Proclets in Healthcare

R.S. Mans[1,2], N.C. Russell[1], W.M.P. van der Aalst[1], A.J. Moleman[2], P.J.M. Bakker[2]

[1] Department of Information Systems, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{r.s.mans,n.c.russell,w.m.p.v.d.aalst}@tue.nl

[2] Academic Medical Center, University of Amsterdam, Department of Quality Assurance and Process Innovation, Amsterdam, The Netherlands.
{a.j.moleman,p.j.bakker}@amc.uva.nl

Abstract. Healthcare processes can be characterized as weakly-connected interacting lightweight workflows coping with different levels of granularity. Classical workflow notations are falling short with regard to supporting these kind of processes as they support monolithic processes which describe the life-cycle of individual cases and allow for hierarchical decomposition. The proclets framework is one of the formalisms providing a solution to this problem. Based on a large case study, describing the diagnostic process of the gynecological oncology care process of the Academic Medical Center (AMC), we identify the limitations of "monolithic workflows". Moreover, by using the same case study, we investigate whether healthcare processes can be effectively described using proclets. In this way, we provide a comparison between the proclet framework and existing workflow languages and identify research challenges.

1 Introduction

In healthcare organizations, such as hospitals, many complex, non-trivial processes are performed which are lengthy in duration. These processes are *diverse*, *flexible* and often involves *several medical disciplines* in diagnosis and treatment. For a group of patients with the same condition, a number of different examinations and treatments may be required and the order in which they are conducted can vary greatly.

In order to guarantee the correct and efficient execution of healthcare processes, there is a need for technological support in controlling and monitoring their delivery to patients [24]. Workflow Management Systems (WfMSs) are an interesting means of achieving this goal. Based on a corresponding process definition, which specifies which tasks need to be executed and in which order, i.e. the control-flow, they support processes by managing the flow of work such that individual work-items are done at the right time by the proper person [6].

Contemporary WfMSs have difficulties dealing with the dynamic nature of processes [4]. One of the main problems is that they require that the complete workflow is described as *one* monolithic overarching workflow. This assumes that

a workflow process can be modeled by specifying the *life-cycle of a single case in isolation*. For real-life processes this assumption can not be made. As a result, the control-flow of several cases need to be artificially squeezed into a single model. Obviously, if a complex healthcare process is described in this way, this results in an unreadable process definition where essential parts of the control-flow are ultimately hidden inside custom-made application software.

This can be illustrated when considering a typical healthcare process for the diagnosis of patients. In general for a patient this consists of multiple visits to a hospital in order to meet with doctors and undergo diagnostic tests (e.g. a lab test). However, there also steps in which several medical specialists meet in order to discuss the status of patients. Clearly, some tasks may operate at the level of a single patient, whereas other tasks operate at the level of a group of patients. So, processes may rely on information that is at different levels of *aggregation*.

The process of diagnosing a patient typically consists of the execution of a number of smaller processes that run in conjunction to each other. Flexibility in healthcare processes originates from the fact that these small processes can be instantiated and synchronized at any point in time. For example, at any point in the process of diagnosing a patient, a doctor may order a lab test. However, although these process fragments execute independently from each other, a certain "magnetism" exists between them. Such process fragments can best be characterized as *weakly-connected interacting lightweight workflows*.

To date, contemporary WfMSs do not offer support for weakly-connected interacting lightweight workflows which can deal with information that is at varying levels of aggregation. An interesting means of solving this issue is provided by *proclets* [4, 5]. Proclets are a framework for lightweight workflow processes. Together with *performatives* and *channels* it is possible to describe how these proclets *interact* with each other. Moreover, the interaction between these proclets is modeled explicitly using structured messages, called performatives, which are exchanged via channels.

As proclets provide an interesting means of modeling and executing a healthcare process in WfMSs, in this paper *we investigate whether healthcare processes can indeed be modeled using this technique and how this compares to existing workflow approaches*. Therefore, we take the following approach. We focus on the gynecological oncology workflow as it is performed at the Academic Medical Center (AMC) in Amsterdam, a large academic hospital in the Netherlands, which is considered to be *representative* of other healthcare processes. The selected healthcare process describes the diagnostic process for patients visiting the gynecological oncology outpatient clinic and is a large process consisting of around 325 activities. As an earlier effort, this healthcare process has already been modeled in full using the workflow languages, YAWL and FLOWer, and has been partially modeled using the Declare and ADEPT1 workflow languages [25]. Additionally, *this allows us to investigate the problems existing workflow approaches are facing*. We discuss in detail how the healthcare process has been modeled using the YAWL workflow language. For FLOWer, ADEPT1, and Declare, we summarize the issues encountered. This leads on to a discussion of how

the same healthcare process is modeled using proclets and how the identified issues can be addressed. It is worth noting that the reason for implementing a hospital process in the four workflow systems mentioned above was to identify the requirements that need to be fulfilled by workflow systems, in order to be successfully applied in a hospital environment. These requirements have been discussed in [27].

This paper is structured as follows: Section 2 introduces the proclets approach. In Section 3, we introduce the gynecological oncology healthcare process and discuss how it is modeled using YAWL, FLOWer, Declare, and ADEPT1. In Section 4, we discuss the modeling of the healthcare process using proclets and elaborate on how the limitations, mentioned in Section 3, are addressed. Related work is outlined in Section 5. Section 6 discusses the experiences associated with modeling the healthcare processes using proclets and concludes the paper.

2 Introduction to Proclets

In this section, we will discuss proclets which form a framework for modeling workflows. The concepts of the framework have already been introduced in [4, 5]. In this section, we give an introduction to this framework in order to assist the reader in better understanding the proclet models that will be shown in the remainder of this document. For complete details we refer the reader to [4, 5]. At the end of this section, the use and operation of proclets will be illustrated by a small healthcare example.

In Figure 1, a graphical representation of the concepts, which underpin the framework, is shown. As can be seen, there are five main concepts of which each will be discussed below.

The framework is centered around a ***proclet***. There is a distinction between a proclet class and a proclet instance. A *proclet class* can best be seen as a process definition which describes which tasks need to be executed and in which order. For a proclet class, instances can be created and destroyed. One instance is called a *proclet instance*. For the definition of a proclet class, a selection can be made between multiple graphical languages. In this paper, we use a graphical language based on the YAWL language [7]. However, other languages, like Petri Nets [1] or EPCs [2], can also be used. With regard to the selection of a graphical language, some limitations apply. First of all, proclet instances need to have a state and they need to support the notion of a task. Second, a proclet class needs to be sound [3].

With regard to the communication and collaboration among proclets, so called *channels*, *ports*, and *performatives* are important. First of all, proclets interact with each other via ***channels***. A channel can be used to send a ***performative*** to an individual proclet or to a group of proclets. A performative is a specific kind of message with several attributes which is exchanged between one or more proclets. A performative has the following attributes:

- *Time*: the moment the performative was created/received.

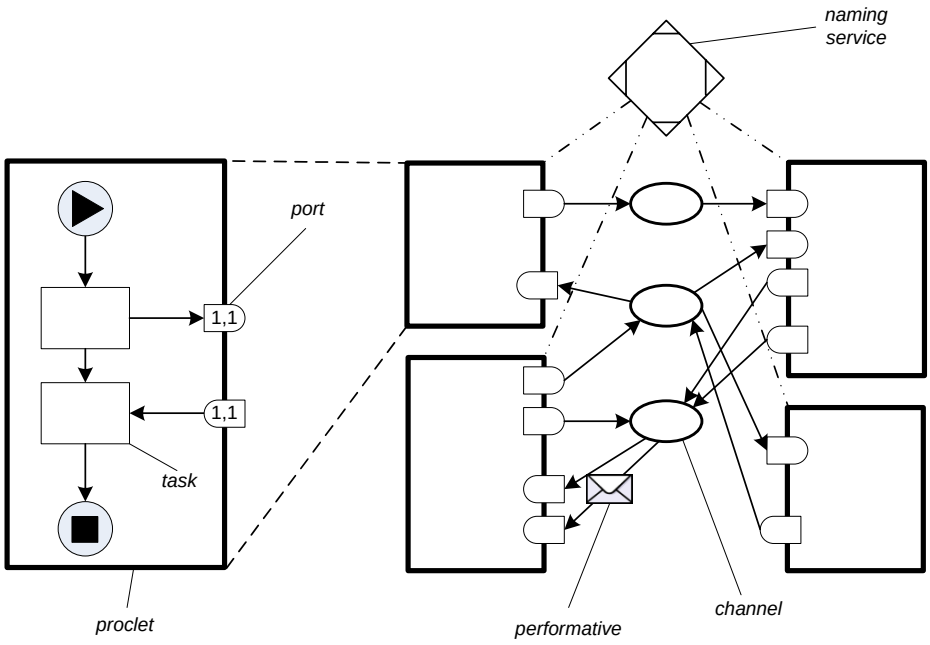


Fig. 1. Graphical representation of the framework.

- *Channel*: the medium used to exchange the performative.
- *Sender*: the identifier of the proclet creating the performative.
- *Set of receivers:* the identifiers of the proclets receiving the performative, i.e. a list of recipients.
- *Action*: the type of the performative. This attribute can be used to specify the illocutionary point of the performative. Examples are request, offer, acknowledge, promise, decline, counter-offer, or commit-to-commit.
- *Content*: the actual information that is being exchanged.

Of course, it is possible to add more attributes to a performative. Note that a channel may have different properties which affect the sending and receiving of performatives, e.g. push/pull or synchronous/asynchronous. In order for proclets to be able to find each other there is a ***naming service*** which keeps track of existing proclets. A proclet class and instances of it are defined in the following way:

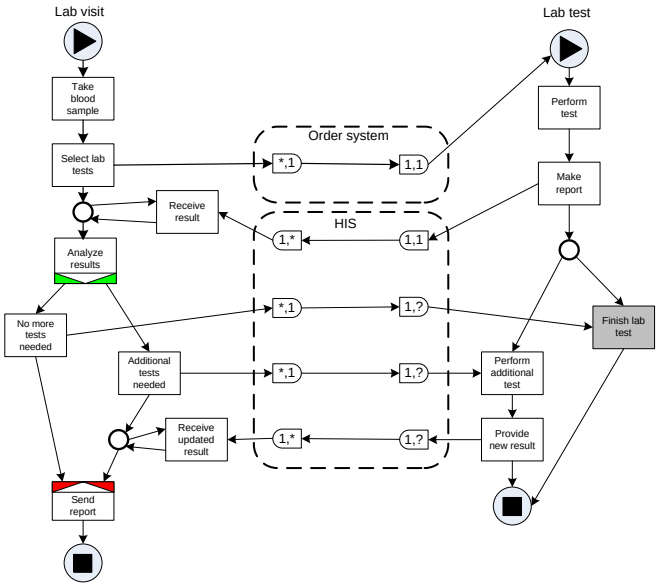
- a proclet class has a **unique name**. In the same way, an instance of a proclet class has an unique identifier.
- a proclet class has ***ports***. Performatives are sent and received via these ports in order for a proclet to be able to interact with other proclets. Every port, either incoming or outgoing, is connected to one task. Moreover, a port has two attributes.

First, the *cardinality* specifies the number of recipients of performatives exchanged via the port. An $*$ denotes an arbitrary number of recipients, + at least one recipient, 1 precisely one recipient, and ? denotes no or just one recipient. Note that by definition an input port has cardinality 1.
Second, the *multiplicity* specifies the number of performatives exchanged via the port during the lifetime of an instance of the class. In a similar fashion to the cardinality, an $*$ denotes that an arbitrary number of performatives are exchanged, + at least one, 1 precisely one, and ? denotes that either one or no performatives are exchanged. Note that by definition an input port has a multiplicity of 1 or ?.

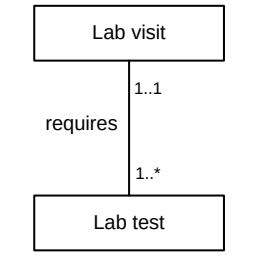
- a proclet instance has its own **knowledge base** for storing performatives that are received and sent. Parts of the knowledge base can be public or private. The public part is identical for all instances of the class, i.e. this part resides at the class level even though it holds information about instances. The private part resides exclusively at the instance level.
- The knowledge base can be queried by tasks. A task may have a *precondition* based on the information that can be found in the knowledge base. A task can only fire if (1) the task in the net itself is enabled, (2) each input port contains a performative, and (3) the precondition evaluates to true. Note that for the YAWL language, as can be seen in Figure 2, multiple ports can be connected to an input condition. In this case, an instance is created on the receival of each performative.
- A task connected to an output port may have a *postcondition*. The postcondition specifies for the output ports, the number of performatives generated and the content. The postcondition may also depend upon information that can be found in the knowledge base.

In order to illustrate the framework, we use the small healthcare-related example shown in Figure 2(a). The example deals with the process of taking blood from a patient so that several lab tests can be performed in order to make a diagnosis. Therefore, there are two proclet classes. The proclet class "lab visit" is instantiated for every patient who visits the lab for whom a blood sample is taken. Proclet class "lab test" is instantiated for every lab test that needs to be performed on the blood sample. Hence, there is an one-to-many relationship between "lab visit" and "lab test" as shown by the relationship *requires* in the class diagram in Figure 2(b).

When a patient visits the lab, a blood sample is taken ("Take blood sample" task) after which the doctor decides which lab tests need to be performed for the sample ("Select lab tests" task). As a consequence, a trigger for each required lab test is initiated, so that for every lab test a single instance of proclet class "Lab test" is created. Consequently, the cardinality of the outgoing port of the "Select lab tests" is $*$. Moreover, the multiplicity is 1 which means that during the lifetime of an instance of the class "Lab visit" exactly one performative is send via this port. The creation performative is send via the lab order system, which explains why the name of the channel is "Order system". The input port connected to the input port of the "Lab test" proclet class has cardinality 1



(a) Two proclet classes connected through two channels



(b) Class diagram containing the two proclet classes

Time	Channel	Sender	Receivers	Action	Content	Scope	Direction
11:00	Order system	Lab visit - John	Lab test – HGB John	Create	Can you perform a HGB test for John?	Private	OUT

(c) Example of a performative

Fig. 2. Example of two proclet classes.

and multiplicity 1 as an instance can only be created once. Figure 2(b) shows an example of a performative that is sent by a "lab visit" proclet to a "Lab test" proclet. From the figure, we can see that at 11 'o clock a performative is sent by the "Lab visit" proclet for patient John in order to create an instance of the "Lab test" proclet called "Lab test - HGB John". More specifically, an instance of a "Lab test" proclet class is created so that a HGB blood test can be performed. The performative is stored in the private knowledge base of the "Lab visit" proclet.

After an instance of the "Lab test" proclet class is created, a test is performed on the blood sample ("Perform test" task) which is followed by the creation of a report which contains the result of the test ("Make report"). This has as consequence that a performative is sent to the instance of the initiating proclet class "Lab visit". Note that each instance of "Lab test" sends performatives via the hospital information system (HIS). The results of the individual lab tests are received by the "Receive result" task. Note that the input port of task "Receive result" has cardinality 1 and multiplicity $*$, indicating that multiple test results may be received. Each performative received is stored in a knowledge base. The "lab visit" proclet continuously inspects this knowledge base and may

decide to start analyzing the results to see if more tests are needed ("analyze results" task). If no more tests are needed, the "No more tests needed" task is performed after which all instances of the "Lab test" proclet class are destroyed. In the situation where the doctor is not confident, a performative is sent via the "Additional tests needed" task to the "Perform additional test" task of all "Lab test" proclet instances to indicate that additional work needs to be done. Note that the cardinality of the output port of both the "Additional tests needed" and "No more tests needed" tasks is $*$, i.e., in one step all the "lab test" proclets are informed about whether additional tests are needed or not. Moreover, the ports connected to the "Perform additional test" task and "Finished lab test" task both have cardinality 1 (i.e. one recipient) and multiplicity ? (one performative is sent via one of the two ports). Furthermore, the "Finish lab test" task is grey-colored as no human input is required when performing the task. After performing the additional test, the "Update report" task is performed which sends the updated report to the "Lab visit" proclet instance where they are collected via the "Receive updated result" task. Finally, the patient is informed via the "Send report" task after which the "Lab visit" proclet instance is destroyed.

3 Limitations of Monolithic Workflows

In this section, we identify the problems existing monolithic workflow approaches are facing dealing with the dynamic nature of processes. We take the following approach. First, we examine the gynecological oncology workflow as it is performed at the Academic Medical Center (AMC) in Amsterdam which is considered to be representative for other healthcare processes. As an earlier effort this process has been modeled in full using two workflow languages, YAWL and FLOWer [25], and has been partially modeled using the Declare and ADEPT1 workflow languages [25]. We discuss the selected healthcare process in detail by elaborating on how the healthcare process has been modeled using the YAWL workflow language and identify issues that arose when doing so. For FLOWer, ADEPT1, and Declare, we also discuss issues that arose when implementing the process although we do not elaborate on specific implementation details. In doing so, we exemplify the problems existing workflow approaches are facing. Subsequently, in Section 4, we discuss how the same healthcare process is modeled using proclets and how the issues identified are addressed using our approach.

The gynecological oncology workflow is a large process, consisting of over 230 activities, and is performed at the gynecological oncology outpatient department at the AMC hospital. The AMC is the most prominent medical research center in the Netherlands and one of the largest hospitals in the country. The healthcare process deals with the diagnosis of patients suffering from cancer once they are referred to the AMC hospital for treatment. The care process can be considered to be non-trivial and illustrative for other healthcare processes, both at the AMC and in other hospitals.

The healthcare process under consideration consists of two distinct parts. The first one is depicted in Figure 3 and shows the top page of the YAWL model.

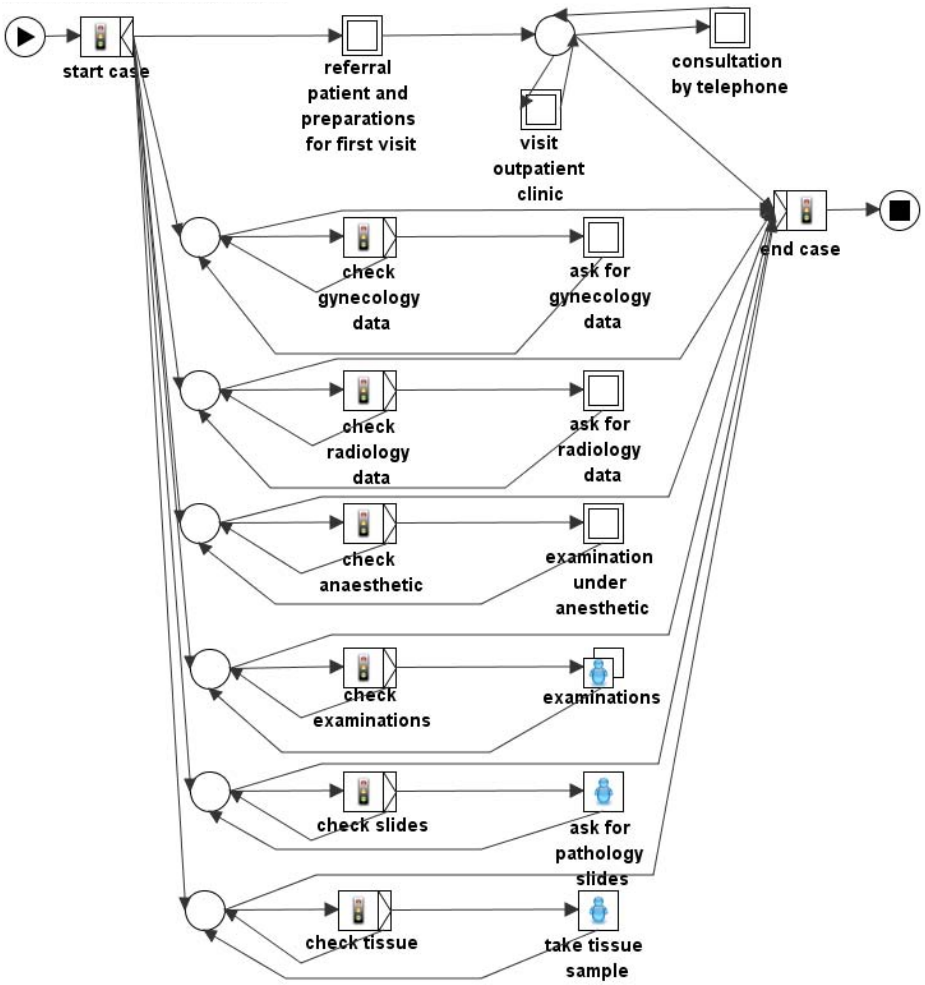


Fig. 3. General overview of the gynecological oncology healthcare process.

The process describes all of the steps that may be taken with a patient up to the point where they are diagnosed. The process starts with the "referral patient and preparations for first visit" composite activity. This subprocess deals with the steps that need to be taken for the first visit of the patient to the outpatient clinic. The next step in the process is the "visit outpatient clinic" composite activity where the patient visits the outpatient clinic for a consultation with a doctor. Such a consultation can also be done by telephone ("consultation by telephone" composite activity). During a visit or consultation, the patient discusses their medical status with the doctor and it is decided whether any further steps need to be taken, e.g., diagnostic tests.

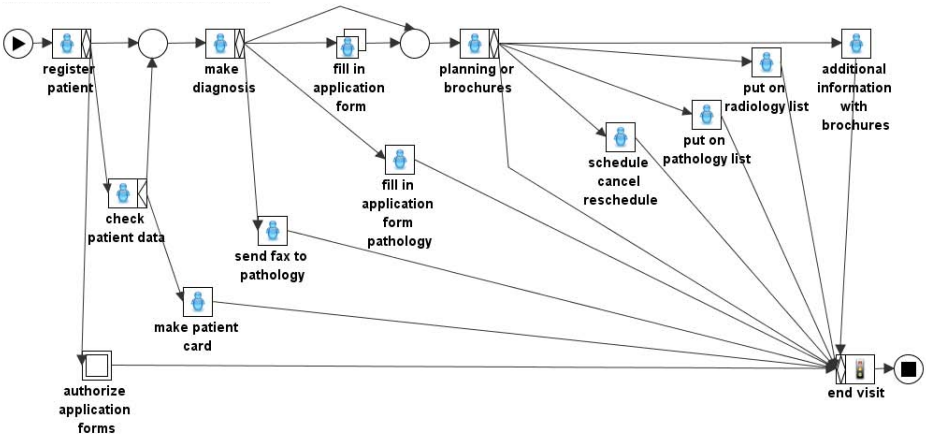


Fig. 4. Visit of the patient to the outpatient clinic.

The execution of the tests that may be needed are modeled by the "examinations" multiple instance task which allows for the concurrent instantiation of a number of different tests for a patient. However, for each patient there are also other steps that may be taken. These are modeled by the "ask for gynecology data", "ask for radiology data", and "examination under anesthetic" composite tasks and the "ask for pathology slides" and "take tissue sample" tasks. For example, the "ask for pathology slides" and "take tissue sample" tasks model the situation where a pathology examination is required after which the referring hospital is requested to send their pathology slides to the AMC or tissue sample is taken at the AMC.

Looking at the overall process we see that while the patient is visiting the outpatient clinic (shown in the top part of Figure 3) it is possible for a series of subprocesses to run concurrently (as shown in the lower part of the figure). As the execution of these subprocesses can be complex and time consuming, there is no guarantee that all of them will be finished before the start of the next patient consultation, e.g. the result of a certain test might be delayed. Consequently, these subprocesses should be seen as separate intertwined life-cycles running at different speeds rather than as one workflow covering different but related cases. However, if we want to denote that there is in fact a connection between these related cases, we need to model them in one monolithic workflow. For the FLOWer, Declare, and ADEPT1 workflow languages, these observations also apply. Therefore, we can conclude that for existing workflow approaches *cases need to be straightjacketed into a monolithic workflow despite the fact that it is more natural to view processes as intertwined loosely-coupled object life-cycles.*

In Figure 4, the subprocess underlying the "Visit outpatient clinic" composite task is shown which describes the visit of a patient to the outpatient clinic. During such a consultation, the medical status of the patient is discussed and a decision is made about the next steps to be taken ("Make diagnosis" task). At

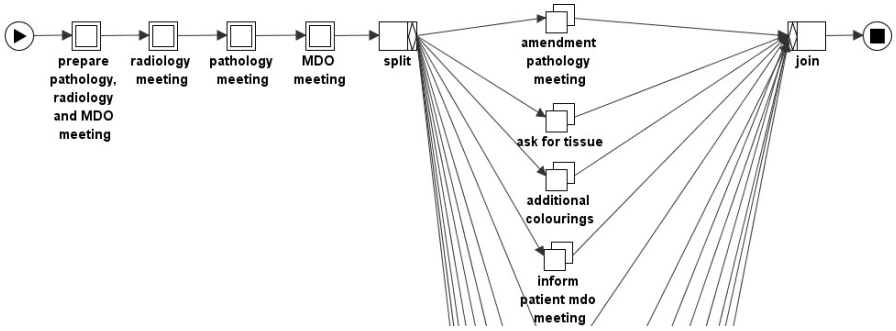


Fig. 5. Meetings which are held on Monday afternoon to discuss the medical status of patients.

different stages during the process, several administrative tasks, such as handing out brochures (task "Additional information with brochures"), and producing a patient card (task "Make patient card") may be necessary. As a result of the execution of the "Make diagnosis" task, subsequent steps in the process need to be triggered, such as further diagnostic tests or a pathology examination. However, these next steps are depicted on the top page of the YAWL model (see Figure 3). As a consequence, they can only be enabled when the process modeled in Figure 4 is already finished. It would be more natural if these kind of processes were instantiated at the moment that it is known that they need to be created, i.e. immediately after execution of the "Make diagnosis" task. In general, for each of the subprocesses modeled in Figure 3, no *direct interaction* can take place during their execution. This is due to the fact that in YAWL there is no way of modeling interactions between (sub)processes. The same observation holds for FLOWer, ADEPT1, and Declare as well. Consequently, facilitating interactions between (sub)processes is far from trivial. Where these need to be supported, they are typically hidden in application logic or in custom built applications.

Note that business process notations exist which support interactions between processes. For example, the Business Process Modeling Notation (BPMN) allows the flow of messages between two entities to be shown via the message flow construct [38]. In general, we can conclude that *as most workflow languages do not provide support for interaction between (sub)processes, it is difficult to model interactions between processes.*

In Figure 5, the second part of the gynecological oncology healthcare process is shown. This involves meetings between gynecological oncology doctors and other medical specialists. First, the participants from the different medical disciplines prepare themselves for these meetings ("prepare radiology, pathology, and MDO meeting" composite task). During the radiology meeting (composite task "radiology meeting"), the doctors from gynecological oncology discuss with a radiologist the results of the radiology tests that have been performed for various patients during last week. The same holds for the "pathology meet-

ing" composite task for the pathology examinations that have been performed during the last week. Finally, during the MDO meeting ("MDO meeting") the medical status of patients is discussed and a decision is made about their final diagnosis before the treatment phase is started. Finally, as a result of these meetings, several subsequent steps may need to be initiated for individual patients. These steps are modeled at the right-hand side of Figure 5. For example, for some patients, existing tissue may need to be re-examined whereas for others, the referring hospital may need to be asked to send their pathology material to the AMC for investigation ("ask for tissue" task).

However, most importantly, compared to the two models discussed earlier, we are dealing with a *group of patients* instead of a single patient. Obviously, compared to the two previous models, we are dealing with a different level of *aggregation*. Due to this difference, the workflows executed for a single patient, shown in Figure 3, and the workflow executed for a group of patients, shown in Figure 5, are modeled separately. Consequently, the two models are completely *disconnected* whereas in reality (examinations for) patients need to be registered for these meetings, which can be initiated from different places in the process described in Figure 3. For example, a patient can be registered during the initial phases of the process and also during a visit to the outpatient clinic. Should these workflows, operating at different levels of aggregation, need to be described in a single model, a decision needs to be made about what is to be considered the case - a service executed for a single patient, or a group of patients. Choosing either these as the unit of modeling causes problems.

For the modeling of the healthcare process using the FLOWer, ADEPT1, and Declare workflow languages, the same problem applies. We are not aware of any workflow language which is able to deal with different levels of granularity. Consequently, *models often need to be artificially flattened as they are unable to account for the mix of different granularities that co-exist.*

Furthermore, the fact that multiple patients can be registered for the afore mentioned meetings (even from different points in the process) indicates that *one-to-many* relationships may exist between entities in a workflow. For example, during a visit to the outpatient clinic, a patient can be registered for discussion during an MDO meeting. This means that a one-to-many relationship exists between the entity "MDO meeting" and the "visit outpatient clinic" entity. However, as models are unable to account for different granularities that co-exist in a workflow this also means that it is impossible to capture one-to-many and many-to-many relationships that may exist between entities in a workflow. Although, *it is impossible to capture the fact that one-to-many and many-to-many relationships exist between entities in a workflow, such relationships are common as can be seen in any data/object model.*

We have discussed problems that we are faced with when modeling the gynecological oncology healthcare process using the YAWL, FLOWer, ADEPT1, and Declare workflow languages. In summary, we may conclude that existing workflow approaches currently exhibit the following problems:

– **Issue 1:** Models need to be *artificially flattened* and are unable to account for the mix of granularities that co-exist in real-life processes.
– **Issue 2:** Cases need to be *straightjacketed into a monolithic workflow* even though it is more natural to see processes as intertwined loosely-coupled object life-cycles.
– **Issue 3:** It is impossible to capture the fact that *one-to-many* and *many-to-many* relationships exist between entities in a workflow, yet such relationships are common as can be seen in any data/object model.
– **Issue 4:** It is difficult to model interactions between processes, i.e., *interaction is not a first-class citizen* in most process notations.

4 Realization of the Gynecological Oncology Workflow using Proclets

In this section, we elaborate on how the gynecological oncology healthcare process is modeled using proclets. First, in Section 4.1, we discuss which entities can be identified in the workflow and how they relate to each other. In Section 4.2, a selection of proclet classes will be discussed, illustrating how the entire healthcare process is modeled using proclets. However, most importantly, it will be explained how the issues identified in Section 3 are addressed using proclets.

4.1 Overview

The class diagram in Figure 6 gives an overview of the entities that exist within the healthcare process and the relationships between them. The dark-grey colored classes correspond to concrete proclet classes. The inheritance relations show which proclet classes have common features, i.e., the light-grey and white colored classes can be seen as abstract classes used to group and structure proclets. The associations show the relationships that exist between proclet classes together with their multiplicity.

Starting with the white colored classes, we see that four main entities exist within the healthcare process. These are:

– **Visit:** A patient can visit a hospital multiple times to see a doctor. This can either be at the outpatient clinic where the doctor examines the patient ("Visit outpatient clinic" class), or an examination under anesthetic ("Examination under anesthetic" class). Moreover, also related to a visit are the initial stages of the process ("Initial phase") in order to prepare for the first visit of the patient.
– **Test:** A doctor can select multiple diagnostic tests that need to be conducted for a patient. The tests that can be chosen range from medical imaging ("MRI", "CT", and "X-ray" classes) to a lab test ("Lab" class), an ECG ("ECG" class), and a pre-assessment ("Pre-assessment" class). For all of these, the presence of the patient is required.

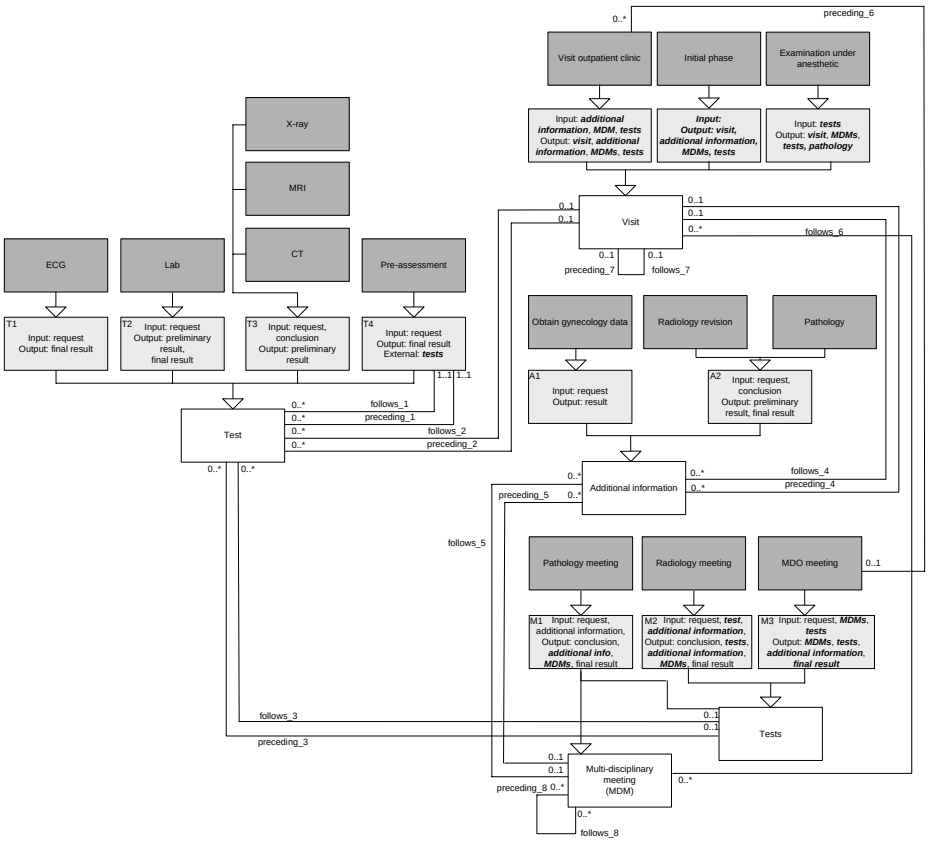


Fig. 6. Class diagram outlining the concepts that exist within the healthcare process and their relationships.

– **Additional information:** A doctor might require additional information in order to reach a diagnosis. This may involve requesting the referring hospital to send patient related data (class "Obtain gynecology data"), and also to send pathology slices ("Pathology" class) and radiology data ("Radiology revision" class) so that they can be reviewed. However, the "Pathology" proclet class also involves (re)examining patient tissue which has been collected in the AMC.

– **Multi-disciplinary meeting:** Every Monday afternoon multiple meetings are organized for discussing the status of patients and/or the outcome of examinations. These meetings involve the departments of radiology ("Radiology meeting" class), pathology ("Pathology meeting" class) and a multi-disciplinary meeting ("MDO meeting" class) involving the departments of gynecological oncology, radiotherapy, and internal medicine (in order to give chemotherapy).

For these four main types, proclets are instantiated a variable number of times and interact in different ways with each other. For these interactions between proclets, a single proclet might require multiple inputs and outputs from other existing proclets. For example, a lab test can be triggered during a visit to the outpatient clinic and also during the initial phases of the process or during an MDO meeting.

To make these interaction related commonalities explicit, the light-grey colored classes in Figure 6, outline these interaction characteristics in terms of inputs and outputs. The items depicted in bold italic indicate that an interaction is optional whereas an item written in normal text indicates that an interaction is mandatory. In this way, the light-grey colored classes explicitly identify (at a high level) the interface that exists for a specific (group of) proclets. Note that not all proclet classes have the same level of aggregation. The multi-disciplinary meeting related proclet classes all deal with a group of patients whereas the other proclet classes are all related to a single patient.

For each main type of proclet class, we will now elaborate on the four main types of proclet classes that have been identified and examine now their interaction with other proclet classes. First, we focus on the "Test" and "Additional information" entities in isolation. Then, we focus on the "Visit" and the "Multi-disciplinary meeting (MDM)" entities and elaborate on the specified associations. In general, an association with the name "follows" indicates that, seen from the viewpoint of the "Visit" and the "Multi-disciplinary meeting (MDM)" entities, an action is initiated (e.g. a lab test). Similarly, an association with name "preceding" indicates that a specific action serves as input to either the "Visit" or "Multi-disciplinary meeting (MDM)" entity (e.g. the result of a lab test is required for a visit to the outpatient clinic).

First of all, for a *test* for which the patient is required to be present, three different ways can be distinguished in which a test is requested and ultimately the result is communicated. One possibility is that a test is requested and the outcome of the test is immediately reported ("T1"), Another possibility is that a test is requested, a preliminary result is communicated, followed by a final result ("T2") at a later time. The third alternative is that a test is requested and a preliminary result is communicated to either the requester or a nominated group of medical specialists. They in turn decide whether an amendment is needed ("T3"). A somewhat special case is "T4" which is similar to "T1". In addition to "T1", proclet classes of this type may also request diagnostic tests for a patient in order to come to a decision. For example, for a pre-assessment test, the anesthetist might require that a lung function test is completed or a consultation with an internist. The act of requesting additional tests in order to come to a final decision are also modeled by the "follows_1" and "preceding_1" associations. These associations indicate that during a pre-assessment multiple tests can be triggered, i.e. the multiplicity is $0..*$, but also that results of multiple tests may be required as input for an examination, i.e. the multiplicity is $0..*$. Note that the requester only initiates an examination and might not be aware

of the fact that additional tests need to be performed in order to arrive at an outcome.

A doctor might decide that *additional information* is required to reach the final diagnosis for a patient. Two different ways can be distinguished in which a request for additional information can be made and the result is delivered to the requester. These are: (1) additional information is requested and the requested information is immediately communicated ("A1"), (2) additional information is requested and a preliminary result is communicated to either the requester or a group of medical specialists. They in turn advise whether further investigation is required ("A2"). Note that the way in which additional information is requested, and the result communicated, is very similar to the way tests are requested and the result communicated.

During a *visit* to the hospital, the patient is examined either at the outpatient clinic or during a procedure under anesthetic. For a visit of the patient at the outpatient clinic (which can also be a consultation by telephone), several inputs might be required. These can be the results of preceding tests, i.e. the multiplicity attached to the "Test" class of association "previous_2" is 0..∗, or additional information that needs to be available, i.e. the multiplicity attached to the "Visit" class of association "previous_4" is 0..∗. Note that the results of tests and additional information may also be required as input to a multidisciplinary meeting. Therefore, the multiplicity attached to the "Visit" class of associations "previous_2" and "previous_4" is 0..1. Moreover, as the status of a patient might be discussed during the MDO multi-disciplinary meeting ("MDO meeting"), the patient may be informed about the discussion afterwards, i.e. the multiplicity attached to "Visit outpatient clinic" and "MDO meeting" of association "previous_6" is 0..∗ and 0..1, respectively.

During a visit by a patient to the hospital, a doctor might require that a subsequent visit, i.e. the multiplicity of associations "follows_7" is 0..1, or that a patient needs to be registered for one or more multidisciplinary meetings, i.e. the multiplicity attached to the "Multi-disciplinary meeting (MDM)" class of association "follows_6" is 0..∗.

Moreover, a doctor might also request additional information, i.e. the multiplicity attached to the "Additional information" class of association "follows_4" is 0..∗, or that tests are triggered for a patient, i.e. the multiplicity attached to the "Test" class of association "follows_2" is 0..∗. Note that tests and additional information may also be triggered for a patient during a multidisciplinary meeting. Therefore, the multiplicity attached to the "Visit" class for associations "follows_4" and "follows_2" is 0..1.

Finally, there are the *multidisciplinary meetings* to discuss the status of multiple patients, to review the outcome of selected diagnostic tests, and to examine additional information that has been requested. Although for a certain meeting distinct inputs and outputs might exist, several commonalities can be identified. As inputs to a meeting, additional information ("previous_5"), tests ("previous_3"), and the outcome of other multidisciplinary meetings ("previous_8") might be required for *multiple* patients. Furthermore, as outputs, it might be

necessary to request additional information ("follows_5"), order further tests for a patient ("follows_3"), or to initiate a multidisciplinary meeting ("follows_8"). This is done for multiple patients. Note that for the above mentioned associations, the reasoning for the multiplicities is similar to those for the "Visit" class.

4.2 Proclets

As already indicated earlier, the dark-grey colored classes in Figure 6 correspond to concrete proclet classes. In total 15 proclet classes have been identified for the gynecological oncology workflow.

The 15 proclet classes identified are connected to other proclet classes via the port and channel concepts. Figure 7 shows a high-level view of the interconnection structure together with the cardinality and multiplicity of the ports. In total, there are 86 possible interactions between the proclet classes which illustrates the complexity of the process.

By using proclets the relationships between different entities can be described in their own process definition. So, it is more natural *to define processes as intertwined loosely-coupled object life-cycles*. Using existing workflow languages, as can be seen in Section 3, it is necessary to flatten this structure into a monolithic workflow model, which is potentially very difficult or even intractable in practice. Moreover, when looking at the cardinalities of individual ports, it is easy to take *different granularities into account*, whereas for existing workflow approaches these relationships can not be captured. In this way, by using proclets, the second issue mentioned in Section 3 can be solved.

Of the 15 proclet classes, we discuss the "Visit outpatient clinic", "Pathology", and "Pathology meeting" proclet classes in detail. The other proclet classes will be discussed in detail in Appendix A. When discussing the proclets, we will show how the limitations of monolithic workflows can be addressed using proclets. Furthermore, we elaborate on the *interaction* of an individual proclet with other proclets, i.e. the *interface* of a proclet. As can be seen in Figure 7, there can be many interactions between proclets and even multiple interactions between the same proclets. In order to show the kind of interactions between two proclets, the following naming strategy is chosen for a port, consisting of several distinct parts: *sending_proclet.task_name_sending_proclet.*[*name*].*S/R*. "*sending_proclet*" refers to the proclet class which sends the performative, "*task_name_sending_proclet*" refers to the specific task (or composite task) in the proclet class that sends the performative, "*S/R*" indicates whether a performative is sent via the port or is received via the port. "*[name]*" refers to a specific (optional) identifier that is added when the naming chosen for the other parts does not lead to a unique name. Note that by using this naming strategy each port will get a unique name. Moreover, each port can only send a performative to one other port and each port can only receive one performative from another port.

The healthcare process involves multiple medical departments, such as radiology and pathology. In order to clearly identify the resource perspective for each task in a proclet class, it is indicated for each task which department and

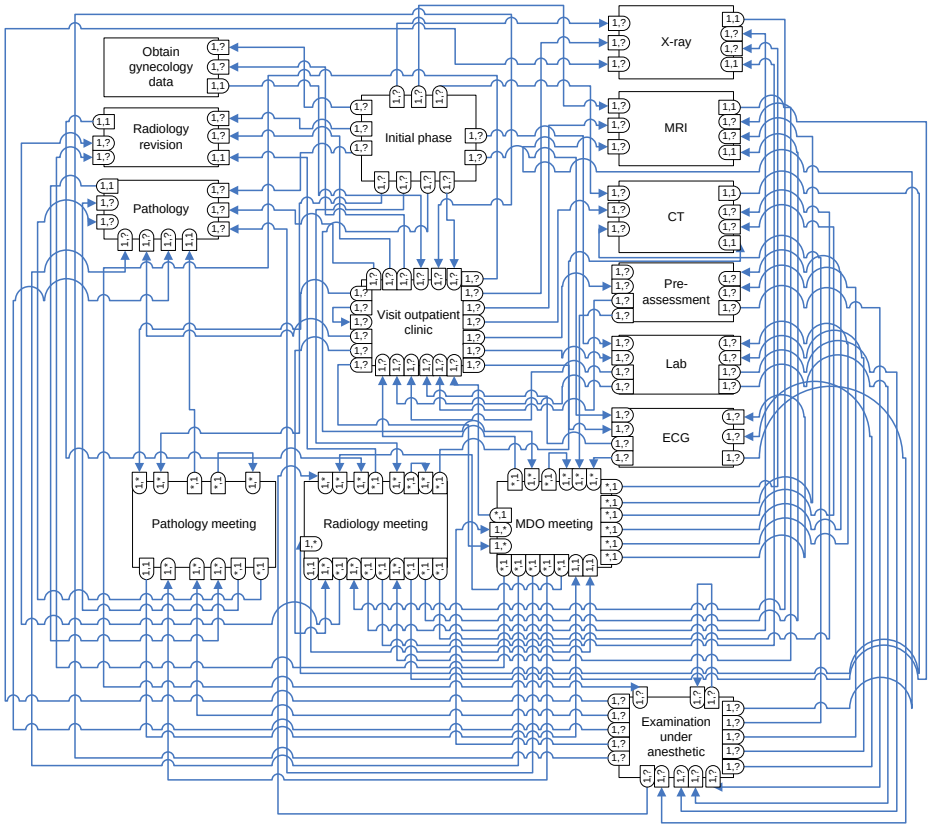


Fig. 7. The proclet classes that are defined for the healthcare process and all of the possible interactions between them.

which role is required. The corresponding organizational model is shown in Figure 8. For example, we can see that for the "gynecological oncology department", the roles "doctor" and "nurse" have been defined, and that for the "radiology" department the roles "radiologist" and "radiology assistant" have been defined.

Note that the proclet classes discussed below (and also the ones that are discussed in Appendix A) are somewhat simplified in comparison to the models produced for the YAWL, FLOWer, ADEPT1, and Declare systems. First of all, the proclet classes do not model all of the tasks that are relevant for a specific workflow. Clearly, our main motivation for defining the proclet classes are to show the *interactions* between these proclets as this is the core focus of the proclet approach. Obviously, by modeling these interactions, information is included which is typically not present in a single monolithic workflow.

Visit Outpatient Clinic We now analyze in detail the "Visit outpatient clinic" proclet class that can be seen in Figure 9. This proclet class deals with a visit

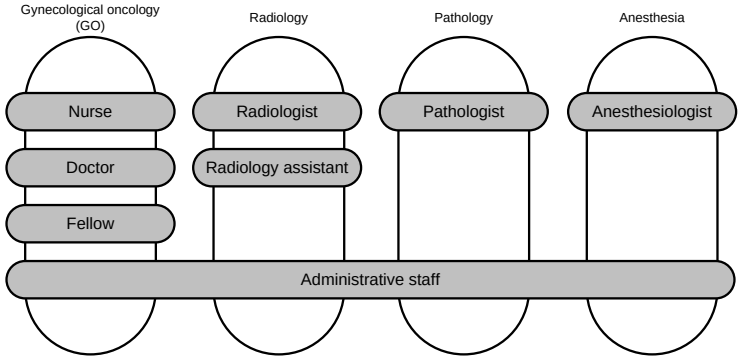


Fig. 8. The organizational model for the healthcare process.

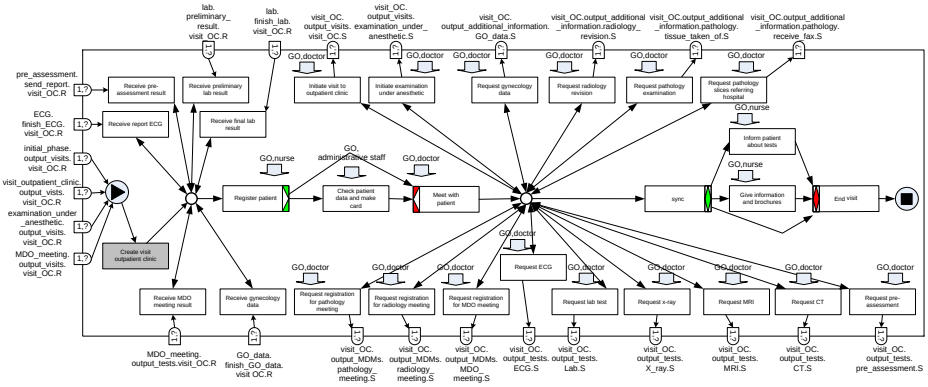


Fig. 9. The "Visit outpatient clinic" proclet class.

by a patient to the outpatient clinic of gynecological oncology in order to see a doctor. The contents of this subprocess has already been discussed in detail in Section 3.

A visit of a patient can be requested at different parts of the process consequently triggering the creation of the respective proclet. This is indicated by the cardinality 1 and multiplicity ? of the ports connected to the input condition. For example, a visit requested during the initial stages of the healthcare process and also during a visit itself or during the MDO meeting. The next few tasks in the proclet class deal with the meeting of the patient with the doctor ("Meet with patient" task). Directly related with such a meeting is that the results of multiple tests ("Receive preliminary lab result", "Receive final lab result", "Receive report ECG" tasks, "Receive pre-assessment result" tasks), additional information ("Receive gynecology data" task), and the result of a MDO meeting ("Receive MDO meeting result" task) might be required as input. The fact that only a selection of them might be required is indicated by the cardinality 1 and multiplicity ? of the associated ports. For example, as input, the outcome of an

MRI and lab test might be necessary but along with the data received from the referring hospital.

Note that the tasks, required for the receipt of all the necessary inputs for a patient meeting, are modeled using a loop. Each performative received is stored in a knowledge base. The proclet continuously inspects this knowledge base and continues with the next step ("Register patient" task) if all required performatives are received.

During a visit to the doctor, it may be decided that several subsequent steps need to be taken in order to diagnose the patient. In general, a doctor can request that additional information is required ("Request gynecology data", "Request radiology revision", "Request pathology examination", "Request pathology slices referring hospital" tasks), that tests need to be undergone by a patient ("Request ECG", "Request lab test", "Request x-ray", "Request MRI", "Request CT", "Request pre-assessment" tasks), and that the patient needs to be discussed during a multidisciplinary meeting ("Request registration for pathology meeting", "Request registration for radiology meeting", "Request registration for MDO meeting" tasks). Moreover, a subsequent visit by the patient might be necessary ("Initiate visit to outpatient clinic", "Initiate examination under anesthetic" tasks). For all of these steps, a doctor makes a selection of those that are necessary. So, either a step is selected once or not at all. This is also indicated by cardinality 1 and multiplicity ? of the associated ports.

Note that in this proclet, *the communication with other proclets is made explicit, i.e. communication is a first-class citizen.* In comparison to Figure 4, interaction with other processes is possible. For example, after the meeting with the doctor, subsequent steps can immediately be triggered (e.g. a lab test), whereas in Figure 4, the subprocess first needs to be finished. Furthermore, in Figure 4, *subprocess dependencies are hidden in the data perspective.* For example, in Figure 4 it is not visible that during the performance of task "Meet with patient", data fields are set, which after completion of the subprocess, cause any subsequent subprocesses to be triggered. In this way, by using proclets, the fourth issue mentioned in Section 3 can be solved.

Note that at run-time information held by proclets might need to be updated or that proclets may need to be canceled. For example, as input to a meeting with a doctor, the result of a lab test might be necessary. However, the result of the lab test may not be available at the moment the meeting should take place. An option is to either cancel the whole proclet involving the lab test or to "relink" the result of the lab test to the next meeting with the patient. At the moment, the models do not cater for the fact that proclets can be updated or even canceled.

It is important to mention that a doctor does not have complete freedom in selecting the next steps to be taken. For example, if a radiology test (MRI, CT, x-ray) is selected or further examination of the radiology material from the referring hospital is required, then the results of these tests need to be discussed during the radiology meeting.

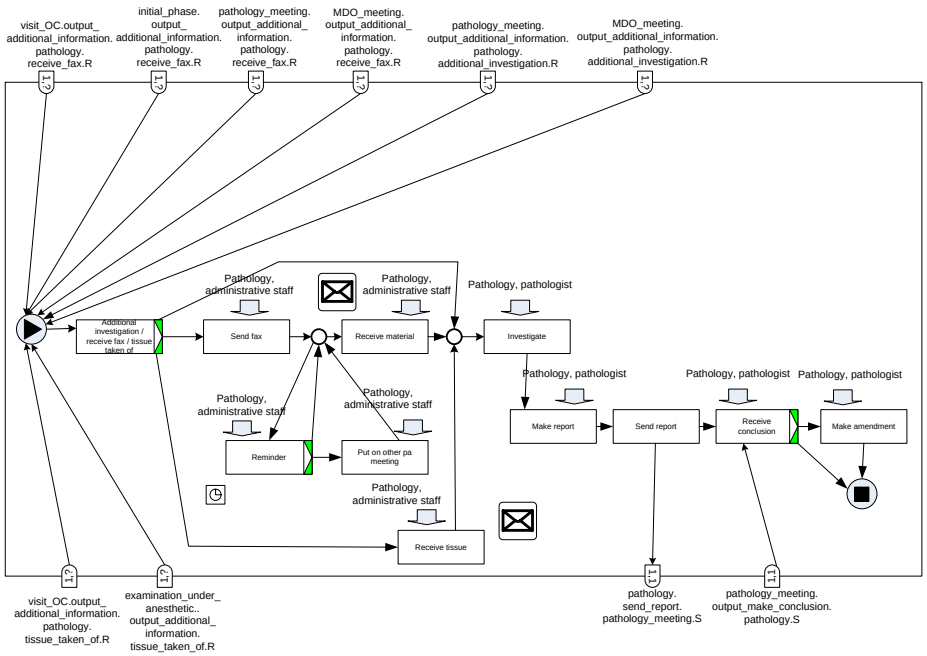


Fig. 10. The "Pathology" proclet class.

In addition to this, a doctor might also influence the way in which the invoked proclets interact based on the way they want the process to execute. For example, it is a general rule that a patient is discussed during the MDO meeting, to decide on the final diagnosis, and that afterwards the patient is informed. Consequently, the MDO meeting needs to take place before the next meeting with the patient. However, a doctor might also first inform the patient first while the patient is also discussed during the MDO meeting. In this scenario, the MDO meeting does not need to take place before the meeting with the patient.

These examples clearly show that the invocation of proclets and the way that proclets interact might be bound by specific domain dependant rules. They may also be influenced based on preferences.

Pathology The "Pathology" proclet class, shown in Figure 10, describes the process in which (1) patient tissue needs to be investigated by a pathologist, (2) a request is made to review pathology material from another hospital, or that (3) a request is made to reinvestigate pathology material. The analysis shows that a proclet class can be used to *represent a workflow process which can handle multiple types of cases*. The resulting model reuses as much as possible, i.e. no duplication of process parts is necessary or creation of separate proclet classes.

For each type of case, specific ports are connected to the input condition in the net. For example, ports of which the names ends with "tissue_taken_of"

are used to create a proclet instance where patient tissue needs to be investigated. Immediately connected to the input condition is the light-grey colored, automatic, "Additional investigation / receive fax / tissue taken of" task. Depending on the performative received, the right path is taken for a specific type of case. Note that as a different path needs to be taken based on the performative received, this illustrates the need for a knowledge base.

In the situation where a request is raised for asking another hospital to send its pathology material for investigation, the "Send fax" task is performed which requires a fax to be send. After this, either the material is received ("Receive material" task) or the other hospital needs to be reminded to send the relevant material to the AMC ("Reminder" task). When the material is received, it can be investigated by a pathologist ("Investigate" task) which prepares a report ("Make report" task) that is discussed during the relevant pathology meeting ("Send report" task). After discussion at this meeting, a performative is returned ("receive conclusion" task) which informs whether any final amendments needs to be made ("Make amendment") or not.

An alternate situation involves a request where a tissue sample taken at the AMC needs to be investigated. After receipt of the tissue ("Receive tissue"), the process follows the same course as for the previous situation, starting from the "Investigate" task.

Finally, a possible alternative involves a request where existing samples need to be reinvestigated, e.g. additional colorings might be necessary. After the request is received, the same steps can be taken as for the two previous situations, starting from the "Investigate" task.

Pathology Meeting The "Pathology meeting" proclet class, shown in Figure 11, describes the weekly meeting in which the gynecological oncology doctors and a pathologist discuss the samples that have been examined by a pathologist and need further review. During this meeting, samples from multiple patients are discussed. For each weekly meeting, a separate proclet is created ("Create pathology meeting" task). In order to discuss a patient sample, this first needs to be registered (tasks starting with "Register for meeting"). This can be done at different points in the process which explains why there are multiple tasks starting with "Register for meeting" each connected to a single port. However, as is indicated by the cardinality 1 and multiplicity * of the associated ports, multiple patients can be registered using the same port. Furthermore, as a consequence of the registration for the "Pathology meeting" proclet, a separate instance of the "Pathology" proclet is created such that the sample can be investigated by a pathologist. The result of this proclet is received via the "Receive pathology report" task. Consequently, there exists a tight connection between one "Pathology meeting" proclet and multiple "Pathology" proclets.

Note that both the tasks for the registration for the meeting and the corresponding receipt of the result are modeled using a loop. By using a loop, multiple performatives can be received, one at a time, as indicated by cardinality 1 and multiplicity * of the associated ports. The "Pathology meeting" proclet con-

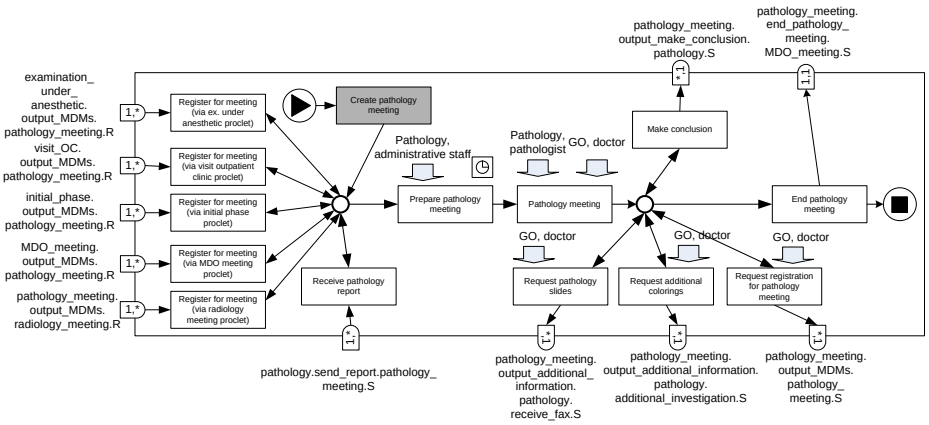


Fig. 11. The "pathology meeting" proclet class.

stantly inspects this knowledge base in order to decide whether all necessary performatives have been received and the process may continue, i.e. the "Prepare pathology meeting" task may be performed in which a pathologist prepares themselves for the pathology meeting that is held afterwards ("Pathology meeting" task).

During this meeting, it might be decided that several subsequent steps need to be taken. For each of the samples that are discussed during the pathology meeting, the corresponding "Pathology" proclet needs to be informed whether an amendment is required or not. This involves sending a performative to each corresponding "Pathology" proclet ("Make diagnosis") as is indicated by the cardinality * of the accompanying port. Moreover, new pathology examinations might be required ("Request Pathology slides", "Request additional colorings" tasks) which subsequently need to be discussed during a later pathology meeting ("Request registration for pathology meeting" task). Here also performatives need to be sent to multiple proclets as indicated by the cardinality * of the accompanying ports. Finally, before destroying the instance, the results of the discussion are transferred to the MDO meeting so that the patient can be discussed during this meeting ("End pathology meeting" task).

As becomes clear from this analysis, *in this proclet class we are dealing with a different level of aggregation (a group of patients) than the previous proclet classes, and performatives are received from proclet classes which are at a lower level of aggregation (a specific service delivered for a patient)*. Using proclets we can easily handle these differences in the level of aggregation whereas most workflow management systems force one to depict the process at an arbitrarily chosen level and to describe them in terms of one monolithic workflow. Obviously, issue number one, mentioned in Section 3, can be solved using proclets.

Moreover, for this proclet several *one-to-many* relationships exist within the "Pathology" proclet. For example, via task "Request additional colorings", for

multiple patients it can be requested that existing samples be reinvestigated. The output port of this task has cardinality $*$, indicating that the performative is sent to potentially multiple recipients. In the class diagram of Figure 6, this relationship is also indicated by the "follows_5" association. Clearly, using proclets *it can easily be captured that there one-to-many or many-to-many relationships exists between entities in a workflow, whereas this is impossible to capture using existing workflow approaches.* So, issue number three, mentioned in Section 3, can be solved.

5 Related Work

The first WfMSs were developed in the early 1970's. See for example the OfficeTalk system of Skip Ellis [20], an office automation system based on Petri Nets. However, only in the late nineties these systems became more mature and used in practice. Currently, there are several hundred WfMS's and workflow technology has become an integral part of numerous products, including Enterprise Resource Planning (ERP) (e.g. SAP/R3), Product Data Management (PDM), and Customer Relationship Management (CRM) systems.

Currently, administrative processes, which tend to be rather rigid, can be well supported by WfMSs. However, as indicated in [34, 35], so called "careflow systems", systems used for supporting care processes in hospitals, have special demands with regard to workflow technology. Successful implementations of workflow systems in healthcare do exist, but "widespread" adoption and dissemination is the exception rather than the rule [32]. One of the problems that must be dealt with in order to effectively support healthcare processes using WfMS's is that process flexibility needs to be provided by the system [29, 37]. Unfortunately, current workflow systems fall short in this area, an observation often reported in the literature [8, 10, 21, 22].

One of the problems related to flexibility is that for a given patient a lot of concurrent processes need to run in conjunction with each other [19, 24]. When complex care needs to be delivered, co-operation between various clinicians across different medical specialties and departments is needed [28], e.g. by having multidisciplinary meeting in which doctors from several medical disciplines discuss the status of a patient. In this paper, these processes can be characterized by *weakly-connected interacting lightweight workflows* in which communication and collaboration between instances of these processes is of particular importance.

Contemporary languages and systems provide limited support for the execution of lightweight interacting workflows. Instead, one is forced to squeeze real-life processes into a single "monolithic overarching workflow" which describes how an individual case is handled in isolation. In doing this, the overview is lost and flexibility is restricted. This issue has been recognized in literature [4, 5, 14, 23, 31] and is not limited to the healthcare domain. It also applies in other areas, e.g. the automotive domain [31], or when reviewing papers for a conference [5].

When applied to lightweight interacting workflows, the proclet approach supports the following considerations: (1) different processes interacting with each

other, (2) processes operating at different levels of aggregation, and (3) batch-oriented tasks. There are a limited range of alternate approaches to deal with these issues [23].

The Corepro framework [30, 31] allows for automatic generation and coordination of individual processes, operating at different levels of aggregation, based on their underlying data structure. The initial number of process instances created is decided at run-time. However, the creation of new instances at runtime is possible, but requires an ad-hoc change to the related data structures. For the proclet approach, the number of process instances created is based on post-conditions, evaluated at runtime.

In [16–18], a two-tier, goal-driven model for workflow processes in the healthcare domain is presented. A goal-ontology, presented as a directed acyclic graph, is utilized to represent the business model at the upper level and is decomposed into an extended Petri-net model for the lower level workflow schema. A mapping is defined from the goal-graph to (sub)processes and activities such that each of the (sub)processes is designed such that it achieves one of the upper level goals. This approach leads to a hierarchy of process models with a number of the top-level goals being implemented through subprocesses. However, there is no interaction between subprocesses instead of the classical way in which hierarchical processes communicate with each other which is only in a top-down fashion.

In [13–15, 33], the concept of artifacts is used to distinguish different levels of aggregation. However, the content of one or more artifacts can only be changed by the execution of an activity. Consequently, aggregation issues are only addressed at the activity level instead of at process level.

Batch-oriented tasks are tasks that are based on groupings of lower aggregation elements. The concept of a batch-oriented task is already coined in [12] in order to allow for a task that is executed for multiple instances at the same time. In [36], the problem is defined and deliberations are provided on the required technology support to deal with the issue.

Finally, in [11], worklets are presented which can be seen as micro workflows. Specific activities in a process are linked to a repertoire of possible actions. Based on the properties of the case and other context information, the required action is chosen. The selection process is based on a set of rules. During enactment it is also possible to add new actions to the repertoire. However, these actions can only be enacted at specific parts of the process which need to be decided at design-time.

6 Discussion and Conclusion

In this paper, we have studied the gynecological oncology healthcare process and elaborated on how it can be modeled using existing workflow languages, such as YAWL, FLOWer, Declare, and ADEPT1. Moreover, we have examined how the same process can be modeled using the proclet framework. If we compare the

proclet framework with existing workflow languages, the following differences can be observed.

- Real-life healthcare care processes are often fragmented and composed of separate but intertwined life-cycles running at different speeds. These processes can be effectively described using proclets, in which interaction is considered as a first-class citizen, instead of straightjacketing them into one monolithic workflow.
- Real-life healthcare processes often operate at different levels of granularity. Existing workflow languages cannot take these differences into account, however, by using proclets this is easily supported. Moreover, one-to-many and many-to-many relationships that exist between entities in a workflow, can be captured.

In healthcare, for a given patient, a lot of concurrent processes can run in conjunction with each other. These processes can be created at any point in time and can also be terminated at any point in time. Moreover, these processes interact in different ways with each other. We have characterized these kinds of processes as *weakly-connected interacting lightweight workflows*. Moreover, steps in a healthcare process may either operate at the level of a single patient but also at the level of a group of patients. In other words, these processes may rely on information that is at different levels of aggregation. Given that these characteristics of healthcare processes can be handled using proclets, we may assume that these processes can be effectively modeled using proclets.

However, some challenges still remain. First of all, for the diagnosis and treatment of a patient it can not be decided beforehand how many proclets will need to be instantiated for the patient and the way in which they will interact with each other. This only becomes clear at run-time. Future work needs to investigate how to "connect" proclets at run-time and ways in which they can be enacted. It is important that a proclet is aware of performatives that it will receive. If these can not be handled efficiently, escalation actions need to be taken (e.g. continue without waiting for the performative that needs to be received or cancelation of a proclet).

Moreover, several other interesting issues have been identified. In the gynecological oncology healthcare process many different tests can be needed by patients. These tests can be instantiated via different proclets and the results can be received by different proclets. Clearly, for each distinct point that such a test can be created or the result may be received a specific task with an outgoing or incoming port is required. In the future new diagnostic tests might become available which need to be included in the respective proclets. Therefore, we see an interesting link with the worklet approach described in the related work section. Using the worklet approach, it may be possible to invoke tasks with input or output ports dynamically in proclets instead of needing to model them explicitly.

For the proclet classes described, there is also a link with calendar-based scheduling support [26]. Today's WfMSs offer work-items to users through specific work-lists in which participants select the work-items they will perform.

However, no calendar-based scheduling is offered for these workitems. In [26] it is investigated how a WfMS can be augmented with scheduling facilities such that appointments can be scheduled for these kinds of tasks. Moreover, these appointments are scheduled in the calendars of the participants involved in the actual performance of the task in order to ensure that they occur at a precise pre-agreed time suitable for all of the participants involved. Clearly, in several proclet classes, we also find tasks for which a concrete appointment involving specific resources needs to be made. For example, the "Meet with patient" task in the "Visit outpatient clinic" proclet class is a task which needs to be scheduled for both a doctor and a patient. Furthermore, the "Pathology meeting" task in the "Pathology meeting" proclet class is also a task which needs to be scheduled as a pathologist and several gynecological oncology doctors need to be available at the same time.

Today's information systems record many of the events relevant to a business process. Obviously, there is an abundance of event data, and new and powerful techniques such as process mining [9] provide many opportunities to analyze and improve business processes. In this context, for all process instances that are related to a given case, events are logged. Future work should focus on collecting and correlating information about tasks that were executed for different process instances, but between which a certain relationship exists.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
2. W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.
3. W.M.P. van der Aalst. Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 1–65. Springer-Verlag, Berlin, 2004.
4. W.M.P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Workflow Modeling using Proclets. In O. Etzion and P. Scheuermann, editors, *7th International Conference on Cooperative Information Systems (CoopIS 2000)*, volume 1901 of *Lecture Notes in Computer Science*, pages 198–209. Springer-Verlag, Berlin, 2000.
5. W.M.P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Proclets: A Framework for Lightweight Interacting Workflow Processes. *International Journal of Cooperative Information Systems*, 10(4):443–482, 2001.
6. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, 2002.
7. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
8. W.M.P. van der Aalst and S. Jablonski. Dealing with Workflow Change: Identification of Issues and Solutions. *International Journal of Computer Systems, Science, and Engineering*, 15(5):267–276, 2000.

9. W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M.S. Song, and H.M.W. Verbeek. Business Process Mining: An Industrial Application. *Information Systems*, 32(5):713–732, 2007.
10. W.M.P. van der Aalst, M. Weske, and D. Grünbauer. Case Handling: A New Paradigm for Business Process Support. *Data and Knowledge Engineering*, 53(2):129–162, 2005.
11. M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Facilitating Flexibility and Dynamic Exception Handling in Workflows. In O. Belo, J. Eder, O. Pastor, and J. Falcao e Cunha, editors, *Proceedings of the CAiSE'05 Forum*, pages 45–50. FEUP, Porto, Portugal, 2005.
12. P. Barthelmess and J. Wainer. Workflow systems: a few definitions and a few suggestions. In N. Comstock and C.A. Ellis, editors, *Proceedings of the Conference on Organizational Computing Systems - COOCS'95*, pages 138–147, Milpitas, California, September 1995. ACM Press.
13. K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F.Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4):703–721, 2007.
14. K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 288–304. Springer-Verlag, Berlin, 2007.
15. K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 288–304. Springer-Verlag, Berlin, 2007.
16. E.D. Browne, M. Schrefl, and J.R. Warren. A Two Tier, Goal-Driven Workflow Model for the Healthcare Domain. In *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS 2003)*, pages 32–39, 2003.
17. E.D. Browne, M. Schrefl, and J.R. Warren. Activity Crediting in Distributed Workflow Environments. In *Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS 2004)*, 2004.
18. E.D. Browne, M. Schrefl, and J.R. Warren. Goal-Focused Self-Modifying Workflow in the Healthcare Domain. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS-37 2004) - Track 6*. IEEE Computer Society Press, 2004.
19. P. Dadam, M. Reichert, and K. Kuhn. Clinical Workflows - The Killer Application for Process-oriented Information Systems? In W. Abramowicz and M.E. Orlowska, editors, *BIS2000 - Proc. of the 4th International Conference on Business Information Systems*, pages 36–59, Poznan, Poland, April 2000. Springer-Verlag.
20. C.A. Ellis. Information Control Nets: A Mathematical Model of Office Information Flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, pages 225–240, Boulder, Colorado, 1979. ACM Press.
21. C.A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock, C. Ellis, R. Kling, J. Mylopoulos, and S. Kaplan, editors, *Proceedings of the Conference on Organizational Computing Systems*, pages 10 – 21, Milpitas, California, August 1995. ACM SIGOIS, ACM Press, New York.
22. M. Klein, C. Dellarocas, and A. Bernstein, editors. *Adaptive Workflow Systems*, Special Issue of Computer Supported Cooperative Work, 2000.

23. V. Künzle and M. Reichert. Towards Object-aware Process Management Systems: Issues, Challenges, Benefits. In T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, and R. Ukor, editors, *Proc. 10th Int'l Workshop on Business Process Modeling, Development, and Support (BPMDS'09)*, volume 29 of *Lecture Notes in Business Information Processing*, pages 197–210. Springer-Verlag, Berlin, 2009.
24. R. Lenz and M. Reichert. IT Support for Healthcare Processes - Premises, Challenges, Perspectives. *Data and Knowledge Engineering*, 61:49–58, 2007.
25. R.S. Mans, W.M.P. van der Aalst, N.C. Russell, P.J. Bakker, A.J. Moleman, K.B. Lassen, and J.B. Jorgensen. From Requirements via Colored Workflow Nets to an Implementation in Several Workflow Systems. *Transactions on Petri Nets and Other Models of Concurrency III (to appear)*, 2009.
26. R.S. Mans, N.C. Russell, W.M.P. van der Aalst, A.J. Moleman, and P.J. Bakker. Schedule-Aware Workflow Management Systems. In D. Moldt, editor, *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE09)*, pages 81–96, 2009.
27. R.S. Mans, W.M.P. van der Aalst, N.C. Russell, and P.J.M. Bakker. Flexibility Schemes for Workflow Management Systems. In *Lecture Notes in Business Information Processing*, volume 17, pages 361–372, 2009.
28. R.S. Mans, W.M.P. van der Aalst, N.C. Russell, A.J. Moleman, P.J. Bakker, and M.W. Jaspers. *YAWL - A State-of-the-Art Open Source BPM Environment*, chapter YAWL4Healthcare. Springer Verlag, 20009.
29. L. Maruster, W.M.P. van der Aalst, A.J.M.M. Weijters, A. van den Bosch, and W. Daelemans. Automated Discovery of Workflow Models from Hospital Data. In C. Dousson, F. Höppner, and R. Quiniou, editors, *Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data*, pages 32–36, 2002.
30. D. Müller, M. Reichert, and J. Herbst. Data-Driven Modeling and Coordination of Large Process Structures. In Z. Bellahsène and M. Léonard, editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803 of *Lecture Notes in Computer Science*, pages 131–149. Springer-Verlag, Berlin, 2007.
31. D. Müller, M. Reichert, and J. Herbst. A New Paradigm for the Enactment and Dynamic Adaptation of Data-Driven Process Structures. In R. Meersman and Z. Tari, editors, *Advanced Information Systems Engineering*, volume 5074 of *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag, Berlin, 2008.
32. M. Murray. Strategies for the Successful Implementation of Workflow Systems within Healthcare: A Cross Case Comparison. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pages 166–175, 2003.
33. A. Nigam and N.S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.
34. S. Quaglini, M. Stefanelli, A. Cavallini, G. Micieli, C. Fassino, and C. Mossa. Guideline-based Careflow Systems. *Artificial Intelligence in Medicine*, 20(1):5–22, 2000.
35. S. Quaglini, M. Stefanelli, G. Lanzola, V. Caporusso, and S. Panzarasa. Flexible Guideline-based Patient Careflow Systems. *Artificial Intelligence in Medicine*, 22(1):65–80, 2001.
36. S. Sadiq, M. Orlowska, W. Sadiq, and K. Schulz. When workflows will not deliver: The case of contradicting work practice. In W. Abramowicz, editor, *Proc. BIS'05*, 2005.
37. M. Stefanelli. Knowledge and Process Management in Health Care Organizations. *Methods Inf Med*, 43:525–535, 2004.

38. S.A. White. Introduction to BPMN. *BPTrends*, 2004.

A Appendix

In this section, the "Initial phase", "Examination under anesthetic", "Radiology meeting", "MDO meeting", "Obtain gynecology data", "Radiology revision", "CT", and "Pre-assessment" proclets classes will be discussed respectively.

A.1 Initial Phase

The "Initial phase" proclet class concerns the initial stages of the gynecological oncology healthcare process in which the necessary preparations are made for the first visit of a patient. The process starts when a doctor at a referring hospital calls a nurse or doctor at the AMC and obtaining patient related information ("Receive info from doctor referring hospital" task). In order to register the patient at AMC and to be able to trigger subsequent steps, patient data is entered into the hospital information system ("Enter patient data into system" task) and a document and stickers are made ("Make document and stickers" task). After this, several steps are performed which involve the first visit of the patient to the hospital. These include the initiation of the first visit itself ("Initiate visit to outpatient clinic" task), tests that need to be undergone by a patient ("Request ECG", "Request lab test", "Request x-ray", "Request MRI", "Request CT", "Request pre-assessment" tasks), the corresponding forms that need to be filled in ("Fill in application forms" task), and additional information that needs to be gathered ("Request gynecology data", "Request radiology revision", "Request pathology slices referring hospital" tasks). Moreover, depending on the additional information that is required, the patient may need to be discussed during a multidisciplinary meeting ("Request registration for pathology meeting", "Request registration for radiology meeting", "Request registration for MDO meeting" tasks). Note that the selection of tests, additional information, multidisciplinary meetings, and the next visit of the patient proceeds in much the same way as for the "Visit outpatient clinic" proclet class. The only small difference is that during the initial phase, there is no examination under anesthetic.

The last steps involve informing the patient about the first visit to the hospital ("Call patient") and sending a confirmation of the appointment by mail ("Send confirmation appointment").

Note that during the initial phase some processes can be started immediately (e.g. tests) in order to speed up the whole process even before the patient has seen a doctor. This has the consequences that for some proclets there is not yet enough information available about the way they need to interact with other proclets or that proclets may need to be canceled lateron. For example, during the initial phase a lab test is triggered for a patient which means that the patient needs to visit the lab after the first meeting with the doctor. At this point it is still not known whether the result of the test is needed for the second visit

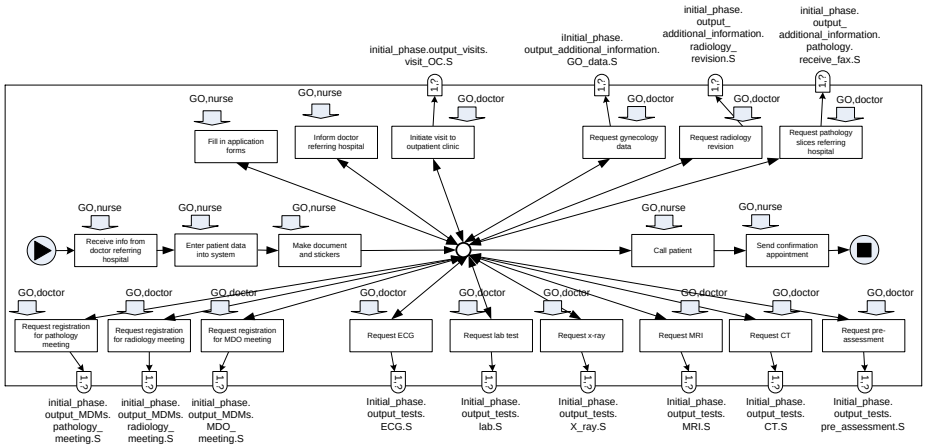


Fig. 12. The "Initial phase" proclet class.

of the patient or for a discussion of the patient at an MDO meeting. At the time the doctor sees the patient for the first time, it is determined when the results need to be available. At this time the lab proclet needs is updated with information so that it is known with which proclet it should interact. Another possibility is that during the initial phase an MRI test is ordered meaning that the MRI test can only be done after the first meeting with the doctor. However, during the first meeting, the doctor decided that a CT test was sufficient instead of an MRI. This requires the triggering of a CT proclet and the cancelation of the MRI proclet. Furthermore, the cancelation of the MRI proclet might require other dependent proclets to be updated or canceled. At the moment, the models do not cater for the fact that proclets may be updated or even canceled.

A.2 Examination under Anesthetic

The "Examination under anesthetic" proclet class, Figure 13, concerns the admission of a patient to the hospital for an examination under general anesthesia. The proclet starts when a request is made for an examination under anesthetic ("Create examination under anesthetic" task) which can be triggered either during a visit to the outpatient clinic or during the process of another examination under anesthetic. In order to prepare for the examination, the results of some preceding tests might be necessary ("Receive preliminary lab result", "Receive final lab result", "Receive report ECG" tasks, "Receive pre-assessment result" tasks) the procurement of which proceeds in a similar way as for the "Initial phase" proclet class. The results of these tests need to available before the admission of the patient to the nursing ward ("Admission to nursing ward" task). Prior to the examination, the doctor, who will do the examination, has an initial meeting with the patient to get acquainted ("Acquaintance meeting with

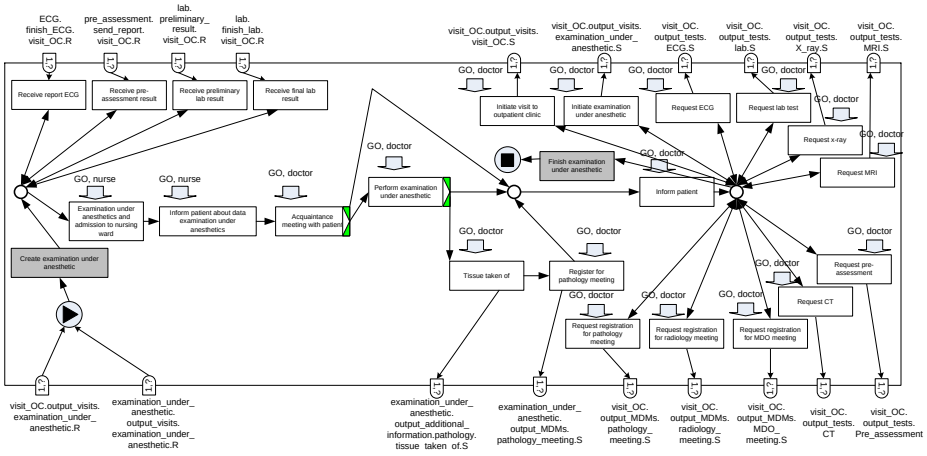


Fig. 13. The "Examination under anesthetic" proclet class.

patient" task). At this point, there is the opportunity not to proceed with the examination and to cancel the process.

During the examination ("Perform examination under anesthetic" task), some tissue might be taken from the patient, which needs to be investigated by a pathologist. This requires both the creation of a "Pathology" proclet instance in order to examine the tissue ("Tissue taken of" task) and the registration of the patient for a pathology meeting in order to discuss the examination results ("Register for pathology meeting").

After the examination under general anesthesia, the patient is informed of the results ("Inform patient" task). Several subsequent steps can be taken after which the process fragment is finished. These include tests that need to be undergone by a patient ("Request ECG", "Request lab test", "Request x-ray", "Request MRI", "Request CT", "Request pre-assessment" tasks), discussion of the patient during a multidisciplinary meeting ("Request registration for pathology meeting", "Request registration for radiology meeting", "Request registration for MDO meeting" tasks), and arranging a subsequent visit of the patient to the hospital ("Initiate visit to outpatient clinic" task). The selection of these next steps proceeds in a similar way as for the "Initial phase" proclet class.

A.3 Radiology Meeting

The "Radiology meeting" proclet class, whose class is shown in Figure 14, describes the weekly meeting carried out after the pathology meeting, in which the gynecological oncology doctors and a radiologist discuss the results of the radiology examinations that have been performed. These radiology examinations include an MRI, CT, X-ray and the revision of material of radiology examinations that are carried out in other hospitals. The radiology meeting proceeds in a similar way to the process for a pathology meeting discussed earlier in this

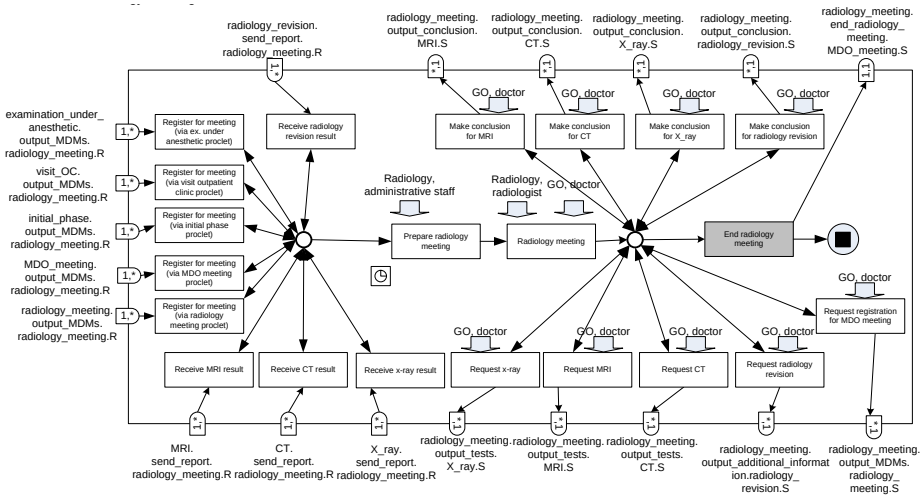


Fig. 14. The "Radiology meeting" proclet class.

paper. The only notable differences are that now instances of an "MRI", "CT", "X-ray" and "Radiology revision" are created in which the corresponding results are synchronized via the "Receive MRI result", "Receive CT result", "Receive X-ray result", and "Receive radiology revision result" tasks. Furthermore, after the execution of the radiology meeting ("Radiology meeting" task), instead of pathology examinations, now only radiology related tasks can be initiated via the "Request X-ray", "Request MRI", "Request CT", and"Request radiology revision" tasks. Note that the aggregation we are dealing with is a group of patients instead of an individual patient.

A.4 MDO Meeting

The "MDO meeting" proclet class, shown in Figure 15, describes the weekly meeting, carried out after the radiology meeting, in which the status of several patients is discussed and a decision is made about the next steps in the diagnostic process for these patients. During the meeting it is mentioned which new patients have been registered during the previous week. During the MDO meeting several medical disciplines are involved in order to decide how the continuation of the treatment process will proceed.

The process proceeds in a similar way to the processes that are carried out for the pathology and radiology meeting, the only difference being that different inputs and outputs are possible. Moreover, it is often the case that after the MDO meeting, a patient has a meeting with a doctor to inform them about the outcome of a meeting. Therefore, before destroying the proclet instance, as indicated by the cardinality * at the accompanying port, a performative is sent to the "Visit outpatient clinic" proclet of the patients that need to be informed.

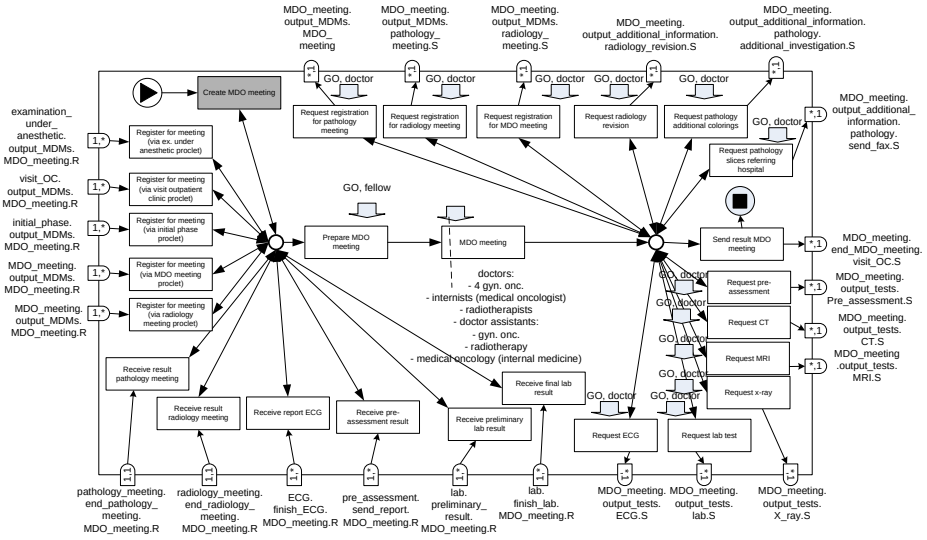


Fig. 15. The "MDO meeting" proclet class.

Note that the aggregation level we are dealing with is a group of patients instead of a single patient. The next proclet classes that will be discussed all have the same aggregation level as the "Visit outpatient clinic", "Initial phase", and "Examination under anesthetic" proclet classes. They correspond to a service that is delivered for a patient.

A.5 Obtain Gynecology Data

The "Obtain gynecology data" proclet class, shown in Figure 16, is a simple proclet class that deals with the retrieval of patient related data from the gynecological oncology department of the hospital that referred them to the AMC for treatment. The trigger for this proclet class is either created during the initial phases of the process or during a visit of the patient to the hospital which is indicated by the cardinality 1 and multiplicity ? of the accompanying ports connected to the input condition in the net. Subsequently, a nurse in the gynecological oncology department contacts the referring hospital to request the material to be sent to the AMC ("Telephone call" task). Either the material is received ("Receive" task) and put in a document ("Put gynecology data in document") or a nurse again calls the referring hospital to remind them to send the material to the AMC hospital ("Reminder" task). Finally, when the received material is put into the document of the patient it can be used as input for the next visit of the patient at the outpatient clinic ("Finish obtain gynecology data"). Note that this interaction always takes place as indicated by the cardinality 1 and multiplicity 1 of the accompanying port.

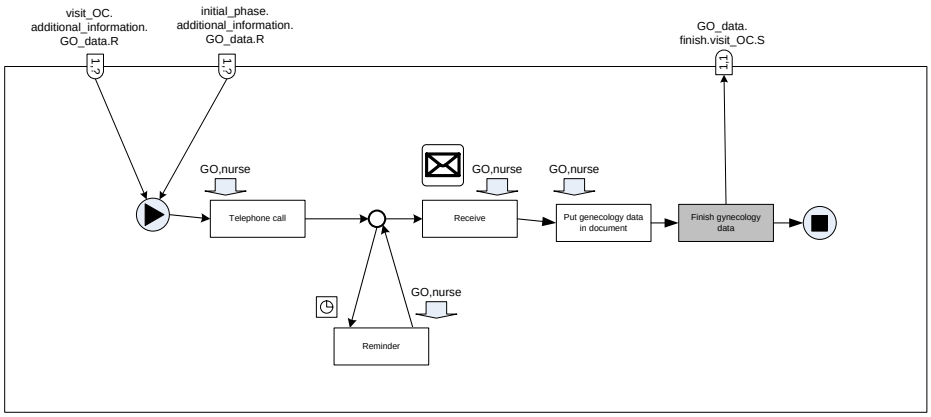


Fig. 16. The "Obtain gynecology data" proclet class.

Note that typically it is expected that the gynecology data is received before the first visit of the patient to the AMC hospital. This has the consequence that from the "Finish gynecology data" task a performative is sent to the "Visit outpatient clinic" proclet instance that is started as a consequence of the first visit of the patient. However, if the receival of the gynecology data is delayed and it cannot be delivered in time, then it can be chosen to either send the performative to another proclet instance, i.e. to "relink" the proclet instance to another one, or that no performative needs to be sent anymore (so the "Obtain gynecology data" proclet continues as a disconnected thread).

A.6 Radiology revision

The "Radiology revision" proclet class, shown in Figure 17, describes a process which proceeds in a similar way to the one described by the "Obtain gynecology data" proclet class. The main differences are that now the radiology department of the referring hospital is requested to send its radiology data to the AMC so that it can be investigated. Also a reminder may be needed, so a similar discussion applies as for the "Obtain gynecology data" proclet class. After receiving the material ("Receive" task), a form is filled in so that everything can be sent to the radiology department ("Fill in form and send to radiology" task) in order to be investigated by a radiologist ("Investigate" task). Afterwards, a report is made ("Make report" task) which is sent to the "Radiology meeting" proclet instance ("Send report" task) where it needs to be discussed. After discussion during the radiology meeting, a performative is received back ("Receive conclusion" task) which indicates whether an amendment needs to be made ("Make amendment" task) or not (in this case the proclet instance finishes immediately).

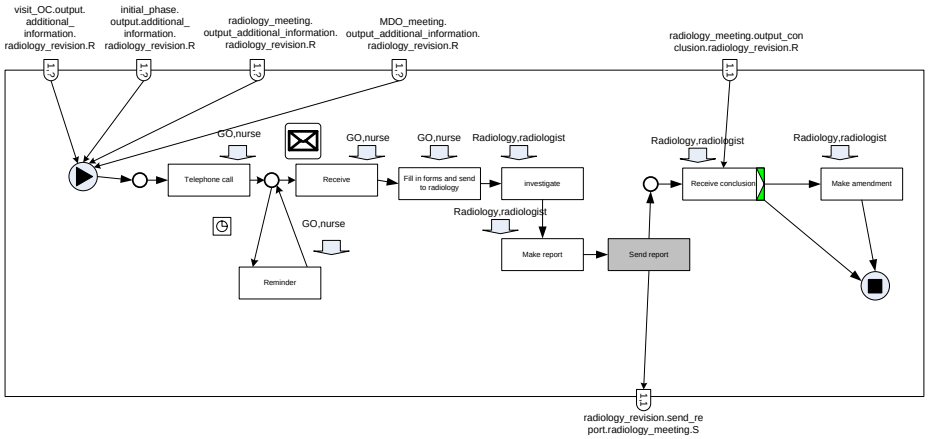


Fig. 17. The "Radiology revision" proclet class.

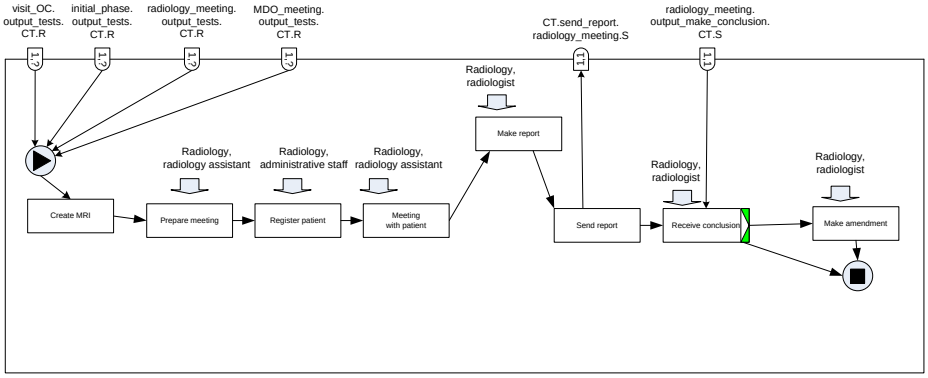


Fig. 18. The "CT" proclet class.

A.7 CT

The "CT" proclet class, shown in Figure 18, deals with the necessary preparations for a CT examination to be undergone by a patient and the discussion of the results afterwards at the radiology meeting for which the CT examination is registered. The process proceeds in a similar way to the "Radiology revision" proclet class, however, instead of requesting radiology material, a CT examination is prepared ("Prepare meeting" task) and performed ("Meeting with patient" task). This involves the registration of the patient when he / she arrives ("Register patient" task).

A.8 Pre-assessment

The "Pre-assessment" proclet class, shown in Figure 19, models the process of a pre-assessment examination that needs to be undergone by a patient. A pre-

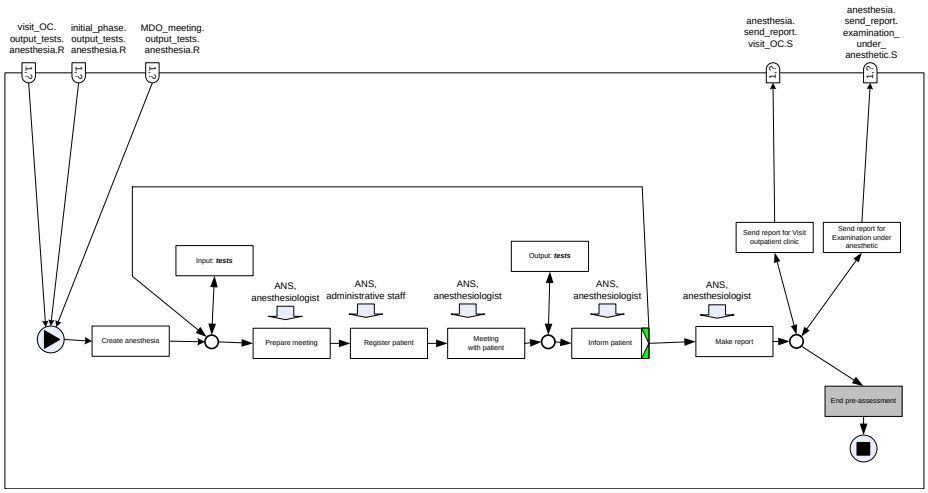


Fig. 19. The "Pre-assessment" proclet class.

assessment test is performed by an anesthesiologist and is required in order to perform an examination under anesthetic or to perform a surgery. Note that this process proceeds in more or less the same way as the process modeled by the "CT" proclet class. However, there are some notable differences.

First of all, the anesthesiologist decides whether it is safe to perform a surgery or examination under anesthetic as general anesthetics are used. In order to be able to make this decision, the anesthesiologist might require the input of another medical specialist (e.g. cardiologist, lung doctor). In other words, the anesthesiologist might require that additional tests are undergone by a patient. As an anesthesiologist can trigger a complete different plethora of tests for a patient, we decided to not model these interactions but to represent the triggering of these tests and the receival of the accompanying result by the "Output: tests" and "Input: tests" tasks respectively. Moreover, seen from the perspective of the requester, they may not be aware of the fact that additional tests are invoked. Instead, the requester may only be aware that the report has not yet been received.

During the performance of the "Inform patient" task a decision is made whether new tests need to be requested and discussed during a subsequent meeting or that no additional tests are needed and a report can be made ("Make report") and sent back to the requester ("Send report for Visit outpatient clinic" and "Send report for Examination under anesthetic" tasks).