A Tool for Supporting Object-Aware Processes

Carolina Ming Chiao, Vera Künzle, Kevin Andrews, Manfred Reichert
Institute of Databases and Information Systems
University of Ulm, Germany
Email: {carolina.chiao, vera.kuenzle, kevin.andrews, manfred.reichert}@uni-ulm.de

***Abstract*—Although the popularity of activity-centric process management systems (PrMS) has increased during the last decade, there still exist business processes that cannot be adequately supported by these PrMS. A common characteristic of these processes, which is neglected by current activity-centric PrMS, is their need for *object-awareness*; i.e., the explicit processing of business data and business objects respectively. In the PHILharmonicFlows project, characteristic properties of object-aware processes were identified and an advanced framework for their proper support was designed. In this paper, we present a proof-of-concept prototype implementing some of the fundamental concepts of the PHILharmonicFlows framework. Overall, this initiative will result in a new generation of process management technology.**

I. Introduction

The increasing economic pressure obliges enterprises to adopt process management systems (PrMS) with the aim to properly manage and automate their business processes. Traditional PrMS are *activity-centric* [13]; i.e., the business processes to be supported are defined in terms of "black-box" activities as well as the control flow elements expressing the orders and constraints for executing these activities. However, activity-centric PrMS treat business objects only as second-class citizens [2], [5], which are usually stored in external databases; i.e., outside the control of the activity-centric PrMS. Moreover, the strict *separation of concerns* realized by traditional PrMS does not allow providing an integrated view to the end-user; i.e., business data, business functions and business processes are managed by different kinds of systems, making it impossible for users to access required context information during process execution [2]. Instead, respective information can only be accessed when invoking external application systems implementing the activities.

Despite their widespread popularity, there still exist numerous processes that cannot be properly supported by traditional PrMS. These processes can be characterized as *knowledge-intensive*, relying on user decisions being characterized as *semi-structured* or *unstructured* [14]. Hence, they cannot be "straight-jacketed" into a set of activities [2]. Recent works [1]–[3], [5], [10]–[12], [15] confirmed that the limitations of existing activity-centric PrMS can be traced back to the lacking integration of processes, data, and users.

In the PHILharmonicFlows[1] project, characteristic processes from a variety of domains were analyzed [4], [6], [9]. In the respective studies, we contrasted different application scenarios with a set of properties, which were extracted from a systematic literature study. This way, we showed that the properties are *related to each other* and their support is required by a variety of processes from different application domains; i.e., *generalization* is possible. Common to all these processes is their need for *object-awareness*; i.e., business processes and business objects must not be treated independently from each other. In general, *object-aware processes* present three major characteristics. First, they can be based on *two levels of granularity*. On one hand, the *behavior* of individual object instances needs to be considered during process execution; on the other, the *interactions* among different object instances must be taken into account as well. Second, the execution of these processes is *data-driven*; i.e., the progress of a process depends on available object instances and the values of their attributes. Third, *flexible activity execution* is crucial. In particular, activities do not have to coincide with particular process steps. In this paper, we officially present for the first time a proof-of-concept prototype implementing the concepts of the PHILharmonicFlows framework. This prototype comprises both a build-time and a run-time environment, which enables the modeling, execution and monitoring of object-aware processes.

Section II summarizes the basic concepts of the PHILharmonicFlows framework, whereas Section III presents the architecture and features of the implemented proof-of-concept prototype. Section IV concludes the paper.

II. PHILharmonicFlows Framework

The PHILharmonicFlows framework enforces a well-defined modeling methodology governing the object-centric specification of business processes based on a well-defined formal semantics [7], [9]. More precisely, the framework enforces the definition of processes at two levels of granularity. While *object behavior* is defined at *micro process* level, *object interactions* are captured at *macro process* level. Due to the lack of space, we only give an overview of PHILharmonicFlows in this paper. For more details, please refer to [7], [9]. For related work, please refer to [8].

As a fundamental prerequisite, first of all, object types and their relations need to be captured in a data model (cf. Fig. 1a). Then, for each *object type*, a corresponding *micro process type* has to be specified (cf. Fig. 1b). The latter defines the behavior of related object instances, and consists of a set of *micro steps* as well as the *transitions* between them. In turn, each micro step is associated with an *object type attribute*. Further, micro steps are grouped in object *states*. At run-time, for each object instance a corresponding micro process instance is created. A micro process instance being in a particular state may only proceed if specific values are assigned to the object instance attributes associated with this state; i.e., a *data-driven process execution* is provided. In turn, *optional data access* is enabled asynchronously to micro process execution

[1]Process, Humans and Information Linkage for harmonic Business Flows http://www.uni-ulm.de/en/in/dbis/research/projects/philharmonic-flows.html

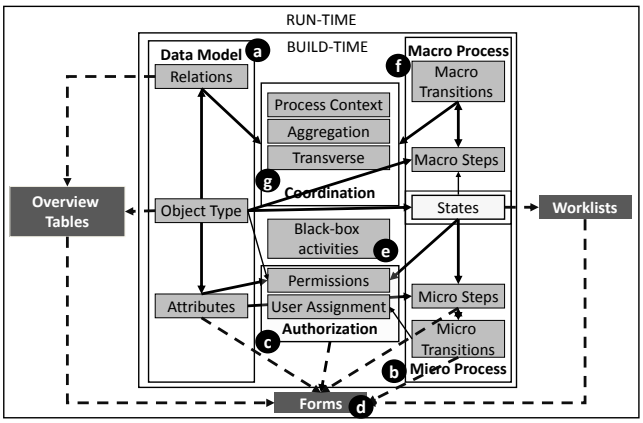

Fig. 1. The PHILharmonicFlows framework

based on the permissions granted for reading or writing object attributes. In particular, access rights for an object instance may also depend on the progress of the corresponding micro process instance. For this purpose, the framework maintains an *authorization table* assigning data permissions to user roles which may additionally depend on the respective state of the micro process type (cf. Fig. 1c). Based on this authorization table, PHILharmonicFlows automatically generates *user forms* at run-time. Which input fields are displayed to the respective user depends on the permissions he has in the currently activated state (cf. Fig. 1d). Additionally, PHILharmonicFlows allows for the integration of *black-box activities*, enabling the realization of more complex business functions for which a specific implementation is required (cf. Fig. 1e).

Taking the relations between the object instances of the overall *data structure* into account, the corresponding micro process instances form a complex *process structure*; i.e., their execution needs to be coordinated in compliance with the given data structure. In PHILharmonicFlows, this is accomplished by means of macro processes. A *macro process type* consists of *macro steps* linked by *macro transitions* (cf. Fig. 1f). Opposed to micro steps, which refer to single attributes of a particular object type, a macro step refers to a particular state of an object type. In addition, for each macro transition, a coordination component must be specified (cf. Fig. 1g). The latter hides the complexity of large process structures from modelers as well as end-users. More precisely, a coordination component coordinates the interactions among the object instances of the same type as well as different types. Opposed to existing approaches, the semantic relations between the object instances and their cardinalities are taken into account as well.

III. Proof-of-Concept Prototype

This section describes the basic features of the PHILharmonicFlows tool we implemented. Besides demonstrating the practical feasibility of the PHILharmonicFlows framework, the following requirements were considered when developing this proof-of-concept prototype:

- **Use of visual models:** The tool shall allow for the graphical definition of data models, micro process types, macro process types, and authorization tables.
- **Enabling correctness-by-construction:** The tool shall enable extensive correctness checks, including a proper visualization of modeling errors.
- **Persistent model storage:** All model artifacts created (e.g., object and process types) as well as run-time data (i.e., object and process instances) shall be stored *persistently*.
- **Automated generation of user interface components:** The tool shall automatically generate end-user components (e.g., overview tables, user forms, and worklists) based on the defined models.
- **Correct process enactment:** The tool shall allow for correct execution of object-aware processes. For this purpose, it must implement the operational semantics defined for micro and macro processes by the PHILharmonicFlows framework.

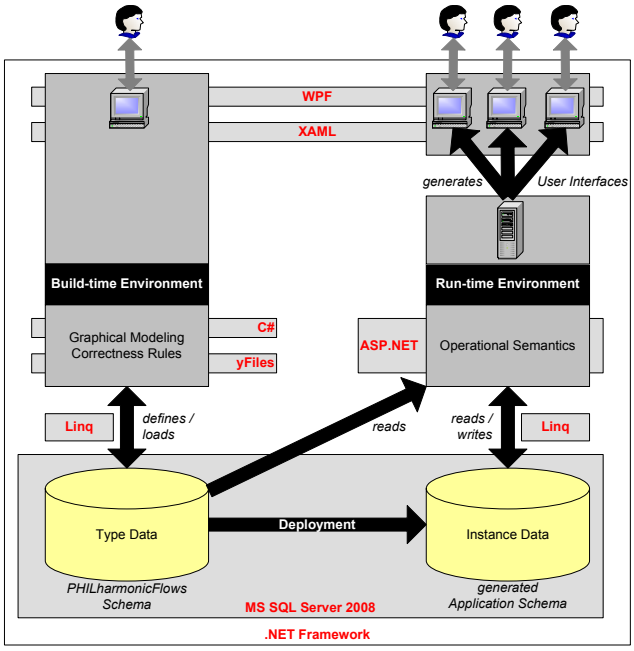

Fig. 2. Architecture of the PHILHarmonicFlows tool and technologies used (from [9])

A. Architecture

The PHILharmonicFlows tool comprises both a build- and a run-time environment (cf. Fig. 2).

The *build-time environment* provides graphical user interfaces for defining data models, micro and macro process types, and user authorizations. In this context, a "correctness-by-construction" principle is applied, and correctness checks for the various models are provided. For example, it is not possible to have two micro step types referring to the same attribute type at the same state type. Further, all created models are stored in the build-time database.

Before creating and enacting instances of an object-aware process, the respective models need to be deployed to the *run-time environment*. For the run-time database, a corresponding application schema is automatically generated based on the pre-specified models. This run-time database is then used to store instance data persistently. Opposed to the build-time

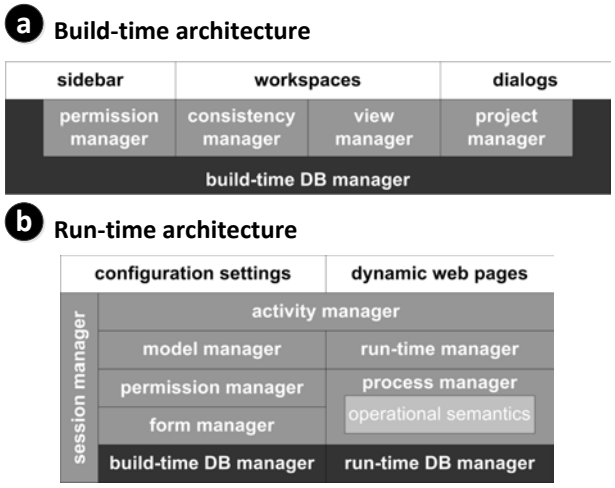

Fig. 3. Run-time and build-time architectures (from [9])

environment, which corresponds to a single-user system, the *run-time environment* is implemented as web application that may be accessed concurrently by various users. In particular, the run-time environment provides generic support for automatically generating user interface components (e.g., user forms and worklists) taking the defined data and process models as input. Furthermore, the run-time environment implements the logic required to correctly execute micro and macro process instances; i.e., it fully supports the operational semantics described by the PHILharmonicFlows framework [9].

The main components of the build-time environment are as follows (cf. Fig. 3a):

- The *build-time DB manager* controls the database connection and provides methods for inserting, deleting, and changing database entries.
- The *permission manager* checks permissions and responsibilities, e.g., which attributes may be accessed by a particular user.
- The *consistency manager* ensures the "correctness-by-construction" principle.
- The *view manager* updates the view of the workspace according to which dimension (i.e., data structure, process structure, or user integration) the user selects on the sidebar (cf. 4).
- The *project manager* opens, saves, and closes modeling projects.

The implemented run-time environment comprises the following components (cf. Fig. 3b):

- The *run-time DB manager* communicates with the database of the run-time environment and manages instance data; i.e., read and write access is provided.
- The *build-time DB manager* communicates with the database of the build-time environment and manages model data.
- The *process manager* allows creating and enacting micro as well as macro process instances.
- The *form manager* automatically generates user forms based on the various build- and run-time artifacts.
- The *permission manager* allows for the integration of user roles; i.e., depending on the currently activated states of the micro process instances, it controls which users may access which data at a certain point in time.
- The *activity manager* controls the functions (e.g., alter data or handle errors) that may be applied to object instances.
- The *run-time manager* is responsible for creating object instances and assigning attribute values to them.
- The *model manager* maintains all models for which there are running instances.

B. Used Technologies

The tool is implemented based on .NET Framework (i.e., C#) and Visual Studio. In this context, the graphical framework Windows Presentation Foundation (WPF) was applied in conjunction with yFiles framework in order to develop the user interface of the various modeling components. The latter are based on the usability concepts presented in [16]. The user interface provides extensive features for creating graphical models. In addition, user interface components are defined in a declarative way using the Extensive Application Markup Language (XAML). In particular, XAML allows separating design from behavior, which enables the re-use of components; i.e., overall development time can be reduced significantly. The run-time environment, in turn, is based on ASP.NET and MS SQL Server 2008 is the database management system. To establish the database connections, we use the Language-Integrated Query (Linq) (cf. Fig. 2).

C. End-User View

The build-time environment of the PHILharmonicFlows prototype is split into three main components: a *menu*, a *sidebar*, and a *workspace* (cf. Fig. 4). The sidebar applies the *accordion navigation* pattern and permits the user to select and access areas representing the different modeling dimensions of a particular object-aware process (i.e., data structure, process structure, and user integration). In addition, each area comprises sub-areas. When selecting an area, it becomes expanded and highlighted. The screen depicted in Fig. 4 represents the data structure dimension, where the user defines the data model with its object types and their relations. In particular, a data model serves as compass for the corresponding process structures; i.e., by using the data model as compass, the user may directly indicate in the data model, for which object type he wants to define a micro process type (cf. Fig. 5). When modeling a macro process type, in turn, both data model and micro process type may be used as structure compasses.

In the *run-time environment*, the user may choose among three different views: *data*, *task*, and *monitoring*. The *data-oriented view* comprises a panel for selecting the desired object type on the left and an overview table listing the object instances (the respective user may access) on the right (cf. Fig. 6). Based on the data-oriented view, users may search for object types, filter object instances, create new object instances,

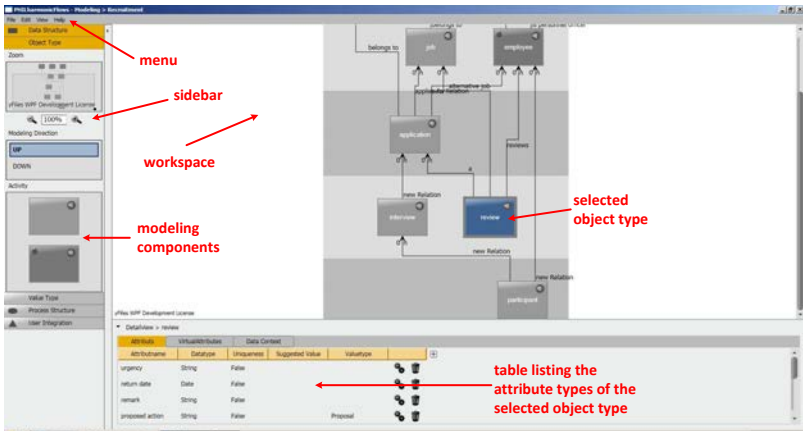

Fig. 4. Build-time user interface - Data structure (from [9])

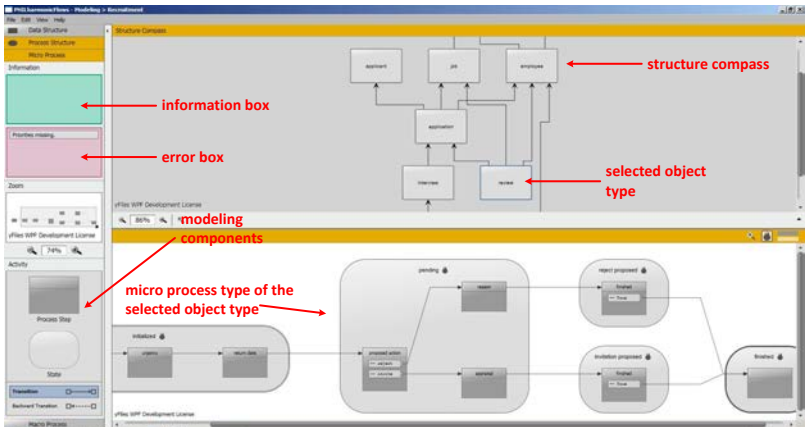

Fig. 5. Build-time user interface - Micro process modeling (from [9])

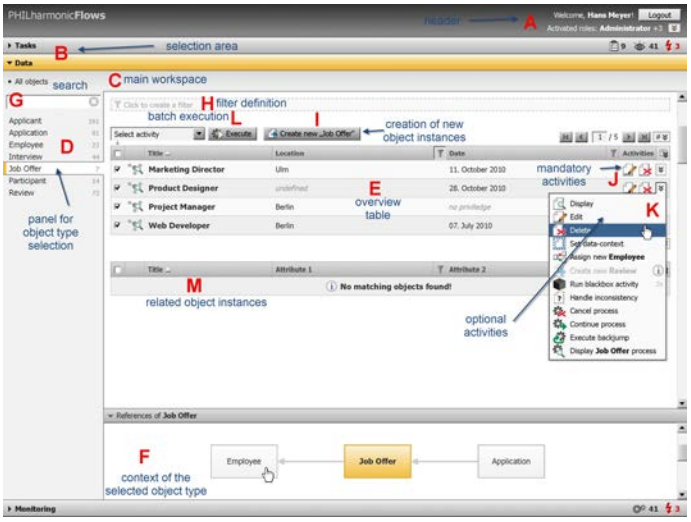

Fig. 6. Data-oriented user view (from [9])

and execute optional as well as mandatory activities on object instances.

The *process-oriented view* (i.e., task view) presents a matrix representing micro process types and corresponding states as rows, and tasks as columns. In particular, each column indicate a particular user task; i.e., the user can whether input data to corresponding attributes, monitor the instances under his responsibility, or handle instance errors. The user may select one of the cells of this matrix and accomplish the corresponding task; i.e., the overview table can be considered as a worklist. In the *monitoring view*, in turn, users may visualize the macro process instances as well as related object instances. In this context, activities for handling macro process instances (and micro process instances respectively) are provided; e.g., for dissolving deadlocks.

IV. Conclusion

In this paper, we presented a proof-of-concept tool we developed in order to validate the technical feasibility of the PHILharmonicFlows framework. This tool supports the modeling, execution, and monitoring of object-aware processes. For this purpose, it provides a build-time as well as a run-time environment. The build-time environment is divided into three dimensions: data structure, process structure, and user authorization. In turn, the run-time environment provides three user views: data-oriented view, process-oriented view, and monitoring view.

As future work, we will extend the framework and its implementation by addressing other issues related to object-aware process management; e.g., historization, traceability, process variability, and process flexibility (including ad-hoc changes and schema evolution of object aware processes).

REFERENCES

[1] W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis and J. Wainer, *"Workflow Modeling using Proclets"*, Proc. CoopIS'00, pp. 198–209, 2000.

[2] W. M. P. van der Aalst, M. Weske and D. Grünbauer, *"Case Handling: A New Paradigm for Business Process Support"*, Data & Know. Eng., 53(2), pp. 129–162, 2005.

[3] K. Bhattacharya, R. Hull and J. Su, *"A Data-Centric Design Methodology for Business Processes"*, Handbook of Research on Business Process Modeling, pp. 503–531, 2009.

[4] C. M. Chiao, V. Künzle and M. Reichert, *"Object-aware Process Support in Healthcare Information Systems: Requirements, Conceptual Framework and Examples"*, Int'l Journal of Advances in Life Sciences, 5(1 & 2), pp. 11–26, 2013.

[5] D. Cohn and R. Hull, *"Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes"*, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 32(3), pp. 3–9, 2009.

[6] V. Künzle and M. Reichert, *"Towards Object-aware Process Management Systems: Issues, Challenges, Benefits"*, Proc. BPMDS'09, LNBIP 29, pp. 197–210, 2009.

[7] V. Künzle and M. Reichert, *"PHILharmonicFlows: Towards a Framework for Object-aware Process Management"*, Journal of Software Maintenance and Evolution: Research and Practice, 23(4), pp. 205–244, 2011.

[8] V. Künzle, B. Weber and M. Reichert, *"Object-aware Business Processes: Fundamental Requirements and their Support in Existing Approaches"*, Int'l Journal of Information Systems Modeling and Design, 2(2), pp. 19–46, 2011.

[9] V. Künzle, *"Object-aware Process Management"*, Ph.D. Thesis, University of Ulm, Germany, 2013.

[10] D. Müller, M. Reichert and J. Herbst, *"Data-Driven Modeling and Coordination of Large Process Structures"*, Proc. CoopIS'07, LNCS 4803, pp. 131–149, 2007.

[11] D. Müller, M. Reichert and J. Herbst, *"A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures"*, Proc. CAiSE'08, LNCS 5074, pp. 48–63, 2008.

[12] G.M. Redding, M. Dumas, A. H. M. ter Hofstede and A. Iordachescu, *"A Flexible, Object-centric Approach for Business Process Modelling"*, Proc. SOCA'09, pp. 1–11, 2009. Reichert and P. Dadam, *A Framework for Dynamic Changes in Workflow Management Systems*, Proc. DEXA'97, pp. 42–48, 1997.

[13] M. Reichert and B. Weber, *"Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies"*, Springer, 2012.

[14] Silver, B., *"Case Management: Addressing Unique BPM Requirements"*, BPMS Watch, Industry Trend Reports, 2009.

[15] I. Vanderfeesten, H. A. Reijers and W. M. P. van der Aalst, *"Product-Based Workflow Support: Dynamic Workflow Execution"*, Proc. CAiSE'08, LNCS 5074, pp. 571–574, 2008.

[16] N. Wagner, *"Entwicklung eines Usability-Konzepts für die Modellierungsumgebung eines datenorientierten Prozess-Management-Systems"*, Diploma thesis, Ulm University, 2010 (in German).