Object-aware Process Support in Healthcare Information Systems: Requirements, Conceptual Framework and Examples

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

***Abstract*—The business processes to be supported by healthcare information systems are highly complex, producing and consuming a large amount of data. Besides, the execution of these processes requires 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 drawback is the missing integration of business processes and business data in existing PrMS; i.e., business objects (e.g., medical orders, medical reports) are usually maintained in specific application systems, and are hence outside the control of the PrMS. As a consequence, most existing PrMS are unable to provide integrated access to business processes and business objects in case of unexpected events, which is crucial in the healthcare domain. In this context, the PHILharmonicFlows framework offers an innovative object-aware process management approach, which tightly integrates business objects, functions, and processes. In this paper, we apply this framework to model and control the processes in the context of a breast cancer diagnosis scenario. First, we present the modeling components of PHILharmonicFlows framework applied to this scenario. Second, we give insights into the operational semantics that governs the process execution in PHILharmonicFlows. Third, we discuss the lessons learned in this case study as well as requirements from the healthcare domain that can be effectively handled when using an object-aware process management system like PHILharmonicFlows. Overall, object-aware process support will allow for a new generation of healthcare information systems treating both data and processes as first class citizens.**

***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 [22], [33], [20]. The latter is usually represented in terms of business objects like, for example, medical orders, medical reports, laboratory reports, and discharge letters. Since healthcare processes require the cooperation among different organizational units and medical disciplines [19], cross-departmental process support becomes crucial.

A. Problem Statement

In this context, process management systems (PrMS) are typically the first choice for implementing and maintaining process-aware information systems. However, despite their widespread adoption in industry, existing PrMS have not been broadly used in healthcare environments so far [8]. One major reason for this deficiency is the fact that contemporary PrMS are *activity-driven*; i.e., 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 [29] or BPEL [18] present numerous limitations [14]: business data is typically treated as second-class citizen [4], [7]. Moreover, most PrMS only cover atomic data elements, which are needed for control flow routing and for supplying the input parameters of activities with respective values [31]. In turn, business objects are usually maintained and stored in external databases and are outside the control of the PrMS. Hence, integrated access to processes and data, which is crucial for any healthcare process support, is missing. Particularly, contemporary PrMS are unable to provide immediate access to important process and business information in case of unexpected events [11].

Regarding the execution of activity-driven PrMS, a process requires a number of activities to be processed in a certain order and be completed to terminate successfully. In turn, healthcare processes and their steps depend on the availability of certain information [19]. For example, if the temperature of a patient is above 38°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 changes of business object attributes.

Typically, it is also not possible to squeeze healthcare processes into one monolithic process model [22]. 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 procedures 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. Thereby, respective coordination mechanisms must take the role of business objects into account.

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, that means, different kinds of working styles need to be supported.

Finally, healthcare processes highly depend on medical knowledge as well as on case-specific decisions [19], [9]. Thus, the type and order of invoked activities may vary from process instance to process instance. In particular, healthcare processes cannot be "straight-jacketed" into a set of predefined activities [4], [38].

B. Contribution

Generally, the described limitations of existing PrMS can be traced back to the missing integration of processes and data [13], [14]. To overcome these limitations, several approaches have pioneered certain concepts for enabling data-driven process execution [4], [26], [10], [23], [24], [39], data-driven exception handling and process adaptation [24], [35], process coordination [2], [23], integrated access to data [4], and process definition based on data behavior [5], [39]. However, none of these approaches considers all identified limitations in a comprehensive and integrated way. Furthermore, many existing approaches solely focus on process modeling or 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 all described limitations (and many others) [13], [17]. In addition, PHILharmonicFlows enforces a well defined modeling methodology governing the object-centric specification of processes based on a precise and formal operational semantics [15], [17]. In this paper, we evaluate the applicability of PHILharmonicFlows framework to healthcare processes. For this purpose, we present a breast cancer diagnosis procedure as performed at a Women's hospital. This paper significantly extends the work we presented in [1]. In particular, besides the modeling issues of the framework, we present the operational semantics governing process execution during runtime as well as some screenshots of our prototype, built as proof-of-concept.

Section II describes the medical scenario we consider and elaborate the fundamental requirements that any PrMS supporting corresponding healthcare processes must meet. In Section III, we model this scenario using the components provided by the PHILharmonicFlows framework. The operational semantics for executing processes in PHILharmonicFlows is presented in Section IV. Following this, Section V discusses how the identified requirements are met by the framework. Related work is discussed in Section VI. Finally, Section VII concludes with a summary and an outlook.

II. Healthcare Scenario

The healthcare scenario we consider is a breast cancer diagnosis process we obtained from a process handbook of a Women's hospital [36], [37]. As illustrated in Figure 1, this process comprises an anamnesis, a physical examination (including the collection and confirmation of symptoms), a set of medical examinations (e.g., MRI, mammography, and blood analysis), and a tumor biopsy. Some of these procedures are illustrated in Figure 2. We describe the different procedures using state charts, which are considered as an intuitive modeling paradigm providing a natural view for end users [21].

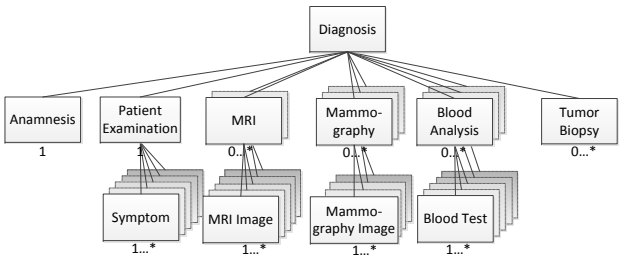

Figure 1. Objects involved in the breast cancer diagnosis process

During anamnesis (see Figure 2b) the doctor 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 (see Figure 2d). The doctor also asks the patient about breast nodules and he performs a physical examination in order to confirm or exclude the symptoms (see Figure 2c). If the symptoms brought up by the patient are not confirmed during the physical examination, the presence of the tumor will be denied (see Figure 2a). In this case, the diagnosis process is finished. Otherwise, the doctor decides about a battery of examinations based on the symptoms confirmed.

One of the examinations, required to detect the presence of a breast tumor or to exclude it, is mammography (see 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 (see Figure 2f). The MRI examination requires a similar process as shown in Figure 2g. The images from both examinations are then analyzed by a specialized doctor 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 (see Figure 2h) the patient may have a case of elevated anxiety due to the enclosure of the MRI equipment. In such cases,

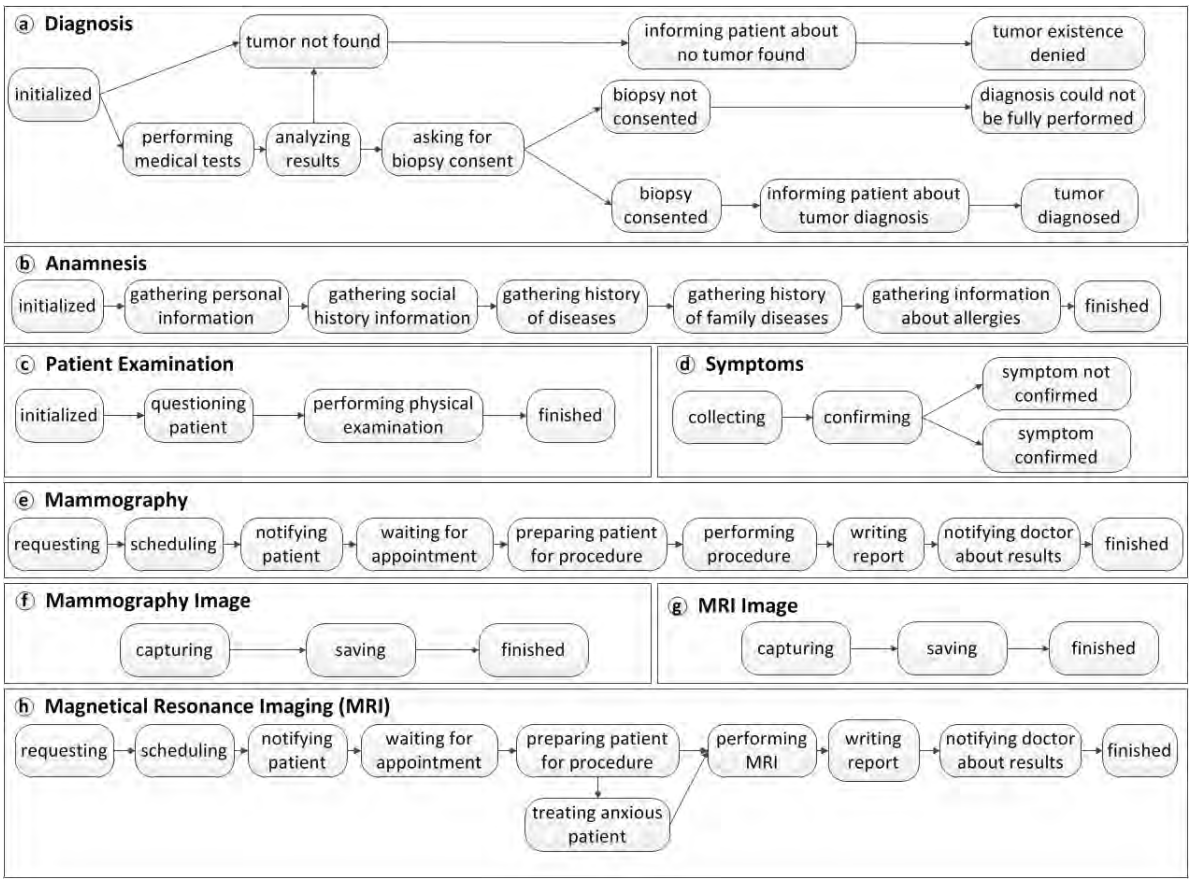

Figure 2. State diagrams for diagnosis, anamnesis, patient examination, symptom, mammography, and breast MRI examinations

the radiology specialist 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 like, 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 (see Figure 2a). In this case, however, the consent of the patient is required. The biopsy report is returned to the doctor who will then 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.

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

Requirement R1 (Data and process integration): Our scenario is composed of many procedures, including an anamnesis, search for 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 during 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 in respect to process execution. In particular, the milestones reached during process execution do less depend on the execution of specific 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, which are typically based on available data, are fundamental for process execution. For example, a radiology specialist may decide whether or not to sedate a patient during an MRI examination.

Requirement R2 (Intense use of forms): Like for other healthcare scenarios, the sketched procedure comprises a large number of medical forms to be filled by authorized medical staff (e.g., doctors, nurses, laboratory staff) with any information being relevant for patient treatment. For example, consider the information obtained when interviewing the patient during the anamnesis.

Requirement R3 (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 at which data from the MRI process is required. In particular,

these processes have *synchronization points*, at which the further execution of a particular process instance depends on the data produced during the execution of one or several related process instances. Such synchronization points do not only correspond to one-to-one relationships. Additionally, the execution of a particular process instance may depend on the execution of multiple instances of another process type as well. Regarding our example, the execution of the diagnosis process depends on the results of various examinations.

Requirement R4 (Flexibility regarding process instantiation): Figure 1 indicates different cardinalities for the various procedures of the diagnosis process. These cardinalities 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 the instances of respective optional procedures. Depending on the concrete case of a patient, doctors may decide which of these optional procedures shall be ordered and which not. Finally, note that it is possible to order them more than once.

Requirement R5 (Authorized user access): To meet security constraints and ensure privacy, only authorized users are allowed to 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 medical order or the scheduling of a mammography or an MRI examination. Besides, the permission to access data may depend on the progress of the process, which means that certain data shall be only accessible at certain points during process execution. For example, the medical report of a mammography may be accessible for the doctor who placed the order solely when the procedure has been completed and the report been approved by the radiologist.

Requirement R6 (Flexible data access): Any healthcare information system must provide the flexibility to its users to access and modify business data at arbitrary points during process execution. Amongst others, this is extremely important in order to be able to properly react to unexpected events [34]. 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 Healthcare Processes with PHILharmonicFlows

In the previous section, we introduced fundamental requirements for adequately supporting healthcare processes. In particular, the requirements imposed by healthcare processes can be easily linked to the major characteristics of *object-aware processes* [16], [34]:

1) *Object behavior:* This first characteristic covers the processing of individual object instances. More precisely, for each object type a separate process definition must be provided. At runtime, the latter is then used for coordinating the processing of individual object instances among different users. In addition, it must be specified in which order and by whom the attributes of a particular object instance shall be (mandatorily) written, and what valid attribute settings (i.e., attribute values) are. At runtime, the creation of an *object instance* is directly coupled with the creation of its corresponding *process instance*. In this context, it is important to ensure that mandatory data is provided during process execution; i.e., during the processing of the object instances. For this reason, object behavior should be defined in terms of *data conditions* rather than based on black-box activities.
2) *Object interactions:* The behavior of a particular object must be coordinated with the one of other related objects. The related object instances may be created or deleted at arbitrary point in time, emerging a *complex data structure*. The latter dynamically evolves during runtime, depending on the types and numbers of created object instances. Furthermore, individual object instances of the same type may be in different processing states at a certain point in time. More precisely, it must be possible to execute individual process instances (of which each corresponds to the processing of a particular object instance) in a loosely coupled manner; i.e., concurrently to each other and synchronizing their execution where needed, taking semantic object relations and cardinality constraints into account.
3) *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. 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 required data is already available (e.g., it may be optionally provided by authorized users before the respective mandatory activity becomes enabled), these activities will be automatically skipped when becoming activated. Furthermore, users shall be able 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*. 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 way as well.
4) *Flexible activity execution:* For creating object instances and changing object attribute values, *form-*

based activities can be used. Respective user forms comprise *input fields* (e.g., text fields or check-boxes) for writing selected attributes and *data fields* for reading attributes of object instances. In this context, however, different users may prefer different work practices. Therefore, 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 object process instances.

5) *Integrated access:* Authorized users should be able to access and manage process-related data objects at any point of time. More precisely, *permissions* for creating and deleting object instances, as well as for reading and writing their attributes need to be defined. However, attribute changes contradicting specified object behavior must be prevented. Which attributes may be written or read by a particular (form-based) activity not only depends on the user invoking this activity, but also on the progress of the corresponding process instance. While certain users must execute an activity mandatorily in the context of a particular object instance, others might be authorized to optionally execute this activity; i.e., a distinction is made between mandatory and optional permissions. Furthermore, for object-aware processes, the selection of actors usually not only depends on the activity to be performed, but also on the object instances processed by this activity. In this context, the relationships between users and object instances must be taken into account.

PHILharmonicFlows has recognized the need to offer flexible support for this kind of processes [13]. More precisely, it provides a comprehensive and well-grounded framework with components for modeling, executing, and monitoring object-aware processes (see Figure 3). To be able to define these processes in tight integration with data, PHILharmonicFlows 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.

First of all, the behavior of a single object may be expressed in terms of a number of possible *states*. Furthermore, whether or not a particular state will be reached at certain time 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 well.

Data Model (Figure 3a): As prerequisite for providing integrated access to data and processes, a *data model* must be provided. The latter must comprise object types (including object attributes) and object type relationships (including cardinalities) [15]. For example, data model depicted in Figure 1 gives an overview of the object types 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 6 illustrates the attributes of object type *Mammography*.

Micro Process (Figure 3b): In PHILharmonicFlows, for each *object type* of the data model, a particular *micro process type* must 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 one of a corresponding micro process instance. In general, a *micro process type* expresses the behavior of the respective object type (e.g., *Mammography*); i.e., it coordinates the processing of an object among different user roles (e.g., nurse or doctor) 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., for each *Diagnosis* instance, there must be exactly one instance of object type *Anamnesis*. 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 doctor to create several instances of this examination as long as cardinality constraints are met. To meet requirement R4 (see Section II), PHILharmonicFlows provides the flexibility to handle a varying number of instances of interrelated examinations. More precisely, it is up to the doctor to decide when and which examinations are required. We will show later, that so-called macro processes enable a coordinated execution of dependent micro process instances.

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 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 responsible for assigning values to the required attributes. At runtime, a micro step can be reached if for the corresponding attribute a value is set. In turn, a state may only be left if, for all attributes associated with its micro steps of this state, respective values are set. Whether or not the subsequent state of the micro process is then immediately activated may depend on user decisions. In this context, micro transition types, which connect micro step types belonging to different state types, are either categorized as *implicit* or *explicit*. Regarding *implicit micro transitions*, the target state will be automatically activated as soon as all attribute values required by the previous state

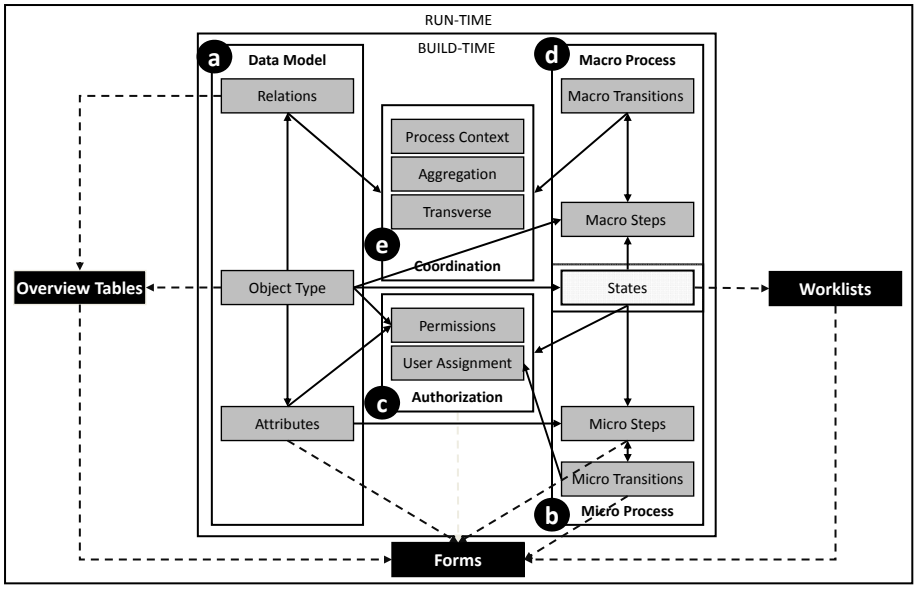

Figure 3. Overview of the PHILharmonicFlows Framework

are available. In turn, *explicit micro transitions* additionally require a user commitment; i.e., users may decide when the subsequent state shall be activated. This way, users still may change attributes even if all attribute values required to leave the state have been already set.

An example of a micro process type is depicted in Figure 5. Object type *Mammography* and its respective micro process type are instantiated when the doctor requests a new *mammography* examination. For *requesting* a *mammography*, the (authorized) user must set the *request date*; i.e., to complete micro step *request_date* a value needs to be assigned to the corresponding attribute. In our example, the micro transition type between state types *requesting* and *scheduling* 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 turn, in state *scheduling*, the *Secretary* must fill attributes *scheduled_date*, *scheduled_doctor*, and *scheduled_room*. She further must decide when to notify the patient about the scheduled appointment; i.e., the subsequent state *notifying patient* will not be activated before explicitly confirmed by the *Secretary*.

In turn, a user decision is required if a micro step type has more than one outgoing micro transition types. For this purpose, each micro step type may comprise a set of corresponding *value step types*. Each value step type represents a particular constraint to the micro step type; i.e., specific value constraints to the attribute associated to the micro step type. In our scenario, the responsible user must decide which of the subsequent states shall be activated. Figure 4 shows a fragment of the *MRI* micro process type; here, the radiology specialist must decide in case of a patient's anxiety scenario (state type *treating anxious patient*) whether or not to sedate the patient (attribute *sedation*). If the doctor decides not to sedate the patient (value step type *no*), at runtime the next activated micro step will be *reason_no_sedation*; i.e., the doctor must provide a reason for his decision. If he decides that sedation is required (value step type *yes*), micro step types *sedative* and *sedation_time* will be activated at runtime.

User authorization (Figure 3c): To coordinate the processing of an object, user roles have to be assigned to the different states of its 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 allowing for a fine-grained access control policy. The right to write an attribute may 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 associated with the state type. This way, users currently not involved in process execution may access process relevant data if required.

Based on the authorization table of a micro process type, PHILharmonicFlows 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 a particular attribute in a certain state, the respective form field will

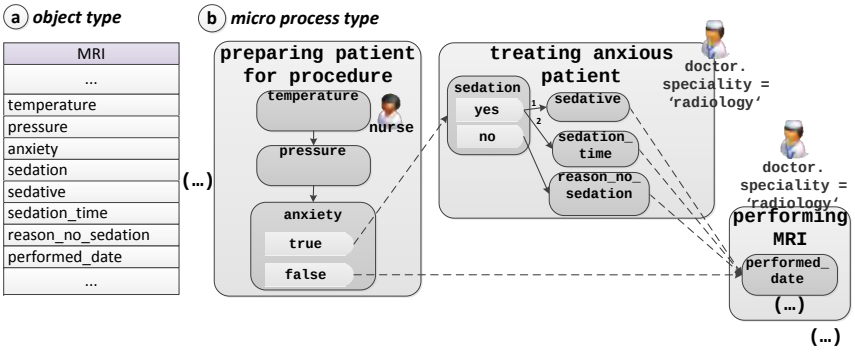

Figure 4. Partial view of the MRI micro process type

not be editable and 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 user form.

The concepts provided by PHILharmonicFlows to enable proper authorization for micro process execution are exemplified in Figure 6. It illustrates the authorization table of micro process type *Mammography*. In this example, state type *requesting* has only one mandatory attribute *request_date* (marked as *MW* in the authorization table). This attribute must be set by the doctor requesting the examination. In addition, attributes *request_desired_date* and *request_observations* are optional (marked as *OW*); i.e., they may be written by the doctor in the respective state. However, this state may be also left as soon as mandatory attribute *request_date* is written. In state *scheduling*, the same doctor 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.

Macro Process (Figure 3d): 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, a corresponding micro process instance exists. As a consequence, a healthcare scenario may comprise dozens or hundreds of micro process instances. Taking their various interdependencies into account, we obtain a complex, dynamically evolving *process structure*. In order to enable a proper interaction between these micro process instances, a *coordination mechanism* is required to realize the interaction points of the micro processes involved. For this purpose, PHILharmonicFlows automatically derives a state-based (i.e., abstracted) 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 (i.e., to particular object types) and consists of both *macro step types* and *macro transitions types* connecting the former. As opposed to traditional process modeling approaches, which define process steps in terms of black-box activities, a macro step type always refers to an object type with a corresponding state type. The macro process type depicted in Figure 8 illustrates this. The process begins with the instantiation of object type *Diagnosis*, which triggers the creation of a corresponding micro process instance. Then, object type *Anamnesis* is instantiated (i.e., the responsible doctor receives a corresponding item in his worklist) and its micro process instance is created. During *Patient Examination*, the *symptoms* may be collected, which are 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 noteworthy that for one primary examination, more than one symptom may be collected.

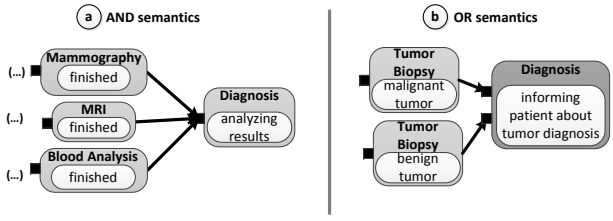

Figure 7. Example of macro input types representing different semantics

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. Such input types can then be associated with several macro transitions. At runtime, a macro step is enabled if at least one of its macro inputs becomes activated. In turn, a macro input is enabled if all incoming macro transitions are triggered. In PHILharmonicFlows, it is possible to differentiate between AND and OR semantics. For representing the semantics of

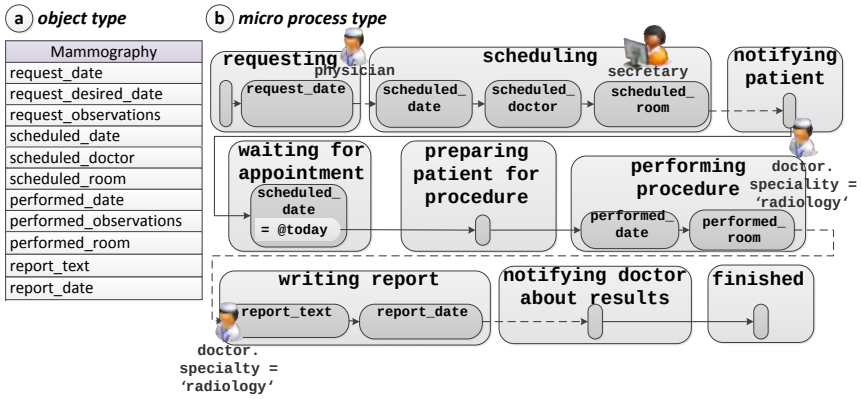

Figure 5. Mammography micro process type

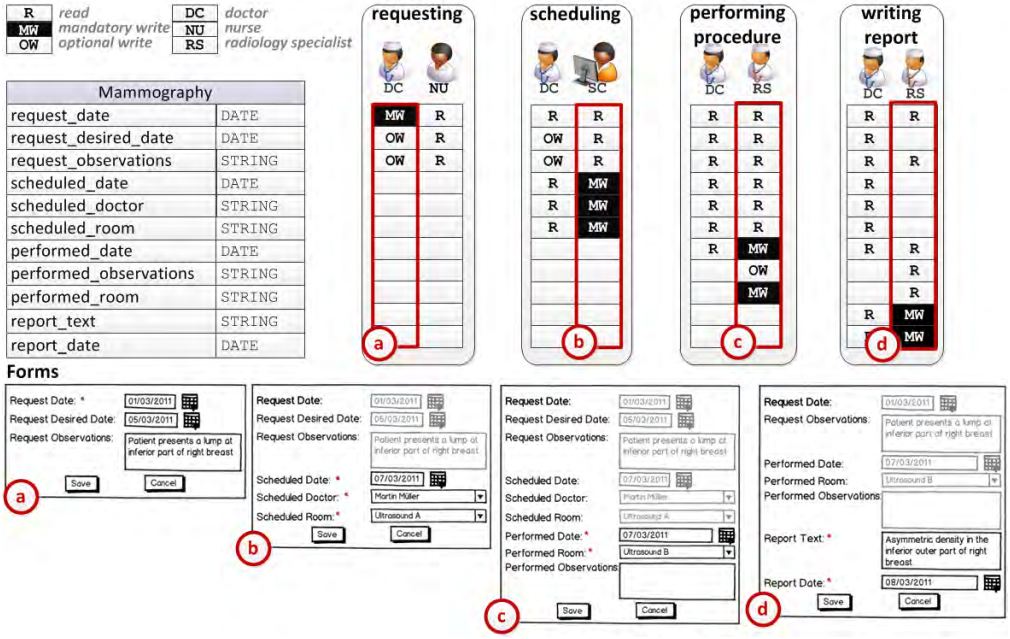

Figure 6. Authorization table and forms of the Mammography micro process type

an AND-join in the macro process, several incoming macro transitions target a single macro input type (see Figure 7a). For representing the OR-join semantics, a macro step type must have more than one macro input type associated (see Figure 7b). In this case, to enable the targeting macro step at least one of its macro inputs must be activated.

Coordination (Figure 3e): 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, making our approach fundamentally different compared to conventional PrMS.

To enable 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 9, in our case study, object type *Diagnosis* is at the top level, while all examinations are placed at a lower level. For example, images refer to respective examinations (i.e., imaging encounters). Hence, 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 [17].

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* (see

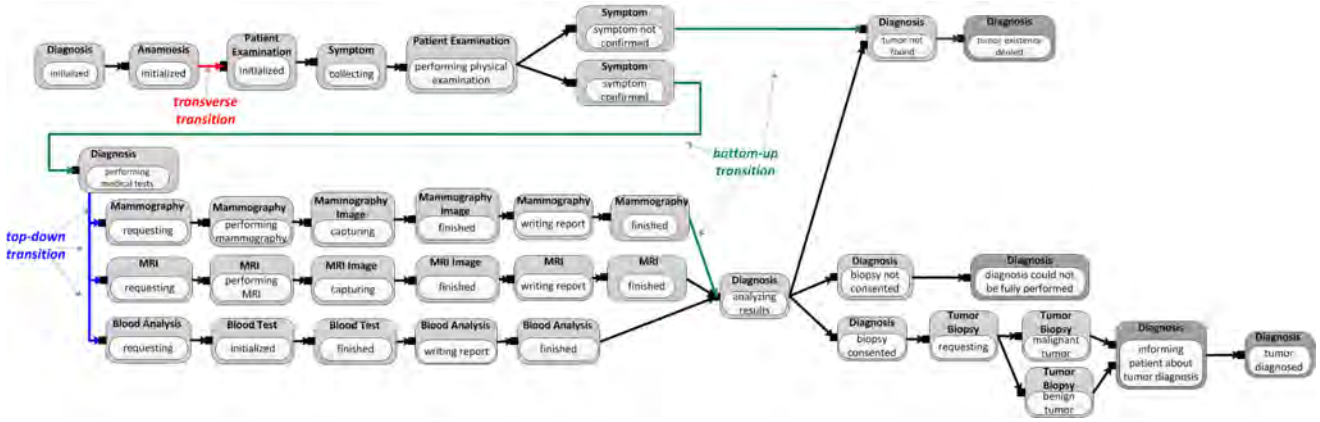

Figure 8. A macro process type coordinating the interactions among the different micro process types

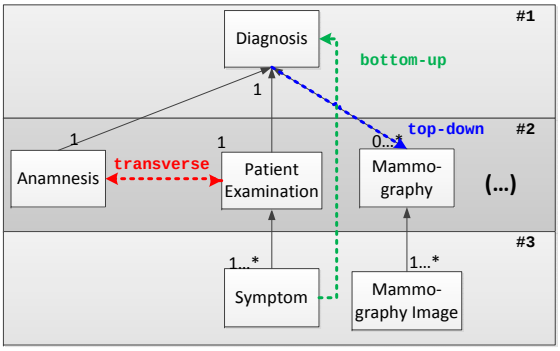

Figure 9. Different kinds of relationships between object types

Figure 9). 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, which refer to the same object type, no coordination component is needed. These transitions are denoted as *self-transitions*. For all other ones, the required coordination component 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 omit further details. We do also not discuss transverse macro transition types. In turn, a *bottom-up transition* characterizes an interaction from a lower-level object type to an upper-level one. In this case, the execution of a higher-level micro process instance depends on the one of several lower-level micro process instances of 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 micro process 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, which establish constraints regarding the lower-level micro process instances in order to activate a particular state of a higher-level micro process instance. As illustrated in Figure 10, the *Diagnosis* process activates state *tumor not found* if all the micro process instances related to instances of object type *Symptom* reach state *symptom not confirmed*. The aggregation condition for this case (`#IN=#ALL`) indicates this constraint. This example illustrates how such counters work. As illustrated in Figure 10, 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 *symptom not confirmed* (`#IN=2`), while one 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 *tumor not found* is activated for the respective *Diagnosis* instance.

As a *proof-of-concept*, we developed a prototype that implements the concepts of the PHILharmonicFlows framework, enabling the modeling and enactment of object-aware processes. Figure 11 shows a screenshot of the data model from our healthcare scenario being modeled in the tool. Figure 12 shows the micro process type regarding object type *Mammography*. In this screenshot, the upper part of the screen presents the object types with their relations and lets the user select for which object type he wants to model the micro process type. The lower part of the screen lets the user model the corresponding micro process type.

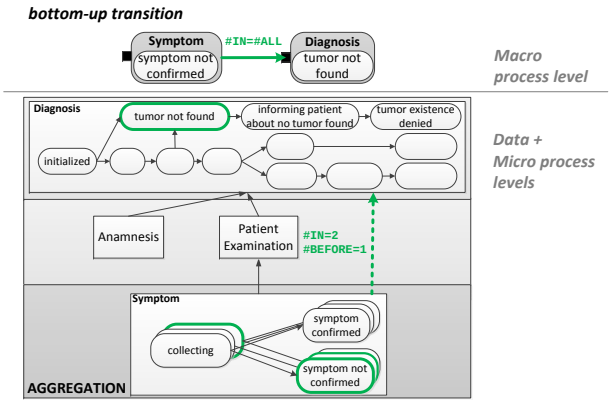

Figure 10. Aggregation example

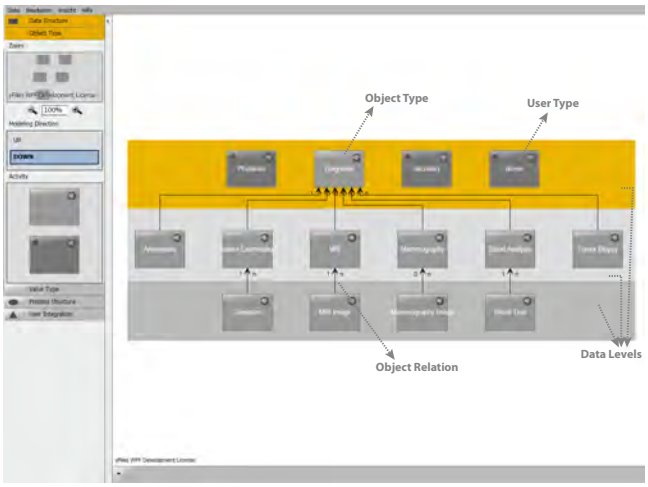

Figure 11. Screenshot of a Data Model

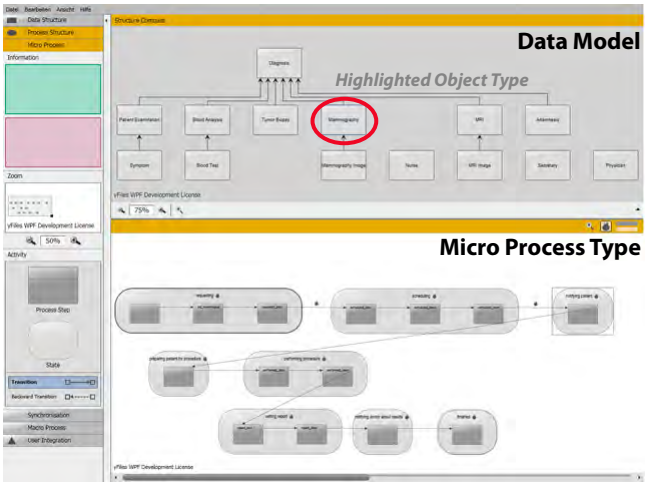

Figure 12. Screenshot of a Micro Process Type

IV. Runtime Environment of PHILharmonicFlows

The *runtime environment* of PHILharmonicFlows 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 operational semantics defined by PHILharmonicFlows enables sound process execution. Additionally, it provides the basis for automatically generating end-user components of the runtime environment (e.g., tables giving an aggregated overview of all processed object instances, user worklists, and form-based activities).

At runtime, the execution of individual micro process instances is based on well-defined *markings* [15]. More precisely, these markings indicate which components of a micro process instance are activated at a certain point of time; i.e., the *processing state* of a micro process instance is defined by the current marking of its states, micro steps, and micro transitions. Based on these markings (see Figure 13), it becomes possible to not only specify which components are activated at a certain point in time, but also the components that may be activated later on and the ones that cannot be activated anymore (since they belong to a *skipped execution path*).

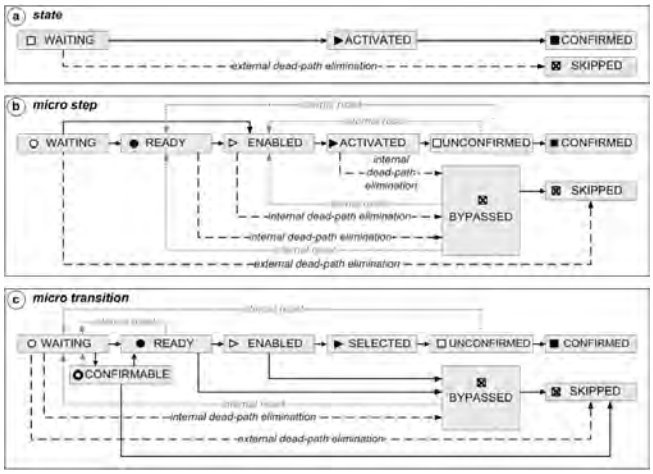

Figure 13. Operational semantics for states, micro steps, and micro transitions

To illustrate how a micro process instance is created, executed, and terminated, we refer to our example. In particular, to illustrate how the operational semantics of PHILharmonicFlows looks like, we refer to the micro process type related to object type *MRI* (see Figure 4).

Creation of a Micro Process Instance: When creating an instance of object type *MRI*, a corresponding micro process instance is automatically generated and initialized as well. According to Figure 14, the start micro step is then marked as CONFIRMED and the state to which it belongs (i.e., state

requesting) is marked as ACTIVATED. In turn, all other states are initially set to WAITING. Further, the outgoing micro transition of the start step is marked as READY, while all other micro transitions are initially marked as WAITING. In our example, this micro transition corresponds to the incoming micro transition of micro step *request_date*, which is marked as ENABLED. All other micro steps not belonging to the state *requesting* are marked as WAITING.

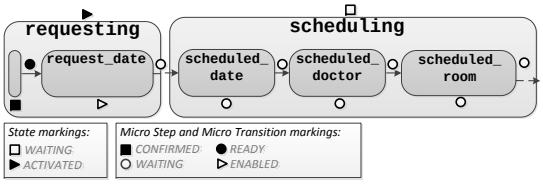

Figure 14. Initiating an instance of a micro process

Execution of a Micro Process Instance: Consider Figure 15a. When state *treating anxious patient* is reached, it is marked as ACTIVATED. The first micro step (i.e., *sedation*) is then marked as ENABLED. This means that a value must be provided for the corresponding attribute. In particular, this micro step refers to a decision to be made by the medical doctor on whether or not sedate the patient. Note that the execution path of the micro process depends on this decision; i.e., if the doctor chooses to sedate the patient, he must fill the values on the form about the *sedative* used and the sedation time chosen. In the micro process, this decision point is represented as a value-specific micro step. Thereby, not only the micro step, but its value steps *yes* and *no* are marked as ENABLED. If the user sets a value corresponding to one of these value steps (see Figure 15b), the selected value step is set to ACTIVATED as well as the corresponding micro step. However, if the user sets a value that does not correspond to any value step type (see Figure 15c), the micro step is marked as BLOCKED. In turn, this blocks the execution of the micro process instance as whole. The latter is indicated to the user by highlighting the input field in the form with an exclamation point. To unblock a blocked micro process execution, the user must set a valid value for the attribute referred by the BLOCKED micro step.

When a valid value is set for the attribute referred by micro step *sedation* (i.e., this micro step is marked as ACTIVATED), the incoming micro transition of this micro step changes its marking to ACTIVATED. In turn, this enables micro step *sedation* to change its marking from ACTIVATED to UNCONFIRMED. However, the corresponding value steps of this micro step must be handled as well. In our example (see Figure 16), the value step marked as ACTIVATED (i.e., *yes*) changes its marking to UNCONFIRMED, while the one still marked as ENABLED is now marked as BYPASSED.

After setting a value for the attribute corresponding to micro step *sedation*, micro steps *sedative* and *sedation_time* become marked as ENABLED (see Figure16a). In the user form, this is visualized by highlighting both input fields, which means that the user must provide a value for at least one of the two attributes; i.e., if for one of these attributes (e.g., *sedation_time*) a value is set, the corresponding micro step will be marked as ACTIVATED (see Figure 16b). Additionally, the incoming micro transition is marked as ENABLED. Since no value for attribute *sedative* is provided, the priorities of the micro transitions (i.e., signalized on the respective micro transitions outgoing from micro step *sedation*) are not relevant for this case. Thereby, the incoming micro transition may be marked as ACTIVATED as well (see Figure 16c). In order to omit the alternative execution path, in this case an *internal dead path elimination* is performed (see Figure 16d). Based on it, all micro transitions and steps belonging to the non-selected path are marked as BYPASSED; i.e., a micro step is marked as BYPASSED if all incoming micro transitions are marked as BYPASSED.

As long as the change of state *treating anxious patient* has not been confirmed (i.e., the transition to the next state is not confirmed by the user), the doctor still may set a value for attribute *sedation*. To accomplish this, an *internal reset* of the currently activated state is performed (see Figure 16e). Normally, the micro steps and transitions will be reset if an attribute value corresponding to a micro step marked as UNCONFIRMED or BYPASSED is changed. However, if values for both attributes *sedation* and *sedation_time* are assigned (see Figure 16f), more than one micro transition becomes ENABLED. Since only one micro step (and one micro state) can be reached, it must be ensured that only one of the execution paths is in fact executed (i.e., only one of the micro transitions is fired). For this purpose, only the micro transition with the highest priority is marked as ACTIVATED (see Figure 16g); i.e., only the one that reaches micro step *sedation* is ACTIVATED. The other micro transition is marked as BYPASSED using an internal dead path elimination. If a state is marked as CONFIRMED afterwards, micro steps and transitions marked as BYPASSED are finally marked as SKIPPED.

When marking a micro step as UNCONFIRMED, outgoing micro transitions are either marked as READY or CONFIRMABLE. More precisely, external micro transitions, for which an explicit user commitment is required, are marked as CONFIRMABLE. Consequently, a mandatory activity enabling this commitment is automatically assigned to the worklist of the responsible user. Regarding our example, after deciding to sedate the patient and filling out in the form which sedative was given to the patient, the outgoing explicit micro transition is marked as CONFIRMABLE. In turn, this requires for the assigned user (e.g., radiologist) to confirm the values of the corresponding attributes. In this case, the explicit micro transition then changes its marking to READY. Opposed to this, implicit micro transitions are immediately marked as READY. If an external

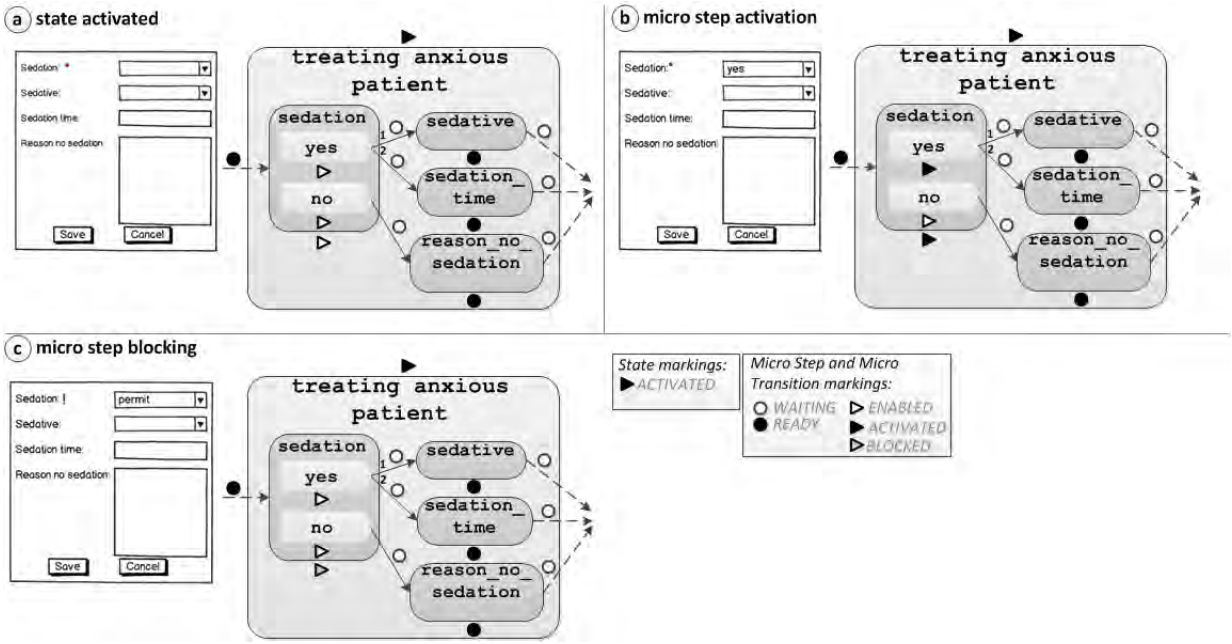

Figure 15. Execution markings for value-specific micro step

micro transition is marked as READY, the currently activated state will be marked as CONFIRMED. Additionally, all corresponding micro steps as well as internal micro transitions (currently marked as UNCONFIRMED) are re-marked as CONFIRMED. Following this, the subsequent state (i.e., state *performing MRI* in our example) is marked as ACTIVATED and its micro steps as READY. The target micro step of the external transition (i.e., in our example *performed_date*) is marked as ENABLED. For this micro step, a value must be set for the corresponding attribute. Moreover, PHILharmonicFlows performs an *external dead-path elimination* in order to mark micro steps, micro transitions, and states, which can no longer be activated, as SKIPPED.

Despite any predefined sequence of micro steps, users may freely choose their preferred execution order; i.e., the order in which attribute values are set within a processed form does not have to coincide to the one of the corresponding micro steps. Particularly, at runtime a micro step may be completed as soon as a value is assigned to the corresponding object attribute.

Termination of a Micro Process Instance: The execution of a micro process instance terminates when a state containing an *end micro step* becomes marked as SELECTED. Using the introduced internal and external dead-path elimination, all other states, micro steps and micro transitions are then either marked as CONFIRMED or SKIPPED.

V. Discussion

In Section II, we have introduced a realistic healthcare scenario, which we modeled in Section III using the PHILharmonicFlows framework. In Section IV, we presented the operational semantics of the execution environment of PHILharmonicFlows framework to indicate how data-driven process execution works in PHILharmonicFlows. In this section, we discuss how the requirements posed by the healthcare scenario are met.

Requirement R1 (Data and process integration): The well-defined modeling methodology provided by PHILharmonicFlows ensures that each procedure (e.g., anamnesis, physical examination, or mammography) 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 respective procedures is stored and managed without need to access external databases during the execution of activities. In particular, this enables users to access and manage process-related data (i.e., object instances) at any point in time (assuming proper authorization) and not only when working on assigned mandatory activities.

Requirement R2 (Intense use of forms): Based on authorization tables, PHILharmonicFlows automatically generates user forms during runtime. 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 user form comprises fields corresponding to read and write permissions for respective object attributes. Moreover, in PHILharmonicFlows, object instances and activities are

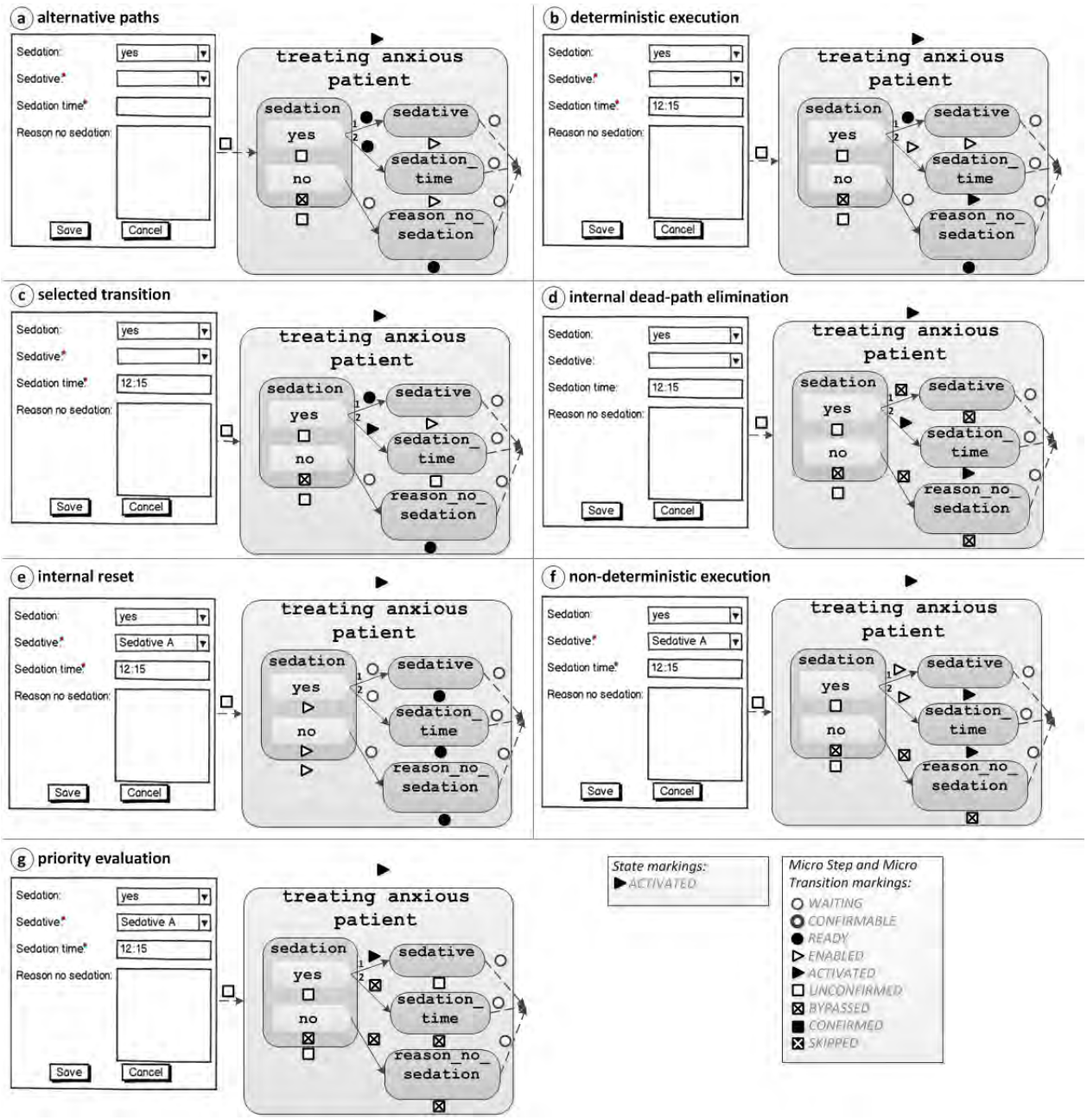

Figure 16. Execution of state *treating anxious patient*

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. In this scenario, 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 as well.

Requirement R3 (Interacting processes): As discussed in Section III, this requirement is met by PHILharmonicFlows by the support of macro processes that coordinate the execution of related micro process instances. 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 instances of other micro processes. 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 purpose, each macro transition type can be specialized using different coordination components. The choice of the latter depends on the relation existing 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.

Requirement R4 (Flexibility regarding process instantiation): Using PHILharmonicFlows it becomes possible to consider a dynamically evolving number of inter-related micro process instances. Taking the defined cardinality constraints into account, users may autonomously 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 asking for 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 during the diagnosis process and to react on unexpected events (e.g., drug prescription in case of intense fever).

Requirement R5 (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 may be written or read by a particular user role in the currently activated micro process state. PHILharmonicFlows ensures that no data is written or read by unauthorized users. Since each state type has an associated user role, the authorization table automatically ensures that this role owns the required data permissions; i.e., the role has mandatory write permission regarding the attributes associated with the micro step types in the state type.

Requirement R6 (Flexible data access): As opposed to traditional PrMS, PHILharmonicFlows presents two different views to the end-users: a process-oriented view (i.e., worklists) and a data-oriented one (i.e., overview tables listing selected 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.

VI. Related Work

Healthcare is a challenging domain for process support, since it comprises structured and unstructured processes. The enactment of such processes requires a high degree of flexibility [19], [20], [34]. In particular, due to their tighter integration of data and processes, data-centric approaches support process flexibility.

One prominent approach is the Case Handling paradigm [4]. It aims at data and process integration by managing the data inside the "case" scope. It also enables form-based activities. If further targets at increasing the degree of flexibility by providing access to information outside the scope of an activity. However, data is provided in terms of atomic elements and may be completely read by any user involved in the case; i.e., no fine-grained access control can be realized. Furthermore, there is no full support regarding interactions among different cases. In [4], the authors mention case studies realized in the healthcare area. However, they focus on administrative processes (e.g., patient registration and administrative processing).

In COREPRO [23], [24], the process structures are generated accordingly to the data structure and the interaction among different process instances is enabled. However, it does not offer the same execution flexibility as PHILharmonicFlows and Case Handling, since the process execution is not directly coupled with the activity of data. The Product-driven Workflow Design approach [39] targets at the precise derivation of a process execution sequence according to the product structure following three design criteria (quality, costs, and time). However, it does not aim at flexible execution of processes driven by data. The Proclets approach [2], [22] enables interactions among process fragments. 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 [27], [28] 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 interested readers to [16].

VII. 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 and we explained how the runtime environment of PHILharmonicFlows works. 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 in which data drives process execution and coordination. In turn, this allows for a higher degree of flexibility enabling data access outside the context of black-box activities.

Like in activity-centered approaches [32], [34], schema evolution is a complex and error-prone task to be accomplished for object-aware processes as well. Therefore, we are working on an extension of the framework to enable controlled schema evolution; i.e., a mechanism to manage and apply changes to object-aware process models as well as their running instances. Since all components of the framework are tightly integrated, the mechanism must take into account that each change (e.g., deleting an object attribute) might affect other components (e.g., a micro step type in a micro process type that must be deleted when the corresponding object attribute is deleted). 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 process correctness. A preliminary work defining the challenges existing in this context is presented in [6].

Concerning healthcare processes, another potential future work is the integration of the PHILharmonicFlows framework with medical information systems. One particular challenge is dealing with complex attribute types (e.g., image data) and making the processes compliant with the DICOM [40] standard.

ACKNOWLEDGMENT

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

REFERENCES

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

[2] W. M. P. van der Aalst, P. Barthelmess, C. Ellis, and J. Wainer, *Workflow Modeling Using Proclets*, Proc. CoopIS'00, LNCS 1901, pp. 198–209, 2000.

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

[4] 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), pp. 129–162, 2005.

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

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

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

[8] P. Dadam, M. Reichert, and K. Klaus, *Clinical Workflows - The Killer Application for Process-oriented Information Systems?*, Proc. BIS'00, pp. 36–59, 2000.

[9] N. Gronau and E. Weber, *Management of Knowledge Intensive Business Processes*, Proc. BPM'04, LNCS 3080, pp. 163–178, 2004.

[10] C. Guenther, M. Reichert, and W. M. P. van der Aalst, *Supporting Flexible Processes with Adaptive Workflow and Case Handling*, Proc. ProGility'08, pp. 229–234, 2008.

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

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

[13] 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.

[14] 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), pp. 19–46, 2011.

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

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

[17] V. Künzle, *Object-Aware Process Management*, PhD Thesis, Ulm University, 2013.

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

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

[20] R. Lenz, S. Miksch, M. Peleg, M. Reichert, D. Riano, and A. ten Teije, *Process Support and Knowledge Representation in Health Care*, LNAI 7738, 2013.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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. ProHealth 2007, LNCS 4928, pp. 335–346, 2007.

[26] B. Mutschler, B. Weber, and M. Reichert, *Workflow Management versus Case Handling: Results from a Controlled Software Experiment*, Proc. SAC'08, pp. 82–89, 2008.

[27] C. P. Neumann and R. Lenz, *alpha-Flow: A Document-based Approach to Inter-Institutional Process Support in Healthcare*, Proc. ProHealth 2009, LNBIP 43, pp. 569–580, 2009.

[28] C. P. Neumann and R. Lenz, *The alpha-Flow Use-Case of Breast Cancer Treatment – Modeling Inter-Institutional Healthcare Workflows by Active Documents*, Proc. WETICE-2010, pp. 17–22, 2010.

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

[30] G. M. Redding, M. Dumas, A. Hofstede, and A. Iordachescu, *A Flexible, Object-centric Approach for Business Process Modeling*, Proc. SOCA, 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, 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, pp. 115–135, 2009.

[33] M. Reichert, *What BPM Technology Can Do for Healthcare Process Support*, Proc. AIME'11, LNAI 6747, 2–13, 2011.

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

[35] S. Rinderle and M. Reichert, *Data-Driven Process Control and Exception Handling in Process Management Systems*, Proc. CAiSE'06, LNCS 4001, pp. 273–287, 2006.

[36] B. Schultheiß, J. Meyer, R. Mangold, T. Zemmler, and M. Reichert, *Designing the Processes for Chemotherapy Treatment in a Women's Hospital* (in German), Technical Report, DBIS-5, DBIS, Ulm University, Germany, 1996.

[37] B. Schultheiß, J. Meyer, R. Mangold, T. Zemmler, and M. Reichert, *Designing the Processes for Ovarian Cancer Surgery* (in German), Technical Report, DBIS-5, DBIS, Ulm University, Germany, 1996.

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

[39] 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.

[40] Digital Imaging and Communication in Medicine, *The DICOM Standard*, http://medical.nema.org/standard.html. Last visited on June 14th, 2013.