Towards Object-aware Process Support in Healthcare Information Systems

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

***Abstract*—The processes to be supported by healthcare information systems are highly complex, and they produce and consume a large amount of data. Besides, they require a high degree of flexibility. Despite their widespread adoption in industry, however, traditional process management systems (PrMS) have not been broadly used in healthcare environments so far. One major reason for this is the missing integration of processes with business data; i.e., business objects (e.g., medical orders or reports) are usually outside the control of a PrMS. By contrast, our PHILharmonicFlows framework offers an object-aware process management approach, which tightly integrates business objects and processes. In this paper, we use this framework to support a breast cancer diagnosis scenario. We discuss the lessons learned from this case study as well as requirements from the healthcare domain that can be effectively met by an object-aware process management system.**

***Keywords*-Process Management, Object-aware Process Management, Data-driven Process Execution.**

I. Introduction

Healthcare processes are characterized by their high complexity and the large amount of data they have to manage [1], [2]. The latter is usually represented through business objects like medical orders, medical reports, laboratory reports, and discharge letters. Since healthcare processes require the cooperation among different organizational units and medical disciplines [3], adequate process support is crucial. In this context, process management systems (PrMS) are typically the first choice for implementing process-aware information systems. However, despite their widespread adoption in industry, existing PrMS are not broadly used in healthcare environments [4]. One major reason for this deficiency is that contemporary PrMS are *activity-driven*. The processes are modeled in terms of "black-box" activities and their control-flow defines the order and constraints for executing these activities. However, activity-centric process modeling approaches like BPMN [5] or BPEL [6] present numerous limitations [29]: business data is typically treated as second-class citizen [7], [11]. For example, most PrMS only cover atomic data elements, which are needed for control flow routing and for supplying the input parameters of activities [8]. Business objects, in turn, are usually stored in external databases and are outside the control of the PrMS. Hence, integrated access to data and processes as crucial in the healthcare domain is missing; i.e., PrMS are unable to provide immediate access to important process information in case of unexpected events [26].

Regarding the execution of activity-driven PrMS, a process requires a number of activities to be completed in order to terminate successfully. Healthcare processes and their steps, in turn, depend on the availability of certain information [3]. For example, if a patient has a temperature of above 38,5°C, the doctor may have to prescribe a medicine to contain the fever. Consequently, the activation of an activity does not directly depend on the completion of other activities, but rather on the changes of business object attributes.

Typically, it is also not possible to squeeze processes from the healthcare domain into one monolithic process model [1]. In healthcare environments, there exists numerous processes depending on each other. For example, the distribution of a medicine in the hospital pharmacy may depend on the patient's treatment process which, in turn, may depend on his diagnosis process. The latter comprises diagnostic processes like blood tests and image examinations (or imaging encounters). To be applicable in a healthcare context, therefore, a PrMS must provide mechanisms for coordinating the interactions between interdependent processes.

Another challenge arises from the fact that activities are not only executed in the context of single process instances. Instead, they may be invoked at different levels of granularity comprising several process instances (of the same and of different type). A medical doctor, for instance, may examine one patient at a time, while a nurse prepares medications for several patients in one go. Finally, healthcare processes are highly dependent on medical knowledge as well as on specific case decisions [3], [25]. Thus, the type and order of invoked activities may vary from process instance to process instance. For this reason, healthcare processes cannot be "straight-jacketed" into a set of pre-defined activities [11], [24].

Generally, the described limitations of existing PrMS can be traced back to the missing integration of processes and data [28], [29]. To overcome these limitations, several approaches have already pioneered concepts for enabling data-driven process execution [11]–[13], [16], [17], [20], data-driven exception handling and process adaptation [17], [18], process coordination [9], [16], integrated access to data [11],

and process definition based on data behavior [14], [20]. However, none of them considers all identified limitations in a comprehensive and integrated way. In addition, some of these approaches do not make a difference between the modeling and execution of a process; i.e., they provide rich capabilities for process modeling, but do not explicitly take runtime issues into account.

Opposed to these approaches, PHILharmonicFlows targets at a comprehensive framework addressing the described limitations [28]. In addition, PHILharmonicFlows enforces a well-defined modeling methodology governing the object-centric specification of processes and being based on a formal operational semantics [30]. In this paper, we evaluate the applicability of PHILharmonicFlows framework to healthcare processes. To limit the scope, we focus on modeling issues in this paper. For this purpose, we present a breast cancer diagnosis procedure as performed at a Women's hospital.

Section II describes the medical scenario considered as well as the list of requirements to be supported by any PrMS in order to be applicable in a healthcare environment. In Section III, we model this scenario using the components provided by PHILharmonicFlows. Following this, in Section IV, we discuss how the requirements listed in Section II are met by the framework. Related work is discussed in Section V. Finally, Section VI concludes with a summary and an outlook.

II. Description of Healthcare Scenario

The healthcare scenario we consider is a breast cancer diagnosis process we obtained from a process handbook of a Women's hospital. As illustrated in Figure 1, this process comprises anamnesis, a physical examination (including the collection and confirmation of symptoms), a set of medical examinations (e.g., MRI, mammography, blood analysis), and a tumor biopsy. Some of these procedures are illustrated in Figure 2. We describe the different procedures using state charts. The latter are typically considered as intuitive modeling paradigm providing a natural view for end users [15].

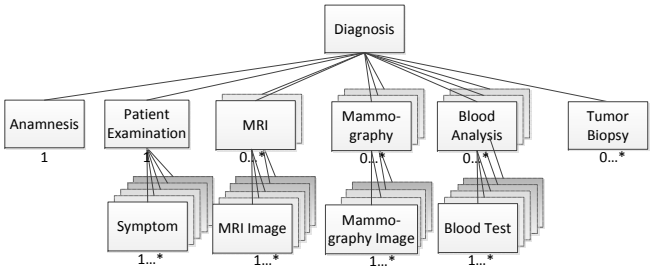

Figure 1. Objects involved in the breast cancer diagnosis process

During anamnesis (cf. Fig. 2b) the physician asks the patient specific questions (e.g., about her history of diseases, family diseases, or current medication). In the meanwhile, the doctor examines the patient and interviews her about the presence of any symptom (cf. Fig. 2d). The physician also asks the patient about breast nodules and he performs a physical examination in order to confirm or exclude the symptoms (cf. Fig. 2c). If the symptoms brought up by the patient are not confirmed during the physical examination, the presence of the tumor will be denied (cf. Fig. 2a). In this case the diagnosis process is finished. Otherwise, the doctor decides about a battery of examinations based on the symptoms confirmed.

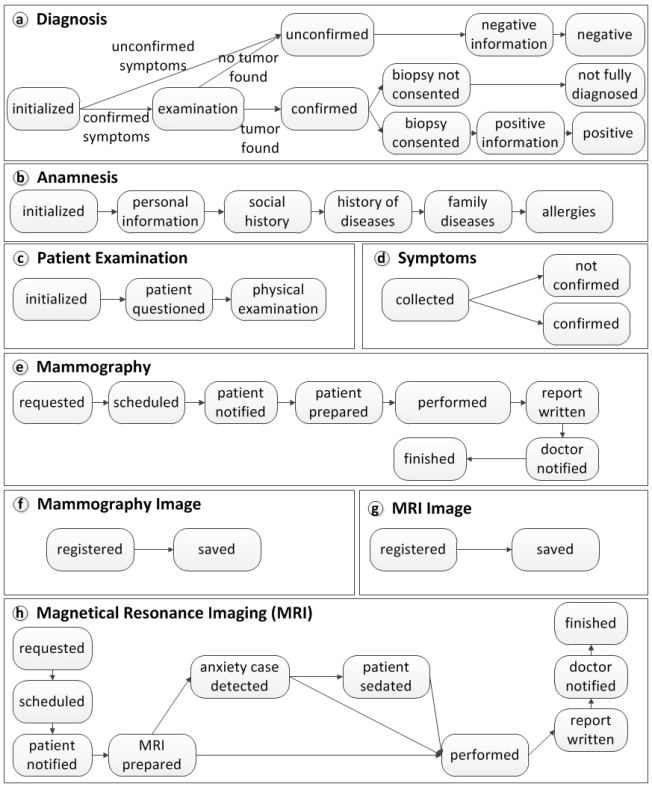

Figure 2. State diagrams for diagnosis, anamnesis, patient examination, symptom, mammography, and breast MRI examinations

One of the examinations required to detect the presence of a breast tumor or to exclude it is mammography (cf. Figure 2e). To perform this examination, the secretary of the radiology department must schedule it. At the day of the appointment the procedure is performed and the resulting images are stored in a database (cf. Figure 2f). The MRI examination comprises a similar process shown in Figure 2g. The images from both examinations are then analyzed by a specialized physician of the radiology department and are added to the respective medical reports. As opposed to the mammography examination, for which the equipment does not cause claustrophobia, during the MRI examination (cf. in Fig. 2h) the patient may have a case of elevated anxiety due to the enclosure of the MRI equipment. In such cases, the

radiology specialist being responsible for the examination must decide whether or not the patient shall be sedated before continuing with the procedure.

In the meanwhile, the doctor may request further examinations as, for example, another MRI examination or additional blood tests. Otherwise, if the existence of a tumor is confirmed, the doctor may want to biopsy this mass in order to confirm the malignancy of the tumor (cf. Fig. 2a). In this case, however, the consent of the patient is required. The biopsy report is returned to the physician who will inform the patient about the malignancy status of the tumor. Finally, the diagnosis process is finished as positive, confirming the presence of a breast tumor.

Though this diagnosis scenario seems to be rather simple, it indicates a number of requirements to be supported by the PrMS in order to be applicable to this healthcare environment:

Req. 1 - Data and process integration: Our scenario is composed of many procedures (e.g., anamnesis, searching symptoms, mammography, and MRI). The product of each of these procedures is data related to the patient's diagnosis; e.g., the data obtained when interviewing the patient in the context of the anamnesis. Respective data is not only important for keeping the patient's history updated or for registering all events for the purpose of auditing, they are also vital for process execution. Milestones reached during process execution do less depend on the execution of certain activities, but more on the availability of certain data. For example, a mammography medical report may only be written after having captured and stored the respective images. In addition, user decisions (typically based on available data) are fundamental for process execution. A radiology specialist, for example, may decide whether or not to sedate a patient during an MRI examination.

Req. 2 - Intense use of forms: Like most healthcare processes, the sketched scenario is characterized by a large number of medical forms to be filled by authorized medical staff (e.g., doctors, nurses, laboratory staff) with information being relevant to patient treatment. As example consider the information obtained when interviewing the patient about her anamnesis.

Req. 3 - Interacting processes: The breast cancer diagnosis process needs to interact with other processes (e.g., MRI); i.e., there are points in the diagnosis process where data from the MRI process is needed. In particular, these processes have *synchronization points*, where the further execution of a particular process instance depends on the data produced during the execution of one or several related instances. Respective synchronization points do not only correspond to one-to-one relationships; i.e., the execution of a particular process instance may also depend on multiple instances of another process type. In our example, the execution of the diagnosis process depends on the results of various examinations.

Req. 4 - Flexibility regarding process instantiation: Figure 1 also shows different cardinalities for the different procedures of the diagnosis process. These indicate whether or not the execution of the respective procedures is mandatory and whether they may be executed more than once. Mandatory procedures (e.g., *Anamnesis*, *Patient Examination*) have cardinality *1*, while optional ones (e.g., *MRI*, *Mammography*, *Blood Analysis*, *Tumor Biopsy*) have cardinality *0...**. The latter indicates that there are no restrictions regarding the number of instances of respective optional procedures. Based on the patient's case, doctors may decide which of these optional procedures shall be ordered and which not. Moreover, it is possible to request them more than once.

Req. 5 - Authorized user access: To ensure privacy, it is necessary that only authorized users may access patient data. In our scenario, for example, the secretary of the radiology department must not access information about the patient obtained during the anamnesis and she must not register symptoms of the patient. However, she may access the data related to the request and the scheduling of a mammography or an MRI examination. Besides, the permission to access data often depends on the progress of the process, which means that certain data should be only accessible at certain points during process execution. For example, the medical report of a mammography is accessible for the ordering doctor only when the procedure is completed and the report has been approved by the radiologist.

Req. 6 - Flexible data access: The system must provide the flexibility to users to access and modify data at arbitrary points during process execution. This is very important in order to be able to react to unexpected events. For example, in case of an emergency, the system must allow the doctor to access examination data before the medical report becomes available.

III. Case Study: Modeling with PHILharmonicFlows

In the previous section, we introduced fundamental requirements for adequately supporting healthcare processes. These indicate that healthcare processes fulfill the major characteristics of *object-aware processes* [31]:

1) *Object behavior:* The processing of individual object instances must be coordinated between different users, and valid attribute settings must be specified.
2) *Object interactions:* The behavior of individual objects must be coordinated with the one of related objects.
3) *Data-driven execution:* The progress of a process instance depends on business objects and their attribute values.
4) *Integrated access:* Authorized users should be able to access and manage process-related data objects at any point in time.
5) *Flexible activity execution:* Activities should be executable at different levels of granularity; e.g., it should

be possible that an activity may relate to one or to multiple process instances.

PHILharmonicFlows has recognized the need to offer flexible support for this kind of processes [28]. More precisely, it provides a comprehensive framework with components for both modeling and executing object-aware processes. To be able to define these processes in tight integration with data, the framework enforces a well-defined modeling methodology that governs the definition of processes at different levels of granularity. In this context, PHILharmonicFlows differentiates between *micro processes* and *macro processes* capturing either the *behavior* of single objects or the *interactions* among multiple objects.

The behavior of an object can be expressed by a number of possible *states*. Whether or not a particular state is reached depends on the values of object attributes. The interactions among objects, in turn, are enabled when involved objects reach certain states. Hence, object states serve as interface between micro and macro processes.

As prerequisite for integrated access to data and processes, a *data model* has to be defined. The latter enables the definition of object types as well as their attributes and relationships (including cardinalities) [30]. The data model depicted in Figure 1, for example, gives an overview of the object types being relevant in the context of our diagnosis process; i.e., there is one object type for each of the phases of the diagnosis process. Furthermore, Figure 5 illustrates the attributes of object type *Mammography*.

In PHILharmonicFlows, for each *object type* defined by the data model, one specific *micro process type* has to be defined. At runtime, object instances of the same and of different object types can be created at different points in time. In this context, the creation of a new object instance is directly coupled with the creation of a corresponding micro process instance. A *micro process type* expresses the behavior of the respective object type; i.e., it coordinates the processing of an object among different users and specifies what valid attribute settings are. Additionally, the cardinality of an object type in relation to other object types defines restrictions regarding the instantiation of micro process types and object types respectively. For example, in our case the cardinality of object type *Anamnesis* in relation to object type *Diagnosis* is *1*; i.e., there must be exactly one instance of object type *Anamnesis* for each *Diagnosis* instance. By contrast, it is not mandatory that there exists an instance of object type *Mammography* for each *Diagnosis* instance. However, it is up to the respective physician to initiate a specific number of instances of this examination as long as cardinality constraints are fulfilled. To meet Requirement 4 (cf. Section II), PHILharmonicFlows provides the flexibility to handle a varying number of instances of interrelated examinations. More precisely, it is up to the user to decide when and which examinations are required. We will see in the following, that using macro processes it becomes possible to define sophisticated execution and instantiation constraints in this context.

At micro process level, 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. Micro step types, in turn, may be connected with each other using *micro transition types*. To coordinate the processing of individual object instances among different users, several micro step types can be grouped into *state types*. The latter are then associated with one or more user roles being responsible to assign values to the required attributes. At runtime, a micro step can be reached if for the corresponding attribute a value is set. A state, in turn, can only be left 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 then may also depend on user decisions. For this purpose, micro transition types connecting micro step types belonging to different state types can either be categorized as *implicit* or *explicit*. Using *implicit micro transitions*, the target state is automatically activated as soon as all attribute values required by the previous state become available. *Explicit micro transitions*, in turn, additionally require a user commitment; i.e., users may decide whether or not the subsequent state should be activated. This way, users are enabled to still change corresponding attribute values even if all attribute values required to leave the state have been already set.

An example of a micro process type is illustrated in Figure 4. Object type *Mammography* and its respective micro process type are instantiated when the doctor orders a new *Mammography* examination. In order to *request* a *Mammography*, the (authorized) user must set the *order date*; i.e., to complete micro step *order_date* a value needs to be assigned to the corresponding attribute. In our example, the micro transition type between state types *requested* and *scheduled* is explicit (dotted line). This ensures that the doctor may still review the examination request before sending it to the secretary of the radiology department. In state *scheduled*, in turn, the *Secretary* must fill attributes *scheduled_date*, *scheduled_doctor* and *scheduled_room*. She further has to decide when to notify the patient about the scheduled appointment; i.e., the next state *patient notified* will only be activated when explicitly being confirmed by the *Secretary*.

A user decision, in turn, is required if a micro step type has more than one outgoing micro transition types. In this case, the responsible user has to decide which subsequent state shall be activated. Figure 3 shows a fragment of the *MRI* micro process type, where the radiology specialist must decide, in case of a patient's anxiety scenario, whether or not to sedate the patient. As we can observe in this example, the dotted lines indicate explicit micro transitions.

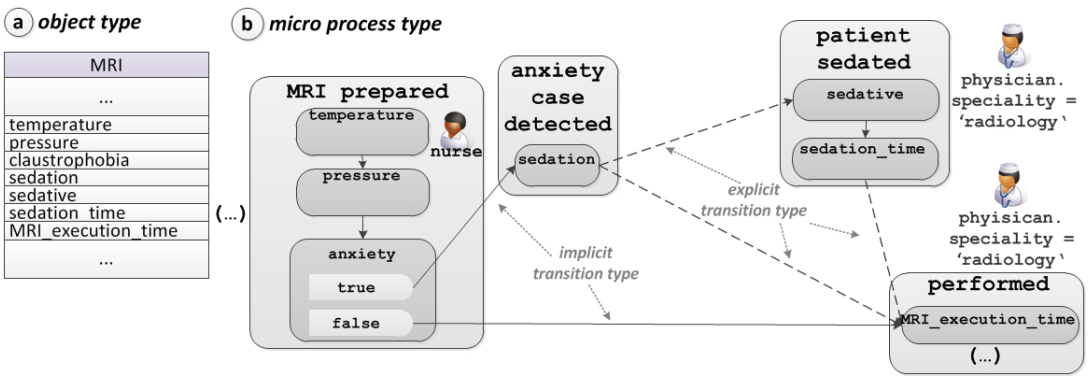

Figure 3. Partial view of the MRI micro process type

To enable coordination, user roles have to be assigned to the different states of a micro process type. Based on these role assignments, a corresponding *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 right to write an attribute can either be *mandatory* or *optional*. When initially generating the authorization table, the user role associated to a state type automatically receives mandatory write authorization for all attributes related to any micro step type of the respective state type. Optional data access may be additionally granted to user roles not being associated to the state type. This way, users currently not being involved in process execution are enabled to access process relevant data if desired.

Based on the authorization table, PHILharmonicFlows also automatically generates user forms. Which input fields are displayed to the respective user depends on the permissions he has in the currently activated state. If he only has the permission to read an attribute in a particular state, the form field will not be editable and be marked as read-only. A mandatory or optional attribute, in turn, is associated with an editable field. In particular, mandatory fields are highlighted in the respective form.

The concepts provided by PHILharmonicFlows to enable data authorization for micro process types are exemplified in Figure 5. It illustrates the authorization table of micro process type *Mammography*. In this example, state type *requested* has only one mandatory attribute *order_date* (marked as *MW* in the authorization table). This attribute has to be set by the physician requesting the examination. In addition, attributes *order_desired_date* and *order_observations* are optional (marked as *OW*). In state *scheduled*, the same physician may change the values of the aforementioned optional attributes, as opposed to the secretary of the radiology department. The latter may only read the values of these attributes (marked as *R*). However, she is allowed to write attributes *scheduled_date*, *scheduled_doctor* and *scheduled_room*, which, in turn, may only be read by the doctor.

Whether or not subsequent object states can be reached may not only depend on object attributes, but also on the states of other micro process instances. At runtime, for each object instance one corresponding micro process instance exists. As a consequence, a healthcare scenario may comprise dozens up to hundreds of micro process instances. Taking their various interdependencies into account, we obtain a complex *process structure*. In order to enable the interaction between these micro process instances, a *coordination mechanism* is required to specify the interaction points of the processes involved. For this purpose, PHILharmonicFlows automatically derives a state-based view for each micro process type. This view is then used for modeling macro process types. A *macro process type* refers to parts of the data structure and consists of both *macro step types* and *macro transitions types* between the latter. As opposed to traditional process modeling approaches, where process steps are defined in terms of black-box activities, a macro step type always refers to an object type together with a corresponding state type. The macro process type resulting for our example from Figure 6 illustrates this. The process begins with the instantiation of object type *Diagnosis*, which triggers the initiation of its micro process. Then, object type *Anamnesis* is instantiated (i.e., the responsible doctor receives a corresponding item in his worklist) and its micro process instance is initialized. During *Patient Examination*, it is possible to have the *symptoms* collected, which are then confirmed after the *physical examination* has taken place. If the symptoms are not confirmed, the diagnosis will be finished as negative, indicating that no tumor was found. Otherwise, the diagnosis process continues with requesting imaging encounters. It is important to note that for one primary examination, there may be more than one symptom collected.

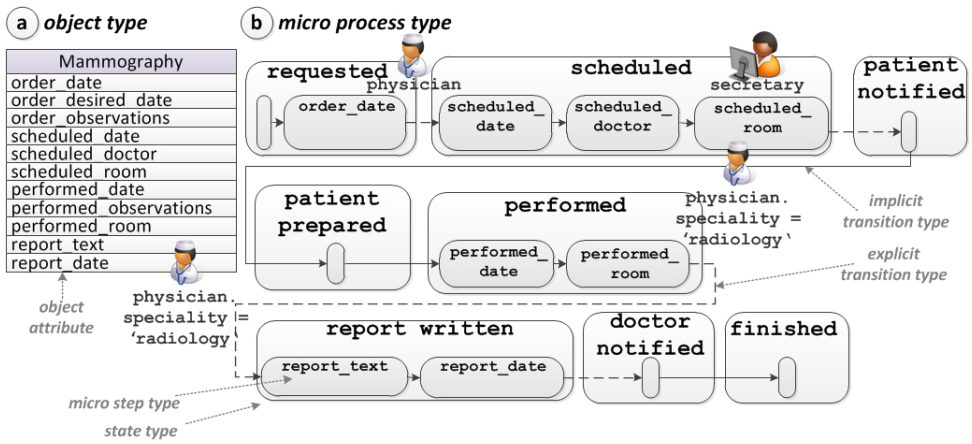

Figure 4. Mammography micro process type

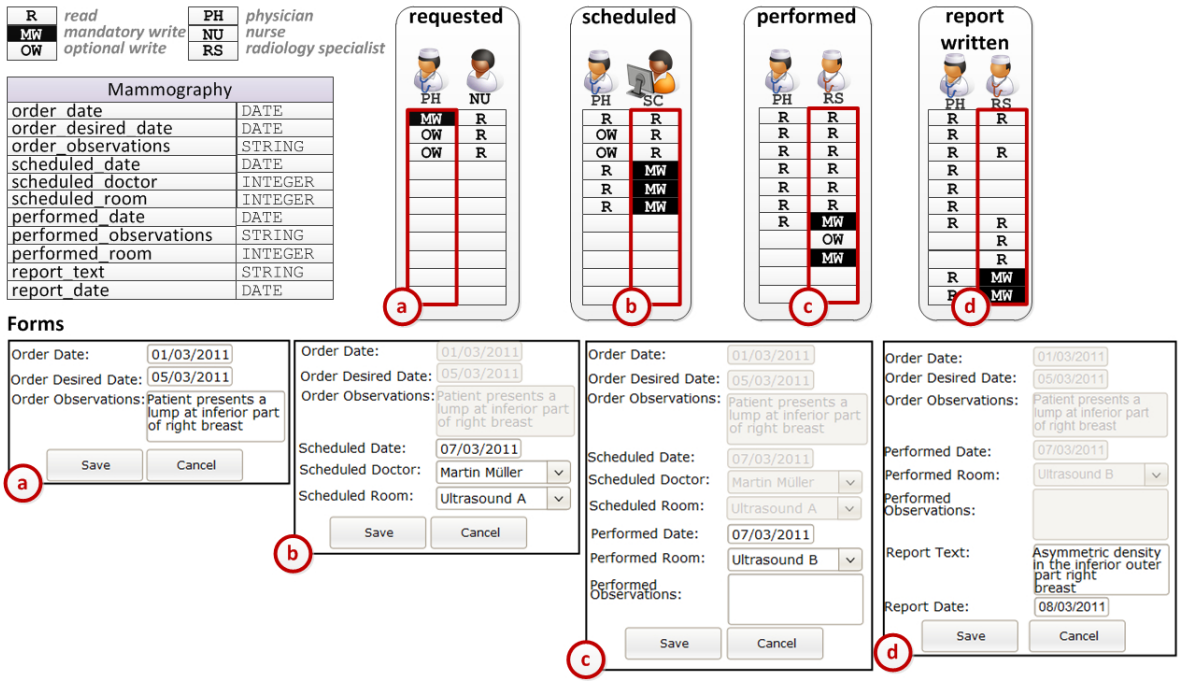

Figure 5. Authorization table and forms of the Mammography micro process type

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

To take the dynamically evolving number of object instances as well as the asynchronous execution of corresponding micro process instances into account, for each macro transition a corresponding *coordination component* needs to be defined. For this purpose, PHILharmonicFlows takes the relationship between the object type of the source macro step type and the one of the target macro step type into account. To cover this, the framework automatically structures the data model into different *data levels*. All object types not referring to any other object type are placed on the top level (Level #1). Generally, any other object type is always assigned to a lower data level as the object types it references. As illustrated in Figure 7, in our case study, object type *Diagnosis* is at the top level, while all the examinations are placed at a lower level. For example, images refer to respective examinations (i.e., imaging encounters). Hence,

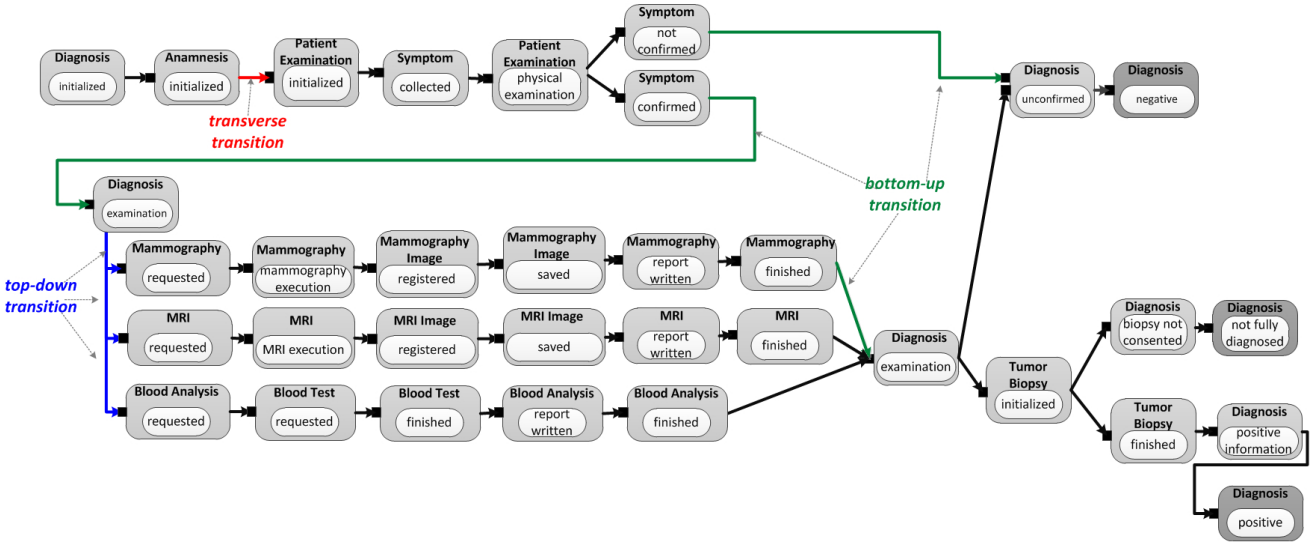

Figure 6. A macro process type coordinating the interactions among the different micro process types

they are placed at Level #3. In this paper, we do not discuss self-references and cyclic relations, but they are considered by PHILharmonicFlows framework.

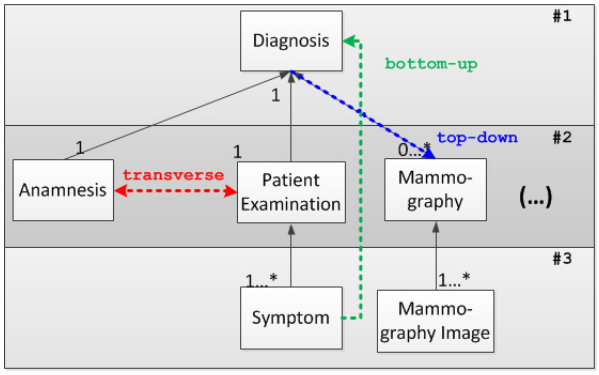

Figure 7. Different kinds of relationships between object types

By organizing the object types of the data model into different levels, PHILharmonicFlows automatically categorizes macro transitions either as *top-down* or as *bottom-up* (cf. Figure 7). Furthermore, if the object types of the source and sink macro state refer to a common higher-level object type, the macro transition is categorized as *transverse*. For macro transitions types connecting macro step types that refer to the same object type no coordination component is needed. These transitions are denoted as *self-transitions*. For all other ones, the coordination component required depends on the type of the respective macro transition. A *top-down transition* characterizes the interaction from an upper-level object type to a lower-level one. Here, the execution of a varying number of micro process instances depends on one higher-level micro process instance. In this context, a so called *process context type* must be assigned to the respective macro transition type. Due to lack of space, we do not go into details. We also do not discuss transverse macro transition types here. A *bottom-up transition*, in turn, characterizes an interaction from a lower-level object type to an upper-level one. In this case, the execution of one higher-level micro process instance depends on the execution of several lower-level micro process instances of the same type. For this reason, each bottom-up transition requires an *aggregation component* for coordination. For this purpose, PHILharmonicFlows provides *counters* managing the total number of lower-level micro process instances and the number of micro process instances for which the state corresponding to the source macro step type is currently activated. To enable asynchronous execution, additional counters for reflecting the number of micro process instances currently being before or after the respective state or being skipped are provided. These counters can be used for defining aggregation conditions enabling the higher-level micro process instance to activate the state. As illustrated in Figure 8, the *Diagnosis* process is finished in state *negative* if no *Symptoms* is confirmed. The aggregation condition for this case (`#IN=#ALL`) indicates that all micro process instances of object type *Symptom* must reach state *not confirmed* in order to activate the state *unconfirmed* from the respective instance of micro process type *Diagnosis*. In this example, we illustrate how such counters work. As illustrated in Figure 8, there are three micro process instances of *Symptom* related to one micro process instance of *Diagnosis*. In this example, the counter indicates that two of the running instances of *symptom* have already reached state *not confirmed* (`#IN=2`), while instance has not yet reached this state (`#BEFORE=1`). When all three instances reach this state (i.e., the condition

defined in the aggregation is met), state *unconfirmed* is activated at the respective *Diagnosis* instance.

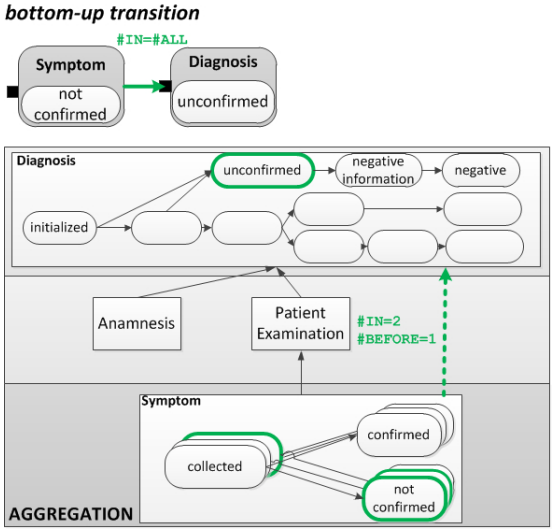

Figure 8. Aggregation example

The *runtime environment* provides data- as well as process-oriented views to end-users. In particular, authorized users may invoke activities for accessing data at any point in time as well as activities needed in order to proceed with the execution of micro process instances. In this context, the formal operational semantics of PHILharmonicFlows enables a well-defined execution logic and additionally enables us to automatically generate most end-user components of the runtime environment (e.g., tables giving an overview on object instances, user worklists, and form-based activities).

Insights into the operational semantics provided by PHILharmonicFlows can be found in [30].

IV. Discussion

In Section II, we have introduced a healthcare scenario, which we have then modeled in Section III using the PHILharmonicFlows framework. In this section, we discuss how the requirements posed by the healthcare scenario are covered.

Req. 1 - Data and process integration: The well defined modeling methodology of PHILharmonicFlows ensures that each procedure (e.g., anamnesis, primary examination, mammography, etc.) is modeled from a data-oriented perspective (i.e., by using object types) as well as from a process-oriented one (i.e., by using micro process types). Hence, all the data produced by such procedures is stored and managed without need to access external databases during the execution of black-box activities. This also enables users to access and manage process-related object instances at any point in time (assuming proper authorization) and not only in the context of upcoming mandatory activities.

Req. 2 - Intense use of forms: Based on authorization tables, PHILharmonicFlows automatically generates user forms. For this purpose, it takes the currently activated state of a micro process instance as well as the user and his data access permissions into account. Each form comprises fields corresponding to read and write permissions of respective attributes. Moreover, in PHILharmonicFlows, object instances and activities are not strictly linked with each other. For example, it is also possible to execute a particular form in relation to a collection of object instances of the same object type. Here, entered attribute values are assigned to all selected object instances in one go. In addition, a user may invoke additional object instances of different (related) types. When generating corresponding forms, the currently activated states of these instances as well as the permissions assigned to the respective user in these states are taken into account.

Req. 3 - Interacting processes: As discussed in Section III, this requirement is met by PHILharmonicFlows using the macro process component. Using macro step types it becomes possible to define the required synchronization points. At runtime, it is possible to execute the individual micro process instances asynchronously to each other as well as asynchronously to the micro process instances of other types. In addition, it is possible to instantiate them at different points in time. Consequently, the resulting process structure comprises a varying number of interrelated micro process instances being in different execution states. For this reason, each macro transition type can be further specialized using different coordination components. The choice of the latter depends on the relation between the corresponding object types within the overall data structure. This way, not only the asynchronous execution but also the different cardinalities between different sets of dependent micro process instances are considered.

Req. 4 - Flexibility regarding process instantiation: Using PHILharmonicFlows it becomes possible to consider a dynamic number of inter-related micro process instances. Taking the defined cardinality constraints into account, users can freely decide which and how many micro process instances shall be created. If the minimum cardinality is not met, PHILharmonicFlows automatically assigns a corresponding mandatory activity to the worklists of responsible users demanding the creation of new instances of the respective micro process type. Opposed to this, if the maximum cardinality is reached, PHILharmonicFlows prohibits the creation of additional micro process instances. By specifying the cardinality of each object type, it is possible to define which of them must be instantiated (cardinality *1*) and which ones are optional (cardinality *0...*). This enables qualified staff members to request examinations at arbitrary points of the diagnosis process and to react on unexpected events (e.g., drug prescription in case of intense fever).

Req. 5 - Authorized user access: The *authorization table*

enables the level of data privacy required by healthcare processes. For each micro process type, it is possible to define which attributes can be written or read by a particular user (role) according to the currently activated micro process state. PHILharmonicFlows ensures that no data is written or read by unauthorized users. Since each state type has one user role associated to it, the authorization table automatically ensures that this same role owns the corresponding data permissions; i.e., the role has mandatory write permission to the attributes associated with the micro step types, which belong, in turn, to the state type.

Req. 6 - Flexible data access: As opposed to traditional PrMS, PHILharmonicFlows presents two different views to the end-users: a process-oriented (i.e., worklists) and a data-oriented (i.e., overview tables listing individual object instances together with their attribute values). The latter enables the access to data at any point in time by authorized users. Thus, data access does not depend on the activation of an upcoming activity; i.e., the data can be accessed beyond the context of a particular mandatory activity.

V. Related Work

Healthcare is a challenging domain for process support, since it comprises structured and unstructured processes whose support requires a high degree of flexibility. There are many researchers showing interest in this area [11], [23]. The case-handling paradigm focuses on administrative processes and, like PHILharmonicFlows, aims at data and process integration by managing the data inside the "case" scope and by enabling form-based activities. It also intends to increase the degree of flexibility by providing access to information outside the context of an activity. However, data is only provided in terms of atomic elements and can be read by all users involved in the case. Furthermore, there is no full support regarding interactions among different cases.

Interactions among process fragments, in turn, are supported by the Proclets approach [1], [9]. However, data is managed outside the scope of the process management system and can only be accessed when an activity is being executed.

The document-based workflow approach α-flow [21], [22] incorporates workflow semantics into the documents involved. Such documents are edited and viewed taking the separation of responsibilities and inter-institutional collaboration into account.

For more details about existing data-aware process management approaches, we refer readers to [31].

VI. Summary & Outlook

We analyzed a breast cancer diagnosis scenario. By modeling it with PHILharmonicFlows we studied how effectively this framework covers the semantics of healthcare processes. First, we elicited a list of requirements not adequately met by traditional process management systems in this context. Following this, we modeled the considered scenario by using components of the PHILharmonicFlows framework. Finally, we discussed the effectiveness of this approach and showed how it covers the requirements of healthcare processes.

Healthcare processes are knowledge-intensive and need a high level of flexibility in order to allow qualified staff members to flexibly react to unexpected events. Compared to other data-oriented approaches, in a very effective way PHILharmonicFlows covers the requirements posed by healthcare processes. By tightly integrating data and processes, our approach enables an environment where data drives the process and permits a higher degree of flexibility allowing data access outside the context of black-box activities as well. Furthermore, the distinction between two levels of granularity permits the interaction of processes being executed independently. It further enables flexibility by allowing users to decide which processes to instantiate when.

Like in activity-centered approaches [32], schema evolution is a complex and error-prone task to be accomplished for object-aware processes as well. We are working on an extension of the framework to support it; i.e., a mechanism to manage and to apply changes in object-aware processes as well as their running instances. Since all components of the framework are tightly integrated, the mechanism must take into account that each change operation may affect more than one component, causing a cascading effect. Thus, the mechanism must be able to detect such interdependencies between components and to assist the user to apply the changes in the process without affecting correctness and compliance.

Acknowledgment

The authors would like to acknowledge financial support provided by the *Deutscher Akademischer Austausch Dienst (DAAD)*.

References

[1] R. S. Mans, N. C. Russell, W. M. P. van der Aalst, A. J. Moleman, and P. J. M. Bakker, *Proclets in Healthcare*, Eindhoven: BPM Center, 36, 2009.

[2] M. Reichert, *What BPM Technology Can Do for Healthcare Process Support*, Proc. 13th Conf. on Artificial Intelligence in Medicine (AIME'11), LNAI 6747, 2–13, 2011.

[3] R. Lenz and M. Reichert, *IT Support for Healthcare Processes – Premises, Challenges, Perspectives*, Data & Knowledge Engineering, 61(1), 39–58, 2007.

[4] P. Dadam, M. Reichert, and K. Klaus, *Clinical Workflows - The Killer Application for Process-oriented Information Systems?*, Proc. 4th Int'l Conf. on Business Information Systems (BIS'00), 36–59, 2000.

[5] Object Management Group, *Business Process Model and Notation (BPMN)*, version 2.0, January 2011, http://www.omg.org/spec/BPMN/2.0, 2011.

[6] M. Juric, *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition*, Packt Publishing, 2006.

[7] D. Cohn and R. Hull, *Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes*, IEEE Data Engineering Bull., 32(3), 3–9, 2009.

[8] M. Reichert and P. Dadam, *A Framework for Dynamic Changes in Workflow Management Systems*, Proc. 8th Int'l Workshop on Database and Expert Systems Applications (DEXA'97), 42–48, 1997.

[9] W. M. P. van der Aalst, P. Barthelmess, C. Ellis, and J. Wainer, *Workflow Modeling Using Proclets*, Proc. 5th Int'l Conf. on Cooperative Information Systems (CoopIS'00), LNCS 1901, 198–209, 2000.

[10] W. M. P. van der Aalst and K. van Hee, *Workflow Management: Models, Methods, and Systems*, MIT Press, 2004.

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

[12] B. Mutschler, B. Weber, and M. Reichert, *Workflow Management versus Case Handling: Results from a Controlled Software Experiment*, Proc. 23rd Annual ACM Symposium on Applied Computing (SAC'08), 82–89, 2008.

[13] C. Guenther, M. Reichert, and W. M. P. van der Aalst, *Supporting Flexible Processes with Adaptive Workflow and Case Handling*, Proc. 3rd. IEEE Workshop on Agile Cooperative Process-aware Information Systems (ProGility'08), IEEE Computer Society Press, 229–234, 2008.

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

[15] R. Liu, K. Bhattacharya, and F. Y. Wu, *Modeling Business Contexture and Behavior Using Business Artifact*, Proc. 19th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'07), LNCS 4495, 324-339, 2007.

[16] D. Müller, M. Reichert, and J. Herbst, *Data-Driven Modeling and Coordination of Large Process Structures*, Proc. 19th Int'l. Conf. on Cooperative Information Systems (CoopIS'07), LNCS 4803, 131–149, 2007.

[17] D. Müller, M. Reichert, and J. Herbst, *A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures*, Proc. 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08), LNCS 5074, 48–63, 2008.

[18] S. Rinderle and M. Reichert, *Data-Driven Process Control and Exception Handling in Process Management Systems*, Proc. 18th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'06), LNCS 4001, 273–287, 2006.

[19] G. M. Redding, M. Dumas, A. Hofstede, and A. Iordachescu, *A Flexible, Object-centric Approach for Business Process Modeling*, Proc. IEEE Int'l Conf. on Service-Oriented Computing and Applications (SOCA), 1–11, 2009.

[20] I. Vanderfeesten, H. A. Reijers, and W. M. P. van der Aalst, *Product-Based Workflow Support: Dynamic Workflow Execution*, Proc. Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08), LNCS 5074, pp. 571–574, 2008.

[21] C. P. Neumann and R. Lenz, *alpha-Flow: A Document-based Approach to Inter-Institutional Process Support in Healthcare*, Proc. 3rd. Int'l Workshop on Process-oriented Information Systems in Healthcare (ProHealth 2009), LNBIP 43, 569–580, 2009.

[22] C. P. Neumann and R. Lenz, *The alpha-Flow Use-Case of Breast Cancer Treatment – Modeling Inter-Institutional Healthcare Workflows by Active Documents*, Proc. IEEE Int'l Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2010), 17–22, 2010.

[23] N. Mulyar, M. Pesic, W. M. P. van der Aalst, and M. Peleg, *Declarative and Procedural Approaches for Modelling Clinical Guidelines: Addressing Flexibility Issues*, Proc. 1st Int'l Workshop on Process-oriented Information Systems in Healthcare (ProHealth 2007), LNCS 4928, 335–346, 2007.

[24] B. Silver, *Case Management: Addressing Unique BPM Requirements*, BPMS Watch, 1–12, 2009.

[25] N. Gronau and E. Weber, *Management of Knowledge Intensive Business Processes*, Proc. 2nd Int'l Conf. on Business Process Management (BPM'04), LNCS 3080, 163–178, 2004.

[26] V. Künzle and M. Reichert, *Towards Object-aware Process Management Systems: Issues, Challenges and Benefits*, Proc. 10th Int'l Workshop on Business Process Modeling, Development, and Support (BPMDS'09), LNBIP 29, 197–210, 2009.

[27] V. Künzle and M. Reichert, *Integrating Users in Object-aware Process Management Systems: Issues and Challenges*, Proc. 5th Int'l Workshop on Business Process Design (BDP'09), LNBIP 43, 29–41, 2009.

[28] 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), 205–244, 2011.

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

[30] V. Künzle and M. Reichert, *A Modeling Paradigm for Integrating Processes and Data at the Micro Level*, Proc. 12th Int'l Working Conf. on Business Process Modeling, Development and Support (BPMDS'11), LNBIP 81, 201–215, 2011.

[31] V. Künzle and M. Reichert, *Striving for Object-aware Process Support: How Existing Approaches Fit Together*, Proc. 1st Int'l Symposium on Data-driven Process Discovery and Analysis (SIMPDA'11), 2011.

[32] M. Reichert, S. Rinderle-Ma, and P. Dadam, *Flexibility in Process-aware Information Systems*, LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) 2, LNCS 5460, 115–135.