Integrated Modeling of Process- and Data-Centric Software Systems with PHILharmonicFlows

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

***Abstract*—Process- and data-centric software systems require a tight integration of processes, functions, data, and users. Thereby, the behavioral perspective is described by process models, while the information perspective is captured in a data model. Eliciting and capturing requirements of such software systems in a consistent way is a challenging task, demanding that both process and data model are well aligned and consistent with each other. While traditional software modeling languages do not allow for an explicit integration of data and process models, activity-centric process modeling languages tend to neglect the role of data as a driver of process execution; i.e., business objects are usually outside the control of the process, normally stored in external databases. To overcome this drawback, PHILharmonicFlows provides a comprehensive framework for enabling object-aware process support. In addition, a sound specification of process- and object-centric software systems becomes possible. In this paper, we present a requirements modeling approach that provides methodological guidance for modeling large process- and data-centric software systems based on PHILharmonicFlows. Such guidance will foster the introduction of respective software systems in the large scale.**

***Keywords*—process and data-centric software systems; requirements modeling; process modeling;**

I. Introduction

In general, a software system can be considered as useful, if it meets the requirements of its users and environment [4], [28]. Usually, software modeling techniques are used for capturing such requirements. In particular, corresponding models allow stakeholders to provide proper feedback in early phase during software development.

Process- and data-centric software systems are becoming increasingly popular in the enterprise computing. They necessitate a tight integration of process, data, users, and application services [16]. Eliciting and modeling the requirements of such software systems constitutes a challenging task. Usually, the behavioral perspective of such a system (i.e., what the system shall *do*) is captured by business process models, while the information perspective (i.e., what the system shall *store*) is reflected by a data model [23]. Thus, both kinds of models are complementary and indispensable in order to describe the requirements of a corresponding software system. However, traditional modeling languages like the *Unified Modeling Language* (UML) do not allow for a tight integration of data and process models, which are usually handled separately from each other. In particular, the role of data as driver of process execution is not well understood. To maintain the consistency and compliance of process and data models therefore constitutes a manual task, which must be accomplished by system analyst, and which is usually prone to errors.

On one hand, traditional software modeling languages do not provide well integrated models for capturing the process and data perspectives of such a system. On the other, process modeling languages like *Business Process Model and Notation* (BPMN) [36], [37] and *Event-driven Process Chain* (EPC) [34] lack adequate support for modeling the information perspective and ignore the role of data as driver for process execution. This is due to the fact that these process modeling languages have been mainly designed for modeling *activity-centric* processes. Such processes are described in terms of "black-box" activities and their control-flow defines the order and constraints for executing these activities [1], [17]. In turn, data is usually represented in terms of business objects whose attributes may be written or read by certain activities of the modeled process. However, details concerning these objects (e.g., attributes, relationships, and object behavior) are not handled within the process model. Furthermore, processes instances may interact with process instances of the same or different type; i.e., the processing of a particular process instance might require data from other processes instances. Using traditional modeling languages, it is not possible to model such interactions in a proper way (e.g., modeling at which points during the execution of a process instance, data from other processes instances is needed). This limitation is caused by a missing understanding of the role data has as a driver of process execution [14], [33].

During the recent years, data- and artifact-centric process support paradigms have emerged, which aim at overcoming the limitations caused by missing integration of process and data [1], [2], [26], [27], [30], [35]. However, none of them fully considers the separation of concerns principle to ensure low complexity and to allow for the proper visualization of the models and the respective software requirements; i.e., the models resulting from the use of these approaches do not foster an understanding of the application domain or the various relationships between processes, data, functions, and users (e.g., data access authorization settings) [23].

Opposed to these approaches, PHILharmonicFlows provides a comprehensive framework enabling object-aware process support. We have already described various aspects of this framework in previous work [14], [15], [16], [17], [18], [20]. In turn, the focus of this paper is on the modeling methodology applied in the context of PHILharmonicFlows. In particular, PHILharmonicFlows provides a well-defined process- and data-centric software modeling methodology governing the

object-centric specification of large process-oriented software systems. Data and process models are still handled separately, to enable a proper understanding of the software requirements in respect to the behavior and information perspectives of the software system. Furthermore, the process models are derived from the information perspective, which is covered by the data model (i.e., data objects and their relations). Additionally, the behavioral perspective of the software system is represented in two different levels of granularity: *micro* and *macro* process types. A micro process type defines the behavior of a particular object type; i.e., it defines how the processing of individual object instances of this type shall be coordinated among process participants and how valid attribute settings look like in this context. A macro process type, in turn, defines how object instances interact with other objects instances; i.e., how the processing of inter-related object instances (i.e., of their micro processes) shall be coordinated. Moreover, PHILharmonicFlows allows specifying which users shall be authorized to access and manage process-related data at defined points during process execution. In this paper, we present a requirements modeling approach that provides methodological guidance for modeling process- and data-centric software systems using PHILharmonicFlows framework. In particular, such a methodology is indispensable for the successful use of such a framework.

Section II describes a case study and an example we use for illustrating our modeling methodology. In Section III, we present the main characteristics of object-aware processes. The modeling methodology is described in Section IV. In Section V, we discuss how we validated this methodology by applying it in different application domains. Further, we present a prototype as a proof-of-concept. Related work is discussed in Section VI. Finally, Section VII concludes with a summary and an outlook.

II. Case Study

Our methodology has been applied in several application domains (cf. Section V). As illustrating example, we use a real scenario from a Brazilian university we gathered in one of our case studies. It comprises the project of a software system that manages extension course proposals (cf. Fig. 1). Extension courses are courses for professionals that aim to refresh and update their knowledge in a certain area of expertise. In order to propose a new extension course, the course coordinator must create a project describing it. The latter must then be approved by the faculty coordinator as well as the extension course committee.

Illustrating Example (Extension course proposal): The `course coordinator` creates an `extension course project` using a form. In this context, he must provide details about the course, like `name`, `start date`, `duration`, and `description`. Following this, `professors` may start creating the `lectures` for the extension course. In turn, each `lecture` must have detailed `study plan items`, which describe the topics that will be covered in the lecture.

After creating the lectures, the `coordinator` may request an approval of the `extension course project`. First, an approval must be provided by the `faculty director`. If he wants to `reject` the proposal, the extension course must not take place. Otherwise, the project is sent to the `extension course committee`, which will evaluate it. If there are more `rejections` than `approvals`, the `extension course project` will be `rejected`. Otherwise, it will be `approved` and hence may take place in future.

III. Background

In order to provide a better understanding on how our software requirements modeling methodology emerged, we first introduce the main characteristics of *object-aware processes*. An intensive study concerning these characteristics has shown that existing data-centric approaches do not cover all phases of the process life cycle of object-aware processes (for more details see [19], [32]). To overcome this drawback, we developed the PHILharmonicFlows framework, which aims at adequately supporting the fundamental characteristics of object-aware processes. Moreover, the framework provides a well-defined modeling methodology for large process- and data-centric software systems, which will be presented in the second part of this section.

The PHILharmonicFlows framework resulted from an intensive study analyzing various processes from different domains [14], [15], [19]. As result of this study, a set of properties was gathered, characterizing *object- and process-awareness in information systems*. Basically, for *object-aware* processes data must be manageable in terms of *object types*. At runtime, the different object types then may comprise varying numbers of object instances, whereby the concrete instance number may have to be restricted by lower and upper cardinality bounds. In the example from Fig. 2a, for each `lecture`, at least one and at most five `study plan items` may be initiated.

The modeling and execution of processes must be in accordance with the specified data model. In particular, processes are specified at two different levels of granularity: *object behavior* and *object interactions*.

A. Object Behavior

To cover the processing of individual object instances, the first level of process granularity refers to *object behavior*. More precisely, for each object type a separate process definition is provided. In turn, the latter is then used for coordinating the processing of individual object instances among different users. In addition, it is possible to determine in which order and by whom the attributes of a particular object instance must be (mandatorily) written, and what valid attribute settings (i.e., attribute values) are. Consequently, at runtime, the creation of an *object instance* is directly coupled with the one of its corresponding *process instance*. In this context, it is important to ensure that mandatory data is provided during process execution. Therefore, it must be possible to define object behavior in terms of *data conditions* rather than based on black-box activities.

B. Object Interactions

Since related object instances may be created or deleted at arbitrary points in time, a *complex data structure* emerges

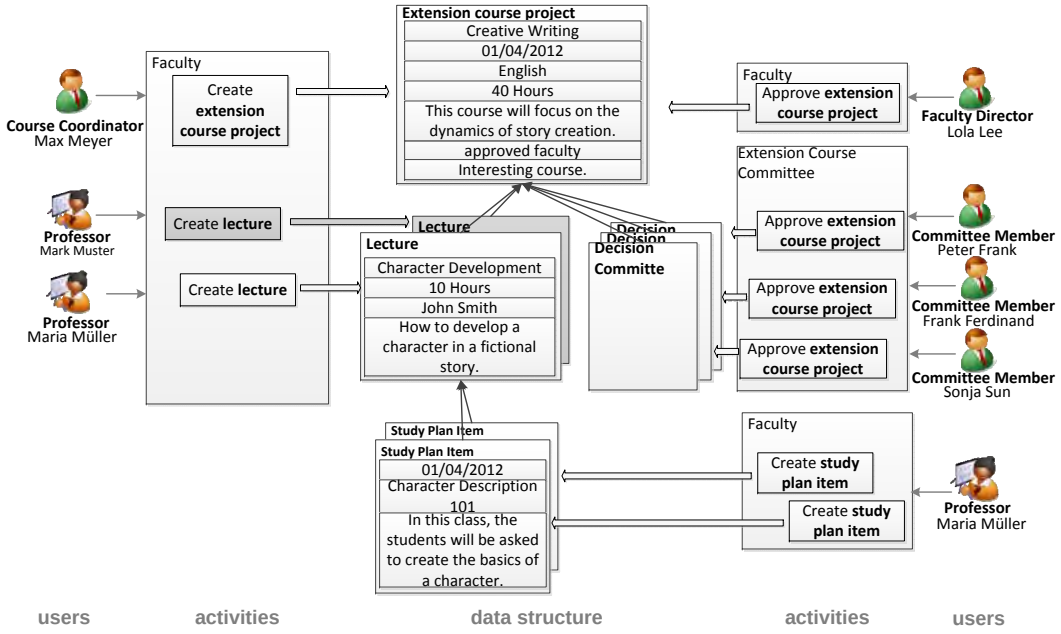

Fig. 1. Case Study and Illustrating Example

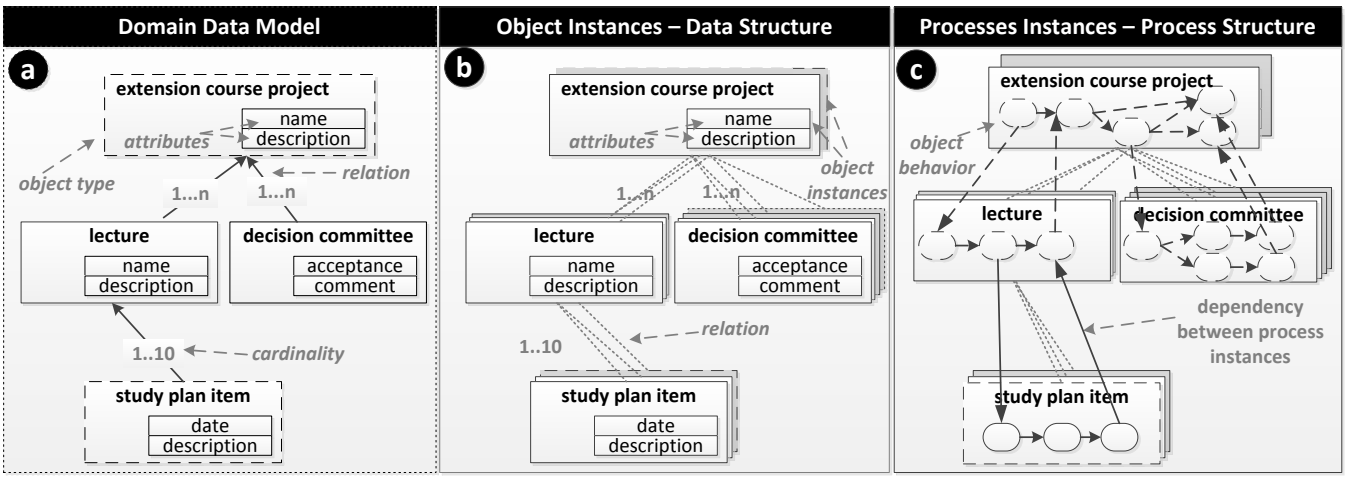

Fig. 2. Example of Data Structure (a and b) and Process Structure (c)

that dynamically evolves during runtime, depending on the types and number of created object instances (cf. Fig. 2c). Furthermore, individual object instances of same type may be in different processing states at a certain point of time. The second level of process granularity we consider, therefore, comprises the *interactions* between related object instances of same and different types. A mechanism is required that allows coordinating the execution of concurrently executed process instances (each one related to a particular object instance).

C. Data-driven Execution

To proceed with the processing of a particular object instance, in a given state, certain attribute values are mandatorily required. Hence, object attribute values reflect the progress of the corresponding process instance (i.e., the processing state of the respective object instance). More precisely, the setting of certain object attribute values is enforced in order to progress with the process through the use of *mandatory activities*. However, if the required data is already available, these activities may be automatically skipped when becoming activated. Furthermore, to enable flexible process execution, users should be allowed to *re-execute* a particular activity, even if all mandatory object attributes have been already set. For this purpose, data-driven execution must be combined with *explicit user commitments*; i.e., users should be allowed to review and update the values that are set for the attributes of a particular activity and to confirm the completion of the latter. Finally, the execution of a mandatory activity may depend on attribute values of related object instances. Thus, the coordination of multiple process instances should be supported in a data-driven manner as well.

D. Variable Activity Granularity

For creating object instances and changing object attribute values, *form-based activities* shall be used. Respective user forms comprise both *input fields* (e.g., text fields or check-boxes) for writing and *data fields* for reading selected attributes of object instances. However, note that different users may prefer different work practices. As a consequence, depending on the work style preferred, an activity may either be related to one or to multiple process instances; i.e., different working styles need to be enabled. Regarding *instance-specific activities*, for example, all input and data fields refer to attributes of one particular object instance, whereas the forms of *context-sensitive activities* also comprise fields referring to different, but semantically related object instances (of potentially different type). For example, when editing attribute values related to a particular instance of object `lecture`, it might be favorable to also edit attribute values of the corresponding instances of object `study plan item`. Finally, *batch activities* involve several object instances of the same type; i.e., the values of different input fields can be assigned to all involved object instances in one go.

In addition to form-based implementation of activities, it must be possible to integrate *black-box activities* as well. The latter might be required, for example, to enable complex computations as well as the integration of advanced functionalities (e.g., as provided by web services).

IV. Modeling Software Requirements with PHILharmonicFlows framework

To support information system engineers in developing object- and process-centric software systems, this section presents a well-defined modeling methodology that governs the object-centric definition of processes at different levels of granularity. For this purpose, we differentiate between *micro* and *macro processes* capturing both *object behavior* and *object interactions*. In detail, our modeling methodology comprises three major steps (cf. Fig. 3): *stakeholders elicitation and domain data modeling*, *behavior and functional requirements modeling*, and *rapid prototyping*. Each of these steps is divided into one or more tasks, of which each produces artifacts covering different aspects of the software system.

During process execution, PHILharmonicFlows automatically generates user forms, taking user authorization and process state into account; no manual efforts are required in this context. Hence, with this approach it becomes more simple for the system analyst to rapidly create prototypes during software development and to let the stakeholders test them and give early feedback. The latter can then be used iteratively to improve and refine the generated artifacts (i.e., software models) until the captured requirements reflect the needs of the stakeholders.

A. Stakeholders Elicitation & Domain Modeling

In this first step, the information perspective is modeled. By creating a data model, domain objects are identified and modeled through the definition of *object types* and their *relations* (including cardinalities). Using the same model, the organizational entities are defined based on so-called *user types*. Overall, the data model constitutes the core artifact based on which the processes describing the behavior and interaction between objects are derived (i.e., micro and macro process types).

1) Organizational Modeling: The organizational model is integrated into the data model; i.e., user roles are considered as object types as well. Each user role type is modeled as a specific object type that is denoted as *user type*. The latter comprises attributes, which can be used to characterize the user role type, e.g., *name*, *e-mail address*, or, as in our case study, the *faculty* the user belongs to. In the example from Fig. 4, the user types include *course coordinator*, *professor*, and *committee member*. To express that the course coordinator and professors belong to the same faculty, the instances of both user types must have attribute *faculty* filled with the same value. In PHILharmonicFlows, one user may possess more than one role during process execution. Finally, the relation between user and user roles is defined during runtime.

2) Domain Data Modeling: In the same data model comprising the user types, the *object types* describing the domain-specific data objects are added. Each object type comprises a set of attributes and may be related to other object types. At runtime, these relations allow for a varying number of interrelated object instances whose processing must be coordinated. In particular, the data model is divided into *data levels* (cf. Fig. 4 for an example of a data model comprising three such levels). All object types not referring to any other object type are placed at the top level (Level #1). Generally, an object type is assigned to a lower data level as the object types it refers.[1] Additionally, cardinality constraints of the data model might restrict the minimum and maximum number of instances of an object type that may refer to the same higher-level object instance. In our example (cf. Fig. 4), object types `lecture` and `decision committee`, for instance, refer to object type `extension course project`. Both have cardinalities `1..n`. In turn, an instance of object type `lecture` may refer to maximum 10 instances of object type `study plan item`.

B. Behavior & Functional Requirements Modeling

In this second step, the behavioral perspective of the software system is defined; i.e, what the system shall *do* and how this shall be accomplished is modeled in this step. Particularly, the process models produced in this context cannot be handled apart from the information perspective (i.e., data model). To enable object-aware processes (i.e., to define business processes in tight integration with business data), their modeling must consider two levels of granularity (cf. Sect. III). More precisely, *object behavior* and *object interactions* must be captured in two different kinds of models. To describe object behavior, we must specify a process definition for each object type. Such definition is called *micro process type*. In turn, the interactions among multiple objects of the same or different types are captured by a *macro process type*.

The behavior of a single object (i.e., a micro process) is expressed by a number of possible *states* and *transitions*. Thereby, the progress of an object instance is driven by changes of its attributes; i.e., a precise link between data and process state is established. In particular, whether or not a

[1] Note that this requires a special treatment of cyclic data objects. We refer for [16] for a respective discussion

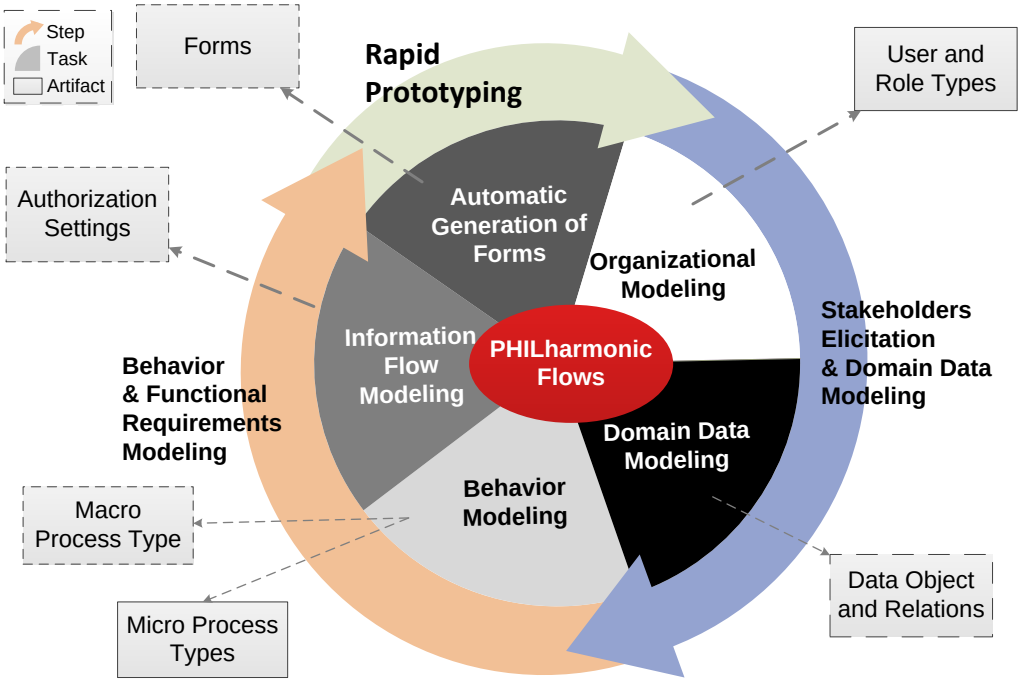

Fig. 3. Modeling Methodology

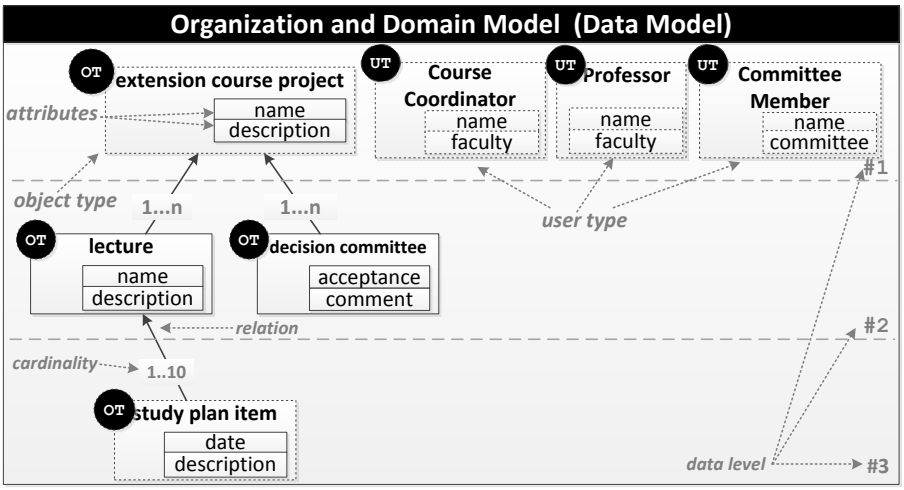

Fig. 4. Example of a Data Model

particular state is reached during runtime depends on the values of object instance attributes. In turn, the interactions among objects instances become enabled when involved objects reach certain states. Consequently, object states serve as an abstract interface between micro and macro processes.

1) Behavior Modeling: When referring to behavior modeling, not only the behavior of a single object type is described, but also the interactions among different objects instances of the same or different types. Two different artifacts result from this: micro process types describing the behavior of involved object types on one hand, and a macro process type describing the interactions among objects of the same or different types on the other.

In PHILharmonicFlows, for each *object type* defined in the the data model, a specific *micro process type* must be defined. At runtime, both object instances of the same and of different object types may be created at different points in time. In particular, the creation of a new object instance is directly coupled with the one of a corresponding micro process instance. The resulting *micro process type* then coordinates the processing of an object among different users and specifies what valid attribute settings are.

Each micro process type comprises a number of *micro step types*, which describe elementary actions for reading and writing object attribute values. More precisely, each micro step type is associated with one particular attribute of the respective object type. In turn, micro step types may be linked using *micro transition types*. To coordinate the processing of individual object instances among different users, micro step

types need to be grouped into *state types*; i.e., each state postulates specific attribute values to be set. Such state types are associated with one or more user roles. During runtime, each association between a user role and a state type is represented as a mandatory activity (i.e., a work item in the work list of the respective user). Such activities are form-based, where each field of the form corresponds to an attribute (i.e., micro step type) associated to the correspondent state type.

At runtime, a micro step may be reached (i.e., completed) if for the corresponding attribute a value is set. In turn, a state may be left (i.e., the next state be activated) if values for all attributes associated with the micro steps of this state are set. Whether or not the subsequent state in the micro process is immediately activated, however, then also depends on user decisions; i.e., process execution may be both data- and user-driven [32]. To enable user involvement, micro transition types connecting micro step types of different state types may be categorized either as *implicit* or *explicit*. Using implicit micro transitions, the target state will be automatically activated as soon as all attribute values required by the previous state become available. In turn, explicit micro transitions additionally require a user commitment before activating the next state; i.e., users may decide whether or not the subsequent state shall be activated. This way, users are enabled to still change attribute values corresponding to a particular state even if all of them implicitly have been already set.

An example of a micro process type is depicted in Figure 5a. A corresponding micro process instance is initiated in state `under creation`, for which values of the attributes `name`, `start_date`, `faculty`, `credits`, and `description` must be set. Regarding state `under approval faculty`, for example, a user decision is modeled, which is related to micro step type `decision_faculty`. At runtime, a user associated with the role `faculty director` must then decide whether to `reject` or `approve` the `extension course project`. Finally, the given micro process type has two end state types: `rejected` and `approved`. Which of these two states becomes activated at runtime and hence will terminate the processing of the object instance depends on the decision made in the context of micro step `decision_faculty` (i.e., the value of set for attribute `decision_faculty`).

In general, whether or not subsequent states may be reached also depends on the processing states of other micro process instances. At runtime, for each object instance a corresponding micro process instance exists. As a consequence, a characteristic process scenario may comprise hundreds of micro process instances. Taking their various interdependencies into account, we obtain a complex and large process structure. In order to coordinate the interactions between the different micro process instances of such process structure, a coordination mechanism is established that allows specifying the interaction points of the micro processes. More precisely, the system analyst must specify at which points during process execution a particular micro process instance needs input from other micro process instances of same or different type (cf. Fig. 6a). In turn, these interaction points can be modeled for a macro process type (cf. Fig.6b). Such a *macro process type* refers to parts of the data structure and consists of *macro steps types* as well as *macro transitions types* linking them. As opposed to traditional process modeling approaches, where process steps are defined in terms of activities, a macro step type always refers to an object type and a corresponding state type.

The macro process type corresponding to our example is illustrated in Figure 6b. The macro process begins with creating an instance of object type `Extension Course Project`. In turn, this triggers the execution of a corresponding micro process instance. Following this, a varying number of instances of micro processes corresponding to instances of object type `Lecture` are created. For each instance of object type `Lecture`, an arbitrary number of instances of `Study Plan Item` may be created as well. When all instances of `Study Plan Item` reach state `finished`, the corresponding instance of `Lecture` may be finished as well.

Since the activation of a particular micro process state may depend on instances of other micro process types, macro input types are assigned to macro step types. The latter may then be associated with several macro transitions. To differentiate between AND and OR semantics in this context, it is further possible to model more than one macro input for a macro step type. At runtime, a macro step will become enabled if at least one of its macro inputs is activated. In turn, a macro input will be enabled if all incoming macro transitions are triggered.

Information Flow Modeling

Regarding information flows, we must specify how the object instances shall be coordinated among the system users; i.e., which users shall be able to read and write which object attributes at which point during the execution of the respective instance. The artifact to be created for this purpose documents the *authorization settings* for each micro process type.

At the micro process level, the state types, which contain a number of micro step types referring to object attributes, must be assigned to user roles. At runtime, the users possessing the respective roles are responsible for setting values of the respective attributes.

To ensure that users that may be assigned to a micro process state have sufficient privileges to write mandatory attributes associated with this state, a minimal *authorization table* is automatically generated for each object type. More precisely, PHILharmonicFlows grants different permissions for reading and writing attribute values as well as for creating and deleting object instances to different user roles. In this context, the different states are considered as well (i.e., users may have different permissions in different states). The authorization to write an attribute may either be *mandatory* or *optional*. When the table is generated, the user role associated to a state automatically receives a mandatory write authorization to all attributes related to micro step types of the respective state type. Optional data permissions, however, may be additionally assigned to user roles not associated to the state type. This way, even users usually not involved in process execution are allowed to access process relevant data. Note that these authorization settings also provides the basis for the automatic generation of user forms in the *prototyping* step.

An example of data authorization settings for the micro process of object type `Extension Course Project` is shown in Figure 5b. The latter illustrates the authorization table for micro process type `Extension Course Project`. In this

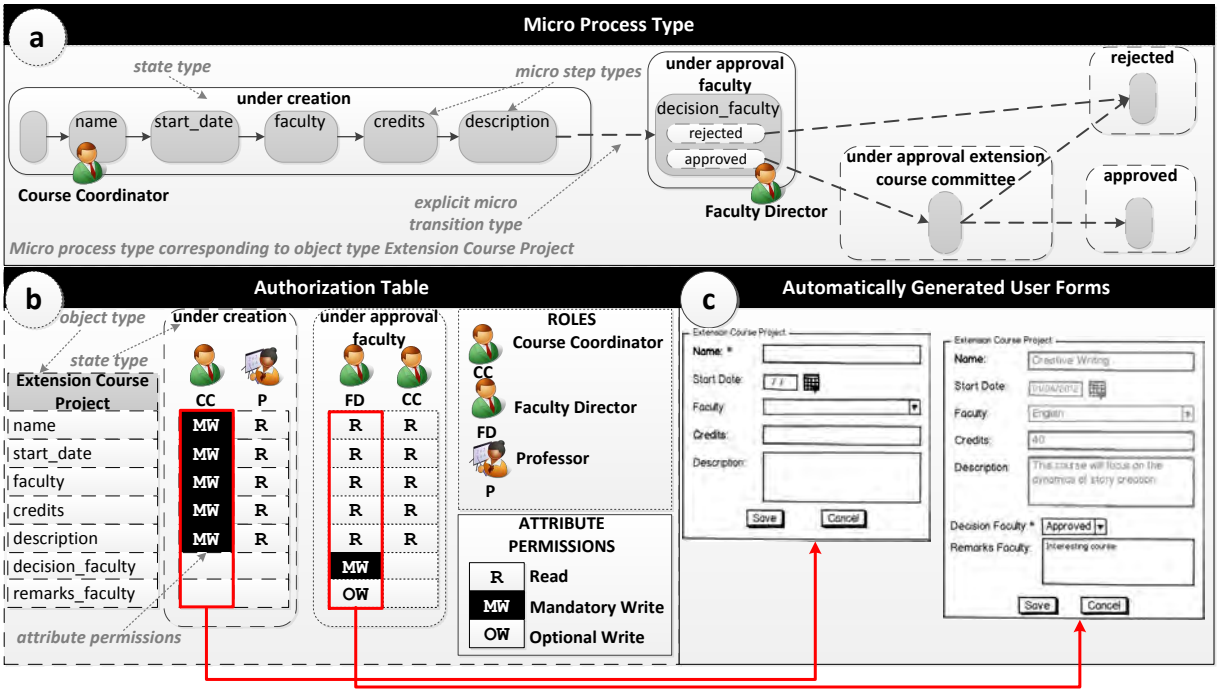

Fig. 5. Example of a Micro Process Type and Related Components

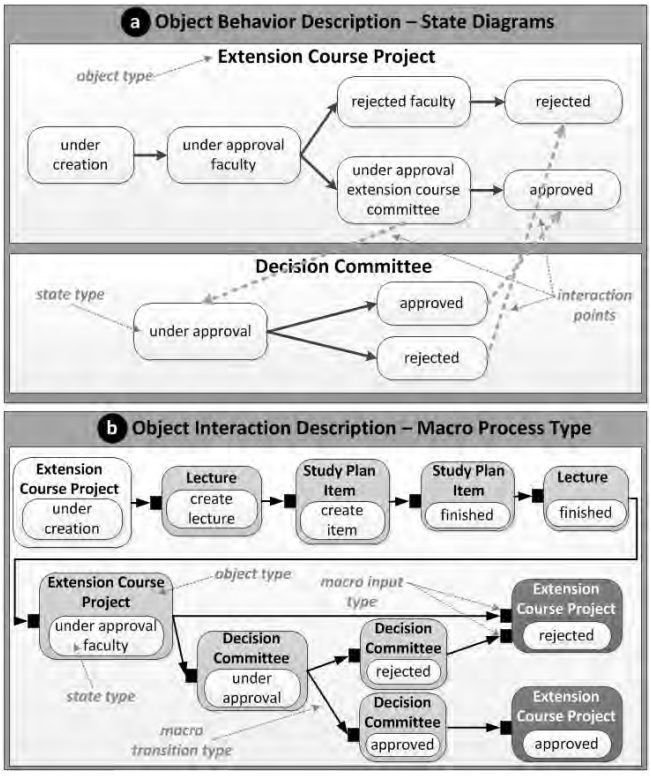

Fig. 6. Example of a Macro Process Type

example, user `course coordinator` (CC) must provide values for attributes `name`, `start_date`, `faculty`, `credits`, and `description` in state `under creation`; accordingly, these attributes are marked as `MW`. In the same state, the `professor` (P) may read the values of these attributes (marked as `R` for the role). In state `under approval faculty`, in turn, the `faculty director` (FD) must fill attribute `decision_faculty` (marked as `MW`). In addition, attribute `remarks_faculty` may be optionally written (marked as `OW`).

C. Rapid Prototyping

Achieving the rapid prototyping step, the system analyst is enabled to rapidly create a prototype of the modeled process- and data-centric software system, which then can be executed and explored.

1) Automatic Generation of Forms: At runtime, based on the authorization settings defined, PHILharmonicFlows automatically generates forms. Which input fields are displayed to a specific user depends on the attribute permissions valid for the currently activated state. When the user only has the permission to read an attribute in a particular state, the form field is disabled and marked as read-only. In turn, a mandatory or optional attribute is represented through an editable field. In particular, mandatory fields are highlighted in the form. In Figure 5c, forms for states `under creation` and `under approval faculty` of micro process type `Extension Course Project` are exemplary depicted. Furthermore, the control flow logic within a form itself is derived from the internal state transitions defined between the respective micro step types. According to the micro process type from Fig. 5a, in state type `under creation`, micro step type `name` precedes micro step type `start_date`. At runtime, the corresponding form (cf. Fig. 5c) will only display the input field corresponding to micro step type `start_date` as mandatory if a value is set for attribute `name`.

V. Validation

In Section IV, we presented a modeling methodology for data- and process-centric software systems. The latter comprises three different steps, where different artifacts (i.e., models and prototypes) regarding the software system are generated (cf. Fig. 3). Differently from other modeling methodologies, this one permits the modeling of the behavioral perspective (e.g., process models) in tight integration with the information one (e.g., data models), without violating the separation of concerns principle. Such methodology was evolved from several case studies, during which we modeled object-aware processes in various domains, including human resource management [14], [15], [16], [17], healthcare [5], [7], scientific paper reviewing, and house construction. In these case studies, we observed that the alignment and consistency between the information (data) and behavioral (process) perspectives can be established at a high level of abstraction when using the PHILharmonicFlows framework. Note that the use of this framework ensures that the various models are treated separately; i.e., the data model is separated from the process models. Furthermore, the behavioral perspective of the software system is represented at two levels of granularity, providing a separate and unique view of object behavior and object interactions. Opposed to many other data-centric approaches [1], [2], [26], [27], [30], [35], the artifacts generated based on our methodology allow for a proper visualization of the models with respect to software requirements, providing a good understanding of the application domain. In addition, by automatically generating the user forms, the PHILharmonicFlows framework allows for the rapid generation of an executable version of the software system. Such rapid prototyping can be used for iteratively improving and refining the generated artifacts until the requirements of the system users are met. Hence, our methodology allows for a complete modeling of requirements.

We developed a proof-of-concept prototype, which supports the modeling and enactment of object-aware processes. Figures 7 and 8 show examples of screens as provided by the modeling environment. Figure 7 shows a data model comprising object and user types. Figure 8 shows a micro process type: the upper part of the depicted screen presents the object types and their relations. Further, the selected object type for which a micro process type is modeled is depicted (see the bottom of Fig. 8).

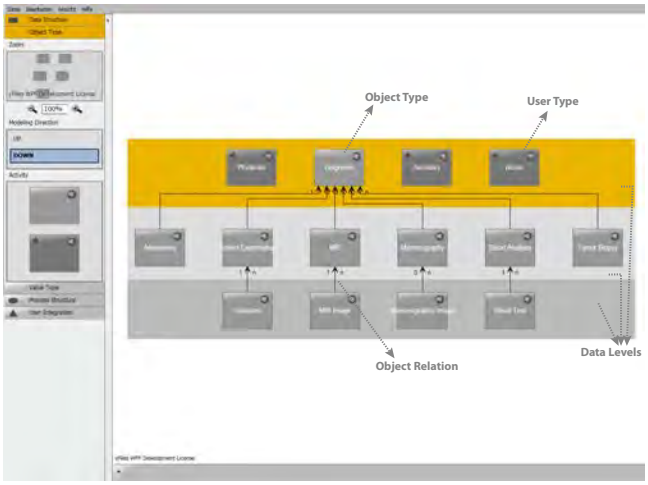

Fig. 7. Example of a Screen Showing a Data Model

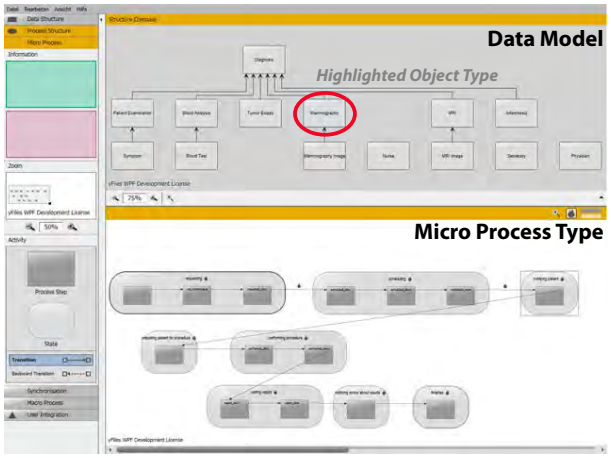

Fig. 8. Example of a Screen Showing of a Micro Process Type

VI. Related Work

Model Driven Architecture (MDA) [29], [3] is a well-known approach in software engineering, which aims at the specification of complex and large software systems. It treats models as proper artifacts during the software development process. Using different kinds of models, it further allows creating requirements specifications and translating these models into platform-specific executable software code. Such a model-driven approach is usually related to modeling standards, including Unified Modeling Language (UML), Meta-Object Facility (MOF), and Common Warehouse Meta-Model (CWM). Although MDA is very powerful regarding software specification and code generation, methodological support on how to define and apply respective models is barely provided. Furthermore, there exist methodologies in the context of distributed applications, such as ODAC [11] and MODA-TEL [10], as well as methodologies dealing with software development process in general (e.g., MASTER [24]). In [12], it is discussed how the concepts of MDA may be applied on process-aware information systems (PAIS) development. However, the data perspective is not taken into account. Finally, [13] presents a model-driven approach for generating form-based activities in the context of activity-centric PAIS.

In [23], the authors present a requirements engineering approach that provides guidelines for integrating process and data models. The approach proposed in [23] is based on two existing approaches: the Business Process-Based approach [21], [22] and the Info Cases Approach [9]. However, opposed to our approach, models are still created separately (i.e., a semantic verification is not provided in order to check the compliance and correctness of both process and data model). Further, information flows are still designed in terms of textual information. These information flows are described in terms of BNF grammar, which may not be very clear for stakeholders. Opposed to this, in our approach, information flow is described based on micro step types and authorization settings for the micro process types, which is much more intuitive for stakeholders. In addition, [23] does not take into account the rapid prototyping features provided by the PHILharmonicFlows framework.

Other data-centric approaches like case handling [1], artifact-centric business processes [2], and product-based workflows [35] do not consider the separation of concerns; i.e., data and process are considered in the same model. Therefore, fully understanding the requirements of an application domain is more difficult and the modeling of the data and functional aspects of the software system is more complex.

VII. Summary & Outlook

This paper introduced an integrated methodology for modeling requirements of process- and data-centric software systems. In order to precisely capture the elicited requirements, we use the PHILharmonicFlows framework, which provides a well-defined process and data-centric software modeling methodology, governing the object- and user-centric specification of processes. This methodology has been already successfully applied in various case studies in different application domains.

Figure 9 illustrates how the artifacts generated by our methodology cover the different aspects of the software system to be realized. Further, it can be seen how these aspects are related to each other. By modeling the object types and their relations, fundamental insights into information perspective can be obtained. Additionally, they constitute the basis of the other artifacts such as micro process types, describing object behavior, and macro process types, which describe object interactions. Moreover, by using our framework, it becomes possible to rapidly generate executable software, without need for implementing user interfaces; i.e., PHILharmonicFlows automatically generates user forms. This allows stakeholders to test the system and to provide early feedback, which improves the overall quality of the software system to be developed.

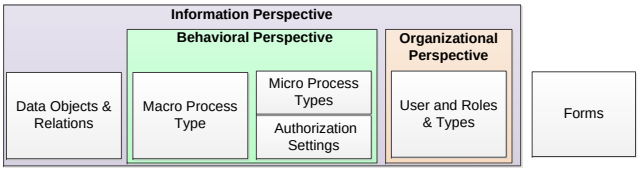

Fig. 9. Organizational, Information, and Behavior Perspectives as Generated by PHILharmonicFlows

In order to further increase the flexibility and adaptability of object-aware process support, we are developing advanced concepts and techniques enabling schema evolution in object- and process-aware software systems. Preliminary work discussing the challenges to be tackled in this context is presented in [6]. Other future work will deal with the mining and analysis of the execution logs of object-aware processes with the goal to discover data objects and the relations among them. By analyzing such logs, we can not only discover which data objects are involved, but also investigate their behavior (i.e., who and in which order may attributes be set) as well as their interactions.

Acknowledgment

The authors would like to acknowledge the financial support provided by the Ernst Wilken Foundation.

References

[1] 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.

[2] 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.

[3] A. W. Brown, *Model Driven Architecture: Principles and Practice*, Software and Systems Modeling, 3(4), pp. 314–327, 2004.

[4] B. H. C. Cheng and J. M. Atlee, *Research Directions in Requirements Engineering*, Proc. FOSE '07, pp. 285–303, 2007.

[5] C. M. Chiao, V. Künzle and M. Reichert, *Towards Object-aware Process Support in Healthcare Information Systems*, Proc. eTELEMED 2012, pp. 227–236, 2012.

[6] C. M. Chiao, V. Künzle and M. Reichert, *Schema Evolution in Object and Process-Aware Information Systems: Issues and Challenges*, Proc. BPM'12 Workshops, LBNIP 132, pp. 328–340, 2012.

[7] 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.

[8] 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.

[9] M. H. Fortuna, C. M. L. Werner and M. R. S. Borges, *Info Cases: Integrating Use Cases and Domain Models*, Proc. RE'08, pp. 81–84, 2008.

[10] A. Gavras, M. Belaunde, L. F. Pires and J. P. A. Almeida, *Towards an MDA based Development Methodology*, Proc. EWSA 2004, LNCS 3047, pp. 71–81, 2004.

[11] M.-P. Gervais, *Towards an MDA-Oriented Methodology*, Proc. COMPSAC 2002, pp.265–270, 2002.

[12] E. Kindler, *Model-based Software Engineering and Process-aware Information Systems*, Transactions on Petri Nets and Other Models of Concurrency II, pp. 27–45, 2009.

[13] J. Kolb, P. Hübner and M. Reichert, *Automatically Generating and Updating User Interface Components in Process-Aware Information Systems*, Proc. CoopIS'12, LNCS 7565, pp. 444–454, 2012.

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

[15] V. Künzle and M. Reichert, *Integrating Users in Object-aware Process Management Systems: Issues and Challenges*, Proc. BPM'09 Workshops, LNBIP 43, pp. 29–41, 2009.

[16] 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.

[17] 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.

[18] V. Künzle and M. Reichert, *A Modeling Paradigm for Integrating Processes and Data at the Micro Level*, Proc. BPMDS'11, LNBIP 81, Springer, pp. 201–215, 2011.

[19] V. Künzle and M. Reichert, *Striving for Object-aware Process Support: How Existing Approaches Fit Together*, Proc. SIMPDA'11, 2011.

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

[21] J. L. de la Vara, J. Sànchez and Ò. Pastor, *Business Process Modelling and Purpose Analysis for Requirements Analysis of Information Systems*, Proc. CAiSE'08, pp. 213–227, 2008.

[22] J. L. de la Vara and J. Sànchez, *Business Process-Driven Requirements Analysis through Business Process Modelling: A Participative Approach*, Proc. BIS 2008, pp. 165–176, 2008.

[23] J. L. de la Vara, M. H. Fortuna, J. Sànchez, C. M. L. Werner and M. R. S. Borges, *A Requirements Engineering Approach for Data Modelling of Process-Aware Information Systems*, Proc. BIS 2009, LNBIP 21, pp. 133–144, 2009.

[24] X. Larrucea, A. B. G. Diez and J. X. Mansell, *Practical Model Driven Development Process*, Computer Science at Kent, 2004.

[25] R. Liu, K. Bhattacharya and F. Y. Wu, *Modeling Business Contexture and Behavior Using Business Artifact*, Proc. CAiSE'07 and WES 2007, LNCS 4495, pp. 324–339, 2007.

[26] 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.

[27] 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.

[28] B. Nuseibeh and S. Eastbrook, *Requirements Engineering: A Roadmap*, Proc. FOSE '00, pp. 35–46, 2000.

[29] Object Management Group, *Model Driven Architecture (MDA)*, http://www.omg.org/mda/, 2013.

[30] 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.

[31] M. Reichert and P. Dadam, *A Framework for Dynamic Changes in Workflow Management Systems*, Proc. DEXA'97, pp. 42–48, 1997.

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

[33] M. Reichert, *Process and Data: Two Sides of the Same Coin?*, Proc. CoopIS'12, LNCS 7565, pp. 2–19, 2012.

[34] A. Scheer, O. Thomas and O. Adam, *Process Modeling Using Event-driven Process Chains*, Process-Aware Information Systems, pp. 119–146, 2005.

[35] 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.

[36] S. White and D. Miers, *BPMN Modeling and Reference Guide*, Future Strategies Inc., 2008.

[37] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede and N. Russell, *On the Suitability of BPMN for Business Process Modelling*, Proc. BPM'06, LNCS 4102, pp. 161–176, 2006.