` framework. The runtime engine [8] automatically generates form sheets for user interactions based on the defined lifecycle structure. To elaborate, each lifecycle state represents one form sheet during the execution of the business process. In more detail, each lifecycle is initiated in one specific start state and subsequently progresses through a series of intermediate states, ultimately terminating in an empty end state, which is characterised by the absence of any steps. Each state can be further refined by a number of steps that refer to the update of business attributes and represent the input fields of a form sheet. Moreover, there are three kinds of steps: the *basic step*, which creates an input field based on business attributes; the *computation step*, which sets a specific value (e.g. a date or random number); and the *predicate step*, which models a decision based on defined expressions (e.g. 'amount < 500').

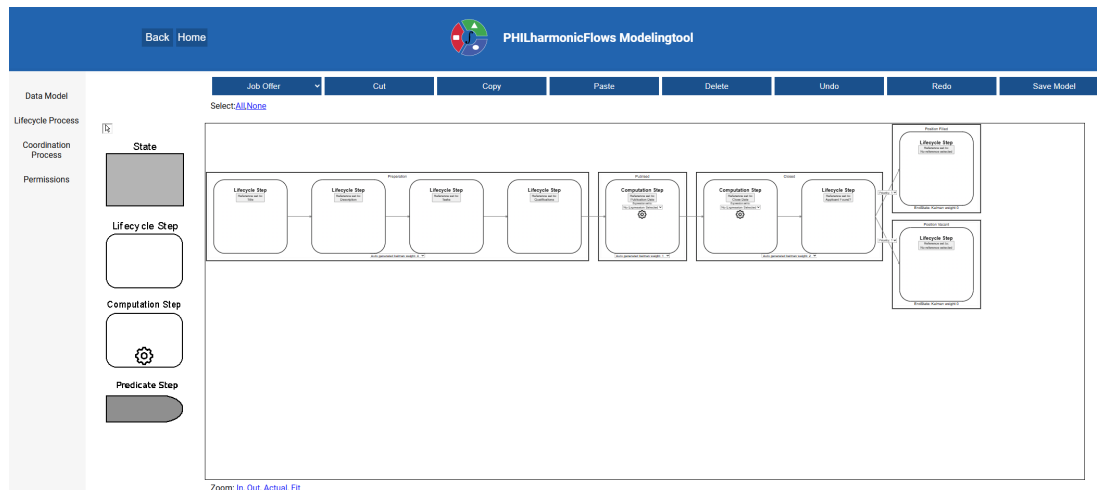


Figure 2: Lifecycle process of the Business Object *Job Offer*.

The original framework's lifecycle is extended through the incorporation of a sophisticated verification process and the automated determination of the *Kalman* weight for progress determination. Verification serves to regulate the lifecycle structure, and in the event of erroneous modelling of lifecycles, an error message is returned to the end user. The verification algorithm employs a preventative and highlighting technique. To illustrate, the lifecycle process, wherein states represent nodes and transitions represent edges, must be a directed, acyclic, and connected graph. The preventative logic block transitions to previous states in order to circumvent the formation of cycles or loops. In order to facilitate comprehension by the modeller, impermissible states are indicated by a red highlight when a transition is drawn, whereas permissible states are indicated by a green highlight. In the event of an error message being displayed, the modeller is able to click on it, which will result in all errors of this error message being highlighted in red within the process model. This technique enables the avoidance of lengthy error-finding processes, particularly in the case of large process models. To illustrate, a multitude of start states exists. Upon clicking on the error message, all start states are indicated by a red highlight. Moreover, the *Kalman* weight for the monitoring tool has been incorporated. The *Kalman* weight is a value between 1 and 5 that is automatically determined based on the number and kind of steps within a state. The *Kalman* filter is capable of predicting the progress of a single lifecycle instance based on the aforementioned *Kalman* weights, thereby obviating the necessity for an event log. In addition, the *Kalman* weight can be set by a modeller manually when the estimated effort is greater or less than the automatically determined one in order to achieve better results in determining progress [7].

2.3. Coordination Processes

A *coordination process* controls the interactions between the lifecycles of multiple objects and defines the sequence of states between multiple lifecycle states. Thereby, a *coordination step* is referred to a state of a lifecycle process. In general, a coordination process is defined from the perspective of one business object. In other words, the lifecycle of one object is extended

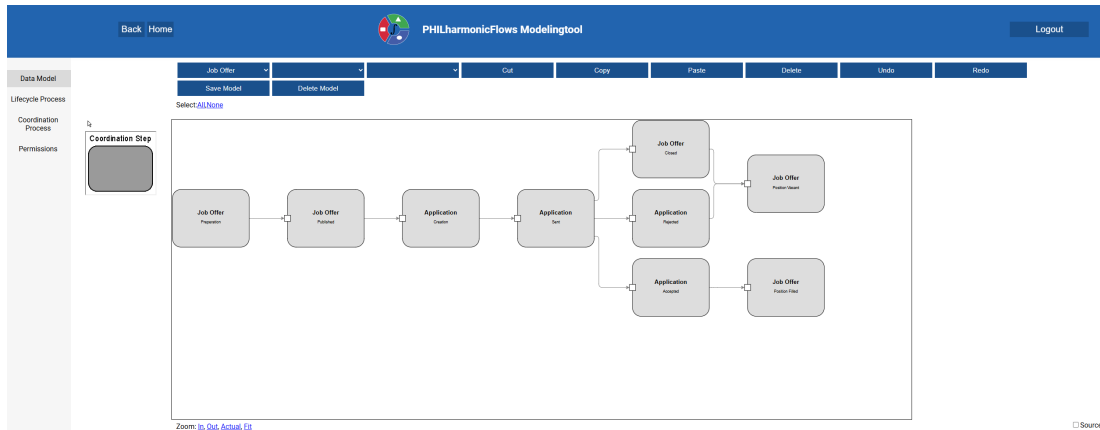


Figure 3: Coordination process of the Business Objects *Job Offer* and *Application*.

with the lifecycle states of other objects to represent their correlations and interactions. More specifically, a coordination process can be viewed as a graph where the vertices represent the coordination steps and the edges represent the *coordination transitions*. The coordination process graph is a directed, acyclic and connected graph that excludes backward transitions or loops to previous coordination steps. Otherwise, cyclic dependencies and thus deadlocks are possible. Therefore, the acyclicity of coordination processes is not a limitation of expressiveness, but a necessity for correctness [6].

In contrast to the original framework, the web-based version is extended by a sophisticated verification algorithm to identify, for example, cycles that span multiple coordination processes and are therefore difficult for a modeller to detect. Figure ?? depicts a screenshot of the Coordination Process module within the PHILharmonicFlows framework. The verification algorithm employed an active mechanism to prevent erroneous modelling of the coordination process, whereby any attempted modifications were blocked and the invalid targets (i.e. coordination transitions and previous ports or coordination steps) highlighted in red. In addition, a passive mechanism was utilised to examine the coordination process in nine distinct error cases, returning error messages to the modeller in the event of any issues.

Furthermore, the web-based framework is augmented with the capacity to impose constraints for each coordination transition. These constraints can be leveraged to facilitate the execution of a business process in a more targeted and precise manner. To illustrate, in the case of a research conference review process, a paper will be deemed appropriate for acceptance when at least 50% of the three to five assessors have determined that the paper meets the requisite standards for acceptance. The web-based PHILharmonicFlows framework allows the modeller to define constraints of this nature.

2.4. Permissions

In the original framework, the permissions are only listed in a table on the *Permissions* tab and cannot be modified there. To illustrate, the permissions for business attributes are established in an additional tab within the data model, while the permissions for lifecycle states are configured in a drop-down menu on the button for the states. When defining permissions for the first time

in the original framework, many modellers encounter difficulties in locating the appropriate setting for the permissions. Consequently, the configuration of permissions can now be carried out directly on the Permissions module as depicted in Figure 4. In detail, permissions can be defined for each business object depending on the individual users or user groups (i.e. user type). Furthermore, the ability to assign execution rights to each user for each state in the lifecycle (i.e. the activation of the 'Next' button on the form sheets at runtime) has been incorporated. In addition, the rights to read, write, or none can be specified for each user with respect to each business attribute.

	Applicant	Employee
Instantiation	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Deletion	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Preparation		
Execution	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Title	None	Write
Description	None	Write
Tasks	None	Write
Qualifications	None	Write
Publication Date	None	None
Close Date	None	None
Applicant Found?	None	None
Publised		
Execution	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Title	Read	Read
Description	Read	Read
Tasks	Read	Read

Figure 4: Permissions of the Business Object *Job Offer*.

3. Structure of the Modelling Tool

The web-based implementation of PHILharmonicFlows was developed primarily in *TypeScript*. The **front end** of the application has been constructed using the *NextJS* framework. The framework employs the *React* library for the component-based development of reusable front end elements. The graphs are created and manipulated using an extension of the *mxGraph* JavaScript library, namely *ts-mxGraph*. This extension extends the *mx-Graph* library with types, thereby making it *TypeScript* compatible. *mxGraph* is a client-side library for creating and modelling diagrams and graphs that works on all common browsers without any additional add-ons. The front end of the web-based PHILharmonicFlows framework is constituted by a set of pages, components, and assets. A page may contain one or more *React* elements, which in turn may contain other elements, components, or simple *HTML* elements. The pages delineate the content that will be rendered in the user's browser while they interact with the application. Components contain reusable code, which may be employed in a variety of contexts. For instance, the editor component is utilized in the data model editor, the lifecycle process editor, and the coordination process editor. Assets contain the styling of the application in *Cascading*

Style Sheets (CSS) files. The front end is in communication with the controllers of the back end and the editor services, receiving data that is necessary for its rendering or sending data from the front end to the database.

The application's **back end** is based on the *Node.js* framework *NestJs*, which is designed for the development of scalable server-side applications. Upon deployment of the application in a Docker container, a *NestJs* web server is initiated, facilitating the transfer of requisite data to the front end. The back end is constituted by modules, controllers and editors. The controllers represent the primary conduit for communication between the front end and the back end. The transfer of data is accomplished by transmitting a request from the front end to a designated route, which is made available by a controller. The controller will then transfer the request and its associated data to a service. This service will then manipulate the request data, if necessary, and execute a database call. Subsequently, a response will be transmitted to the request's originator. In certain instances, the front end may engage in direct communication with the editor or parser services. In such instances, data that was previously retrieved from the database via the front end is injected into an instance of an editor- or parser service, thereby enabling the service to be initialised correctly and process the data. Infrequently, the editor- or parser services are also required to communicate with the controllers in order to retrieve data from the database, which is necessary for the service functions to be executed correctly.

The data of PHILharmonicFlows was managed using the *MongoDB* **database management system** (i.e. *NoSQL* := **Not only SQL**). This is a non-relational database that employs the use of documents structured in a JSON-like format. Regardless, it is possible to utilise logical references between disparate stored documents, thereby representing the relations between them. Furthermore, it offers a multitude of query and aggregation functions, which result in enhanced performance compared to relational databases. Furthermore, *MongoDB* enables straightforward vertical and horizontal scaling due to its non-relational architecture [9, 10]. The front end is responsible for communicating with the *MongoDB* database in order to retrieve the necessary data and present it to the user in an appropriate format. As soon as the user makes any modifications to the modelled process, these changes are sent to the back end, where they are then reflected in the database.

The PHILharmonicFlows application, inclusive of both its front-end and back-end components, as well as the database, is executed within a dedicated **docker** container to ensure portability and scalability. Each container is configured to expose a port, thereby facilitating external access. This ensures that the user is able to interact with the application by sending requests to a designated controller endpoint. Typically, user requests are initiated through the graphical user interface (GUI), which is defined by the front end.

4. Summary and Outlook

This paper presents the redesigned and enhanced web-based framework of the object-centric business process PHILharmonicFlows, originally developed at the local level. The web-based framework eliminates the challenges associated with the installation and management of the application, as well as offering the outsourcing of computing resources. Moreover, the web-based framework incorporates additional features, including sophisticated verification algorithms,

measurement metrics for the monitoring component, as well as the setting of permissions in a user-friendly manner and the possibility of defining expressions for coordination process constraints. The web-based PHILharmonicFlows is currently in a state of development. The further work is focused on enhancing the user-friendliness and user interface of the lifecycle processes. This involves the configuration of the states and steps, which is a highly intricate matter. In addition, further work focused on the validation of expressions defined in the coordination process constraints, the implementation of an export and import function, and the development of additional measurement metrics for the predictions generated by the monitoring tool.

Acknowledgments

This work is part of the ProcMape project, funded by the KMU Innovativ Program of the Federal Ministry of Education and Research, Germany (F.No. 01IS23045B).

References

- [1] B. Ruecker, Practical Process Automation, " O'Reilly Media, Inc.", 2021.
- [2] V. Künzle, M. Reichert, Philharmonicflows: towards a framework for object-aware process management, *Journal of Software Maintenance and Evolution: Research and Practice* 23 (2011) 205–244.
- [3] S. Steinau, K. Andrews, M. Reichert, The relational process structure, in: *Int. Conf. on Advanced Information Systems Engineering*, Springer, 2018, pp. 53–67.
- [4] S. Steinau, K. Andrews, M. Reichert, Executing lifecycle processes in object-aware process management, in: *Int. Symp. on Data-Driven Process Discovery and Analysis*, Springer, 2017, pp. 25–44.
- [5] S. Steinau, K. Andrews, M. Reichert, A modeling tool for philharmonicflows objects and lifecycle processes (2017).
- [6] S. Steinau, K. Andrews, M. Reichert, Coordinating large distributed relational process structures, *Software and Systems Modeling* 20 (2021) 1403–1435.
- [7] L. Arnold, M. Breitmayer, M. Reichert, A one-dimensional kalman filter for real-time progress prediction in object lifecycle processes, in: *2021 IEEE 25th International Enterprise Distributed Object Computing Workshop (EDOCW)*, IEEE, 2021, pp. 176–185.
- [8] K. Andrews, S. Steinau, M. Reichert, A tool for supporting ad-hoc changes to object-aware processes, in: *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*, IEEE, 2018, pp. 220–223.
- [9] A. Nayak, A. Poriya, D. Poojary, Type of nosql databases and its comparison with relational databases, *International Journal of Applied Information Systems* 5 (2013) 16–19.
- [10] S. G. G. Sahib, A review of non relational databases their types advantages and disadvantages, *International Journal of Engineering &Technology* 2 (2013).