

# A Tool for Supporting Ad-Hoc Changes to Object-Aware Processes

Kevin Andrews, Sebastian Steinau, and Manfred Reichert  
Institute of Databases and Information Systems  
Ulm University, Germany  
Email: {firstname.lastname}@uni-ulm.de

**Abstract**—Process management systems are often criticized for not being flexible enough, as they restrict the actions of users to those defined in a process model. When unforeseen events occur during process execution, deviation from the actions the process model permits may become necessary. Such ad-hoc changes to process execution are not widely supported by process management systems as they pose significant challenges. This work presents a new prototypical addition to the PHILharmonicFlows process engine that allows for ad-hoc changes to processes following the object-aware process support paradigm. We demonstrate the extensions to the preexisting PHILharmonicFlows modeling and runtime user interfaces that enable users to change the underlying process models of process instances they are executing. The demonstration is intended to not only show off the ad-hoc change capabilities in the context of object-aware process management, but also inspire other researchers to employ similar ideas in other process support paradigms.

## I. INTRODUCTION & FUNDAMENTALS

Most business process management systems rely on some form of process model that describes the actions that have to be completed as part of the execution of a process instance. Furthermore, the process models describe the choices and alternate paths that exist in a process. However, in some cases, deviation from the course of actions defined in the process model may become necessary. This can be due to exceptional events such as errors, oversights in the process model, or deviations from the process model that are only necessary for a few instances [1] [2]. An example could be including an additional step to e.g. enforce the “four-eyes principle” in small number of process instances to judge its effect.

This paper demonstrates the toolset we created for supporting ad-hoc changes to process instances for the object-aware process paradigm. The object-aware paradigm differs from the common activity-centric paradigm and is more comparable to the artifact-centric or case handling/management paradigms [3] [4] [5]. While an activity-centric process model focuses on the process activities for users and systems, these data-centric paradigms shift the focus to the data and the lifecycle of said data during the execution of a process instance [6].

In essence, a simple example of this contrast can be seen when examining a recruitment process. An activity-centric process model for such a process contains tasks such as *Publish Job Offer*, *Wait for Applications*, *Review Applications*, *Inform Applicants of Acceptance/Rejection*. An object-aware model of the same process, however, defines the so-called *objects*, such as *Job Offer*, *Review*, or *Application*, that exist

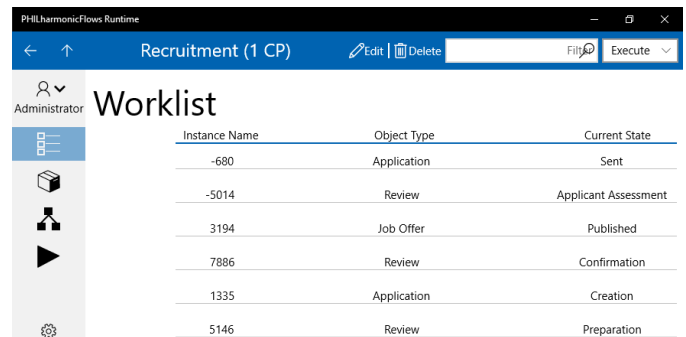
in the process and which attributes (i.e. data elements) they consist of. Furthermore, each of the objects also contains a *lifecycle process* which describes the various *states* an object may enter during its existence in a process instance. An object-aware process is always *data-driven*, i.e., the transitioning of an object from one *state* to another depends on the presence of values for the attributes the object consists of.

Consider the *Job Offer* object as an example. The first two states described by the lifecycle process of *Job Offer* could be *Created* and *Published*. Assuming that *Job Offer* has the attributes *Job Description* and *Salary*, the lifecycle process could dictate that a *Job Offer* object has to have these attributes provided by a user in order for the state to change from *Created* to *Published*. *States* can be used to add restrictions facilitating coordination between different objects, such as “*Job Offer* objects must be in the *Published* state, for corresponding *Application* objects to be created”.

After this short introduction to the fundamentals of object-aware processes, Section II presents our tooling support for ad-hoc changes to such processes at runtime. Finally, Section III gives a summary and an outlook on our future work.

## II. TOOLING SUPPORT FOR AD-HOC CHANGES

In order to enable ad-hoc changes to object-aware processes, we extended our existing object-aware process engine, PHILharmonicFlows [7] [6] [8]. The existing runtime environment takes the conceptual ideas behind object-aware process management, most of which are omitted from this work, and presents the tasks derived from the object-aware process model in the simple fashion of worklists (cf. Fig. 1).



Instance Name	Object Type	Current State
-680	Application	Sent
-5014	Review	Applicant Assessment
3194	Job Offer	Published
7886	Review	Confirmation
1335	Application	Creation
5146	Review	Preparation

Fig. 1. Worklist for Recruitment Process

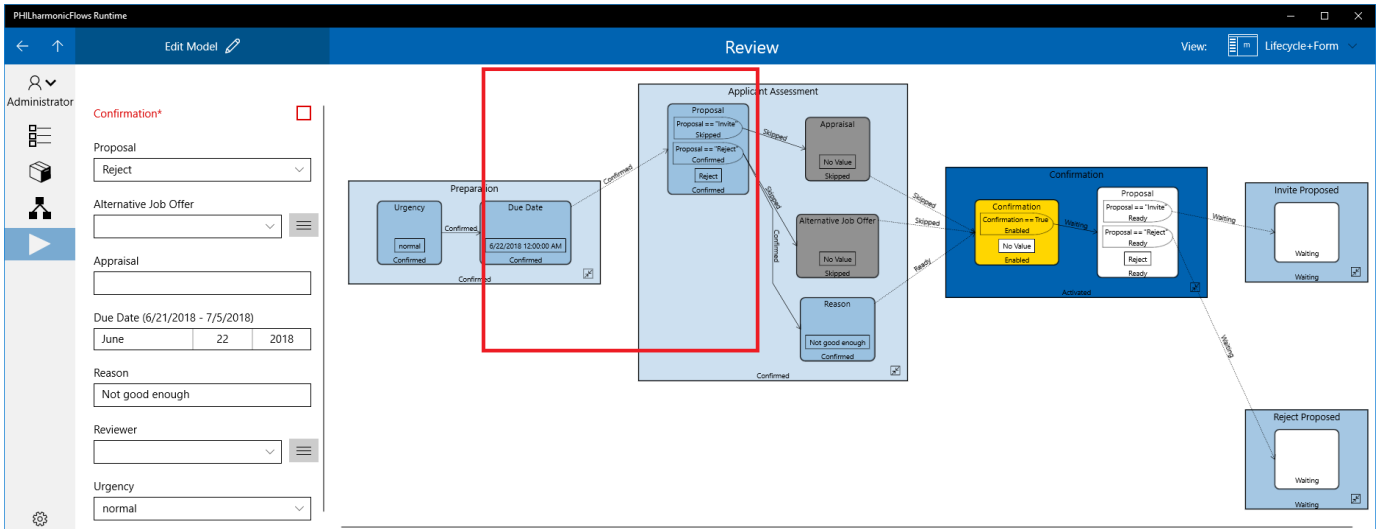


Fig. 2. Review#7886 Object without Ad-Hoc Changes (The red Box indicates where the Ad-Hoc Change from Fig. 3 will occur)

These worklists are generated dynamically based on a multitude of factors and represent the tasks a user may complete concerning the various objects that exist at a given point in time during process execution. For instance, the worklist displayed in Fig. 1 provides the user with the information that he currently has a number of tasks he may complete, e.g. on the *Review* object with the ID 7886.

Clicking on the *Review* object with the ID 7886 in the worklist reveals the combined view (cf. Fig. 2) on its *lifecycle process* (right) as well as the auto-generated *form* for user interaction with the object (left). As discussed briefly in Section I, each object that exists in an object-aware process at run-time consists of attributes, such as those visible in the form in Fig. 2, e.g. *Proposal* (a string) or *Confirmation* (a boolean). Furthermore, the values of these attributes influence the paths taken in the lifecycle process of the object, an example of which can be seen in the transitions exiting the step representing the *Proposal*, with the concrete outcome and subsequent steps being determined by the value of *Proposal*, either “Reject” or “Invite”. Due to their data-driven nature, lifecycle processes, and the objects they are attached to, only change states when all attributes referenced in one state are provided with a value. For the states of the *Review* object visible in Fig. 2, i.e. *Preparation*, *Applicant Assessment*, *Confirmation*, and *Invite/Reject Proposed*, the lifecycle process model gives a clear indication of which attributes have to have values assigned for the object to pass from one state to another. For instance, to advance from *Preparation* to *Applicant Assessment*, the attributes *Urgency* and *Due Date* have to have assigned values. This information can be used to generate forms, such as the one depicted in Fig. 2 (left).

In most object-aware process models, the various states must be completed by different users. Currently, the *Review* object with the ID 7886 is in a state called *Confirmation*, which is modeled to allow a personnel officer to confirm the assessment of the applicant completed in the prior state by someone

else. If the personnel officer provides data for the *Confirmation* attribute by clicking the *Confirmation* checkbox (the generated input field for this boolean attribute) the lifecycle process will advance to the final state *Reject Proposed* automatically.

At this point we would like to demonstrate the capabilities for ad-hoc changes which we implemented into the runtime of PHILharmonicFlows. In the following example on an ad-hoc change supported by our tooling, the decision is made to give a manager the opportunity to add a textual comment to each review before the applicant assessment is conducted. As it is unclear whether this change to the recruitment process will improve the review quality significantly or just increase turnaround times, the change is not incorporated into the base process model, but instead only into already existing *Review* objects (such as *Review#7886*) as an ad-hoc change.

To this end, our process execution UI has a new *Edit Model* button which opens the PHILharmonicFlows process modeling tool in a special ad-hoc mode. This allows for the insertion of a new state, *Manager Review* and a new attribute and corresponding step *Comment* in this new state (cf. Fig. 3).

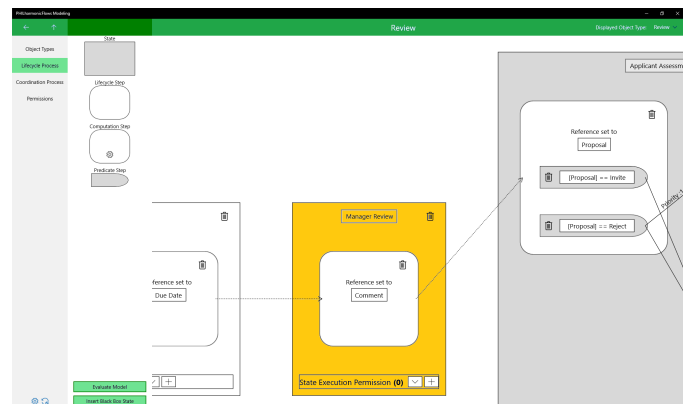


Fig. 3. Modeling Tool with Ad-Hoc changed Lifecycle (cf Fig. 2, red Box)

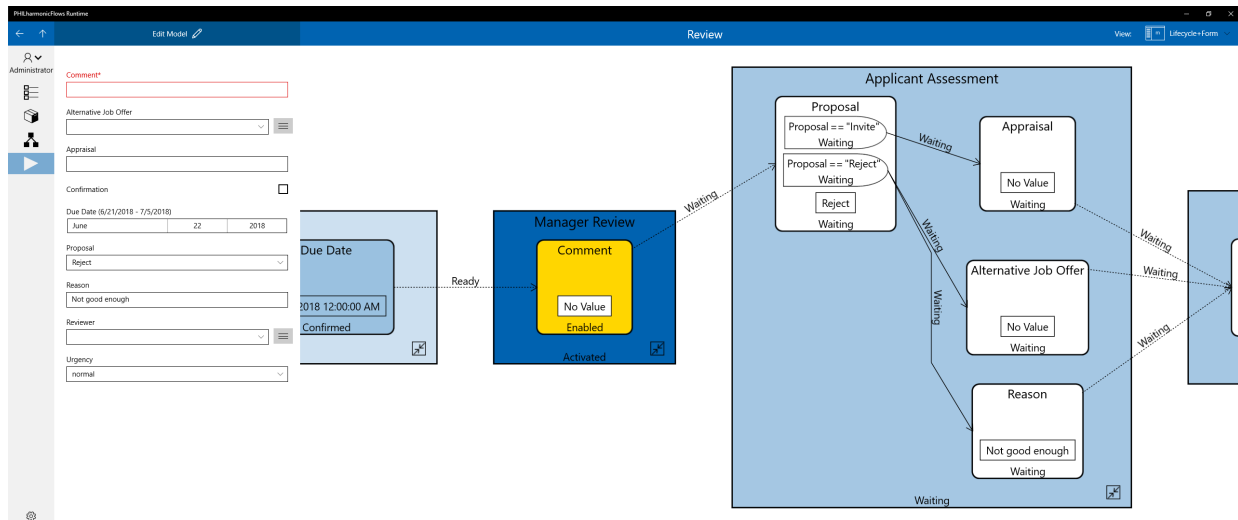


Fig. 4. Review#7886 Object with Ad-Hoc added State “Manager Review”

The aforementioned “special ad-hoc mode” for the modeling environment allows for ad-hoc changes to the attributes and lifecycle process of either one individual object or all objects of the same type that exist in a process instance. From a user perspective this procedure is completely transparent, as the user is simply editing an object-aware process model, as he would when initially modeling the process. The underlying technology, however, is far more complex. There are numerous challenges to be overcome when applying ad-hoc changes to an object, such as where to persist the changed model, which might be different for every individual object existing in a process instance. Furthermore, it must be ensured that while a user is applying ad-hoc changes to an object, i.e., actively editing it with the modeling tool, the same rigorous model correctness criteria can be applied as when the process models are initially created. To this end, all editing must be completed on a temporary copy of the object in question. This allows changes to be evaluated for correctness before propagating them to back to the “live” original object en bloc.

We overcame these challenges by employing a distributed architecture for our process execution engine. In essence, each of the objects that exists in a PHILharmonicFlows process is a *microservice* [8]. These microservices are organized using actor model theory, meaning that each microservice has its own persistent storage, a logical “thread”, and the possibility of communicating with other microservices via messages [9]. We leverage this to give each and every object a private model of itself (including all attributes and the lifecycle process) that only exists within the microservice representing that object. This allows different objects of the same type, e.g., multiple *Review* objects, to have distinct lifecycle process models. As the models of individual objects are very small as they do not need to capture the complexity of the entire process individually, this is not a concern from a memory perspective. Furthermore, this allows us to quickly create temporary editable copies of objects so the originals remain

unchanged until the user is satisfied with his ad-hoc changes and they may be propagated to the live system.

This propagation of the changes is the main challenge that we had to solve. It is not possible to delete the existing object and replace it with the copy being edited, as the object may have many inter-dependencies with other objects. However, as object-aware processes are fully data-driven, we can simply swap out the lifecycle process model in the original object with the ad-hoc changed lifecycle model from the temporary copy. Our extensions to the existing PHILharmonicFlows engine ensure that in this event the lifecycle process is re-executed from the beginning, resulting in an identical execution as the data values of the object attributes are unchanged.

This offers a benefit over other instance migration strategies [10], [2]. As *Review#7886* was executed past the point where the new state was inserted, the change would either not have had any effect or been considered an error in most migration strategies. Through this data-driven model, however, the re-execution of the object with the ad-hoc changed lifecycle model caused it to pause in the new state, awaiting a value for the now mandatory *Comment* attribute (cf. Fig. 4). This is also reflected in the worklist, which can be seen in Fig. 5.

Instance Name	Object Type	Current State
-680	Application	Sent
-5014	Review	Manager Review
3194	Job Offer	Published
7886	Review	Manager Review
1335	Application	Creation
5146	Review	Preparation

Fig. 5. Worklist after Changes

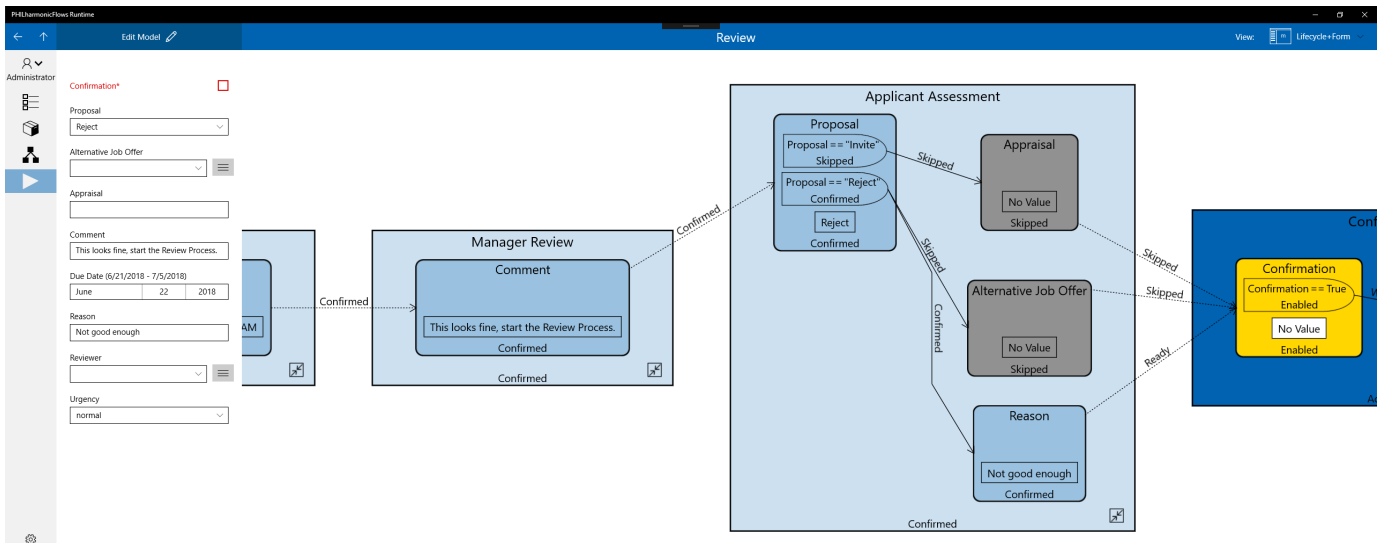


Fig. 6. Review#7886 after Comment is provided

Furthermore, the worklist shown in Fig. 5 also highlights the application of the change to other running instances of the *Review* object, as all instances that were previously in states past the point where state *Manager Review* was inserted (cf. 2 for an overview) are now in state *Manager Review*. Our tooling offers users a choice of whether to only apply ad-hoc changes to one object, multiple objects, or even all objects of a certain type (e.g. *Review*).

Finally, it is important to highlight why forcing objects that have already progressed past a certain point in their lifecycles to return to a previous (and potentially new) state is necessary when new attributes, such as *Comment*, are required for execution after an ad-hoc change. If this were not the case, ad-hoc changed objects could be inconsistent. Take for example the now ad-hoc changed lifecycle process of *Review#7886*, as depicted in Fig. 6. It is in state *Confirmation*, just as before the ad-hoc change (cf. Fig. 2). This is due to the fact that a value for attribute *Comment* was set and the lifecycle progressed to state *Confirmation* as expected, as the values for the attributes *Proposal* and *Reason* were supplied before the ad-hoc change. However, if the lifecycle had not been re-executed from the beginning, attribute *Comment* would never have been marked as required, allowing the object to be in state *Confirmation* without having all attribute values set in accordance with its lifecycle process, which could lead to various problems.

### III. SUMMARY AND OUTLOOK

This paper gave a short demonstration of a simple scenario for which our tooling is capable of supporting ad-hoc changes to object-aware processes at runtime. We utilize a distributed microservice-based software architecture in conjunction with the intrinsically loosely coupled process structure of object-aware processes to allow for ad-hoc changes to the lifecycle processes of individual objects at run-time. While the exact conceptual and technical details are too lengthy for this short

publication, the user perspective on the procedure, i.e., opening a running object instance in the runtime environment, loading the underlying lifecycle model into the modeling environment, editing it ad-hoc and propagating the changes back to the object instance in the runtime, are ideal for a demonstration.

While this is currently still a prototypical extension to the PHILharmonicFlows implementation of object-aware processes, it is very much functional and tested for some scenarios. Clearly, even though we present a working implementation, this is very academic work and real-world applicability is a valid concern. Studies and focus groups will be the topic of future work, determining which features from this prototype would actually be useful in real-world scenarios.

### REFERENCES

- [1] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features," *DKE*, vol. 66, no. 3, pp. 438–466, 2008.
- [2] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. van der Aalst, "Process flexibility: A survey of contemporary approaches," in *Advances in enterprise engineering I*. Springer, 2008, pp. 16–30.
- [3] D. Cohn and R. Hull, "Business artifacts: A data-centric approach to modeling business operations and processes," *IEEE TCDE*, vol. 32, no. 3, pp. 3–9, 2009.
- [4] M. Marin, R. Hull, and R. Vaculín, "Data centric BPM and the emerging case management standard: A short survey," in *Proc BPM*, 2012, pp. 24–30.
- [5] W. M. P. Van der Aalst, M. Weske, and D. Grünbauer, "Case handling: a new paradigm for business process support," *DKE*, vol. 53, no. 2, pp. 129–162, 2005.
- [6] V. Künzle and M. Reichert, "PHILharmonicFlows: towards a framework for object-aware process management," *JSME*, vol. 23, no. 4, pp. 205–244, 2011.
- [7] S. Steinau, K. Andrews, and M. Reichert, "A modeling tool for PHILharmonicFlows objects and lifecycle processes," in *Proc BPM*, 2017.
- [8] K. Andrews, S. Steinau, and M. Reichert, "Towards hyperscale process management," in *Proc EMISA*, 2017, pp. 148–152.
- [9] G. Agha and C. Hewitt, "Concurrent programming using actors," in *Readings in Distributed Artificial Intelligence*. Elsevier, 1988, pp. 398–407.
- [10] M. Reichert and B. Weber, *Enabling flexibility in process-aware information systems: challenges, methods, technologies*. Springer Science & Business Media, 2012.