Workflow Support Using Proclets: Divide, Interact, and Conquer

W.M.P. van der Aalst and R.S. Mans and N.C. Russell
Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
w.m.p.v.d.aalst@tue.nl

Abstract

Classical workflow notations primarily support monolithic processes. They are able to describe the life-cycle of individual cases and allow for hierarchical decomposition. Unfortunately, real-life processes are fragmented and are often composed of separate but intertwined life-cycles running at different speeds and coping with different levels of granularity. The proclets framework was one of the first formalisms to acknowledge this. Proclets are lightweight interacting processes *that can be used to divide complex entangled processes into simple fragments and, in doing so, place increased emphasis on interaction-related aspects of workflows. This paper describes the proclets approach and presents an application of this approach to the gynecological oncology workflow process at a major Dutch hospital.*

1 Introduction

Although most information systems are "process aware" the support for various aspects of operational processes leaves much to be desired. For example, workflow technology is mostly used to automate repetitive well-structured processes. There is little support for less structured processes that require more flexibility. As a consequence of the widespread adoption of database technology in the seventies, the development of information systems is predominantly *data-centric*, i.e., the design and implementation starts with object/information modeling. However, since the nineties, consultants and vendors have been advocating more *process-centric* approaches. Today, the majority of larger organizations spend considerable time identifying and modeling processes. Business Process Management (BPM) techniques and tools support these more process-centric approaches and have received considerable attention. However, when looking at the actual implementations of information systems there is still a *mismatch between the processes modeled and reality* (i.e., the real systems and processes).

This mismatch has several reasons. One is that most actors have a simplistic and often incorrect view of the processes in which they are involved. Process mining techniques can be used to provide a more realistic view of their actuality [3]. It is often the case that processes are more complex and "spaghetti-like" than we expect. Reality cannot be captured in a structured monolithic workflow model. Another reason is that an effective balance/integration between/of the data perspective and the process perspective is missing. It is impossible to separate these perspectives. Moreover, it is obvious that the process-centric approaches used in the initial phases of workflow specification do not fit well with the predominant data-centric implementation approaches.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Currently, there is a renewed interest in the mismatch described above. This is illustrated by the recent NSF Workshop on Data-Centric Workflows that took place in Arlington (Virginia) in May 2009. (See [5] for the workshop report.) During this workshop there was consensus that processes cannot be straightjacketed in monolithic workflows and that the interplay between data and control-flow is essential.

In this paper, we advocate the use of *proclets*, one of the first modeling languages to address these problems [1, 2]. Proclets can be seen as lightweight interacting processes. The proclets framework be used to integrate data-centric and process-centric approaches at both the design and implementation level [1, 2]. To illustrate the framework and the ideas behind it, the gynecological oncology workflow at the AMC hospital in The Netherlands is modeled in terms of proclets.

In the remainder of this paper we first discuss the limitations of "monolithic workflows" (Section 2), followed by a brief introduction to the proclets framework (Section 3). In Section 4, we describe the application of proclets at a Dutch hospital. Section 5 concludes the paper.

2 Limitations of Monolithic Workflows

Proclets aim to address the following problems that existing workflow approaches are currently facing:

- Models need to be *artificially flattened* and are unable to account for the mix of *different perspectives and granularities* that coexist in real-life processes.
- Cases need to be *straightjacketed into a monolithic workflow* while it is more natural to see processes as intertwined loosely-coupled object life-cycles.
- 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.
- It is difficult to model interactions between processes, i.e., *interaction is not a first-class citizen* in most process notations.

In the remainder, we use proclets to address the problems that are experienced in monolithic workflows.

3 Proclets: Lightweight Interacting Processes

A *proclet* can be seen as a lightweight workflow process able to interact with other proclets that may reside at different levels of aggregation [1, 2]. One can think of proclets as objects equipped with an explicit life-cycle or as active documents. Recently, this has been referred to as artifact centric workflows/processes [4]. Proclets interact via *channels*. A channel is the medium used to transport messages from one proclet to another. Via a channel, a message can be sent to a specific proclet or a group of proclets (i.e., multicast). Such messages are called *performatives* since they correspond to explicit actions such as those found in speech act theory and the language/action perspective. Based on the properties of the channel, different kinds of interaction are supported, e.g., push/pull, synchronous/asynchronous, and verbal/non-verbal. Proclets are connected to channels via *ports*. Each port has two attributes: (a) its *cardinality* and (b) its *multiplicity*. The cardinality specifies the number of recipients of performatives exchanged via the port. The multiplicity specifies the number of performatives exchanged via the port during the lifetime of any instance of the class. The life-cycle of a particular type of proclet and its ports are specified in terms of a *proclet class*. Using these concepts, complex monolithic workflow definitions describing the control flow of an entire process can be broken up into smaller interacting proclets, i.e., there is a *shift in emphasis from control to communication*.

Proclets were introduced in the late nineties [1, 2]. In the original publications a variant of Petri nets, called *workflow nets*, was used as a basis. However, the main ideas are independent of the control-flow language

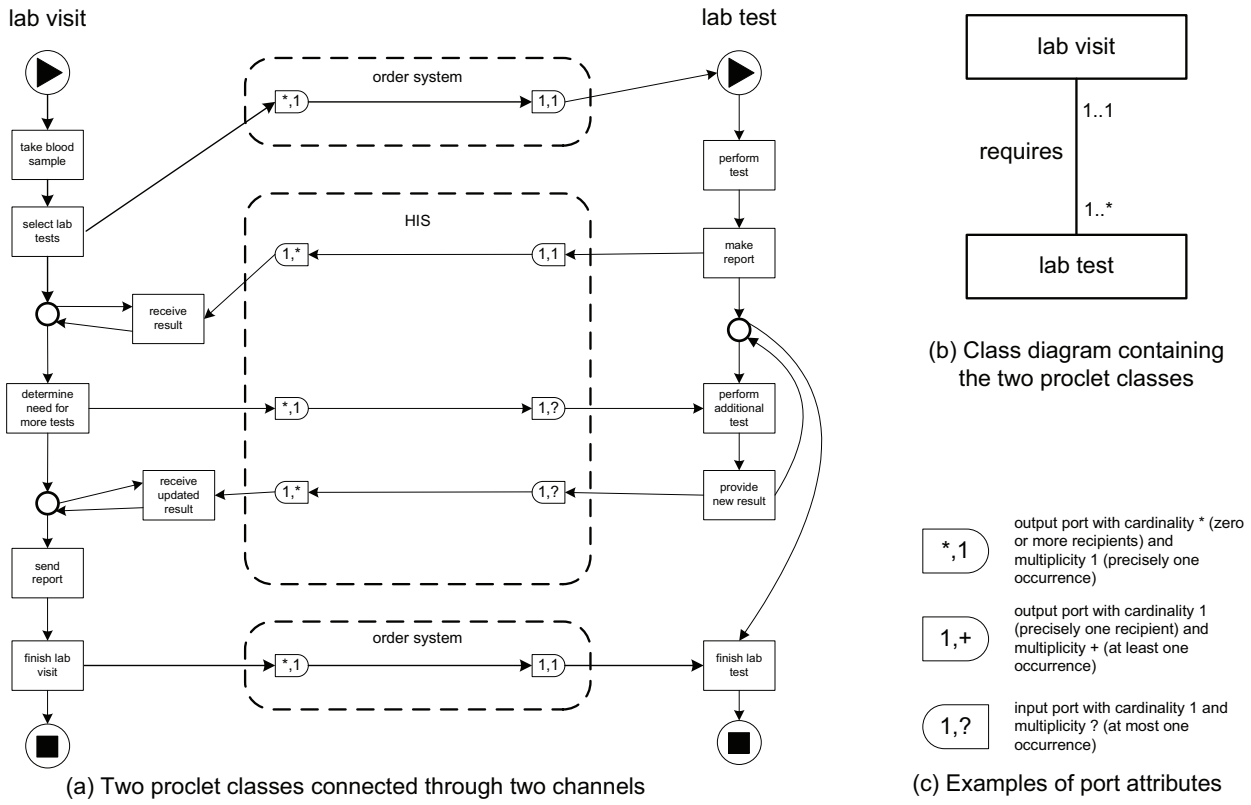

Figure 1: Example using two proclet classes: *lab visit* and *lab test*.

utilized. Therefore, we use the YAWL language rather than workflow nets, because YAWL is more expressive and supported by an extensive set of tools (editor, workflow engine, process mining, services, verification, simulation, etc.). See `www.yawl-system.com` for more information on the language and supporting tools.

Figure 1(a) shows two proclet classes. Proclet class *lab visit* consists of seven tasks and five ports and describes the process of taking a blood sample, ordering lab tests, and consolidating the results into a report. Proclet class *lab test* has five tasks and five ports and describes the life-cycle of a particular test. Note that for one blood sample many lab tests may be initiated. Hence there is a one-to-many relationship between *lab visit* and *lab test* as shown by the relationship *requires* in the class diagram in Figure 1(b). The two proclet classes are connected through two channels (*order system* and *HIS*). The mapping of ports to channels is shown in Figure 1(a).

The control-flow in each proclet class is expressed in terms of the YAWL notation. First, an instance (i.e. proclet) of class *lab visit* is created. After creation a blood sample is taken and lab tests are ordered. The output port of *select lab tests* has cardinality $*$, indicating that the performative is sent to potentially multiple recipients (i.e., lab tests). We will use $*$ to denote an arbitrary number of recipients, $+$ to denote at least one recipient, 1 to denote precisely one recipient, and *?* to denote no or just a single recipient. The performative is passed on via the channel *order system* and instantiates the proclet class *lab test* potentially multiple times, i.e., one proclet is created for every lab test that needs to be executed. The multiplicity of the output port of *select lab tests* is denoted by the number 1. This means that during the lifetime of an instance of class *lab visit* exactly one performative is sent via this port. The input port of the input condition of the *lab test* proclet has cardinality 1 and multiplicity 1. In each created *lab test* proclet a test is performed and the report is sent back to the "parent" *lab visit* proclet via the channel *HIS*. Note that the input port of task *receive result* has cardinality 1 and multiplicity $*$, indicating that multiple results may be received. Each performative received is stored in a knowledge base.

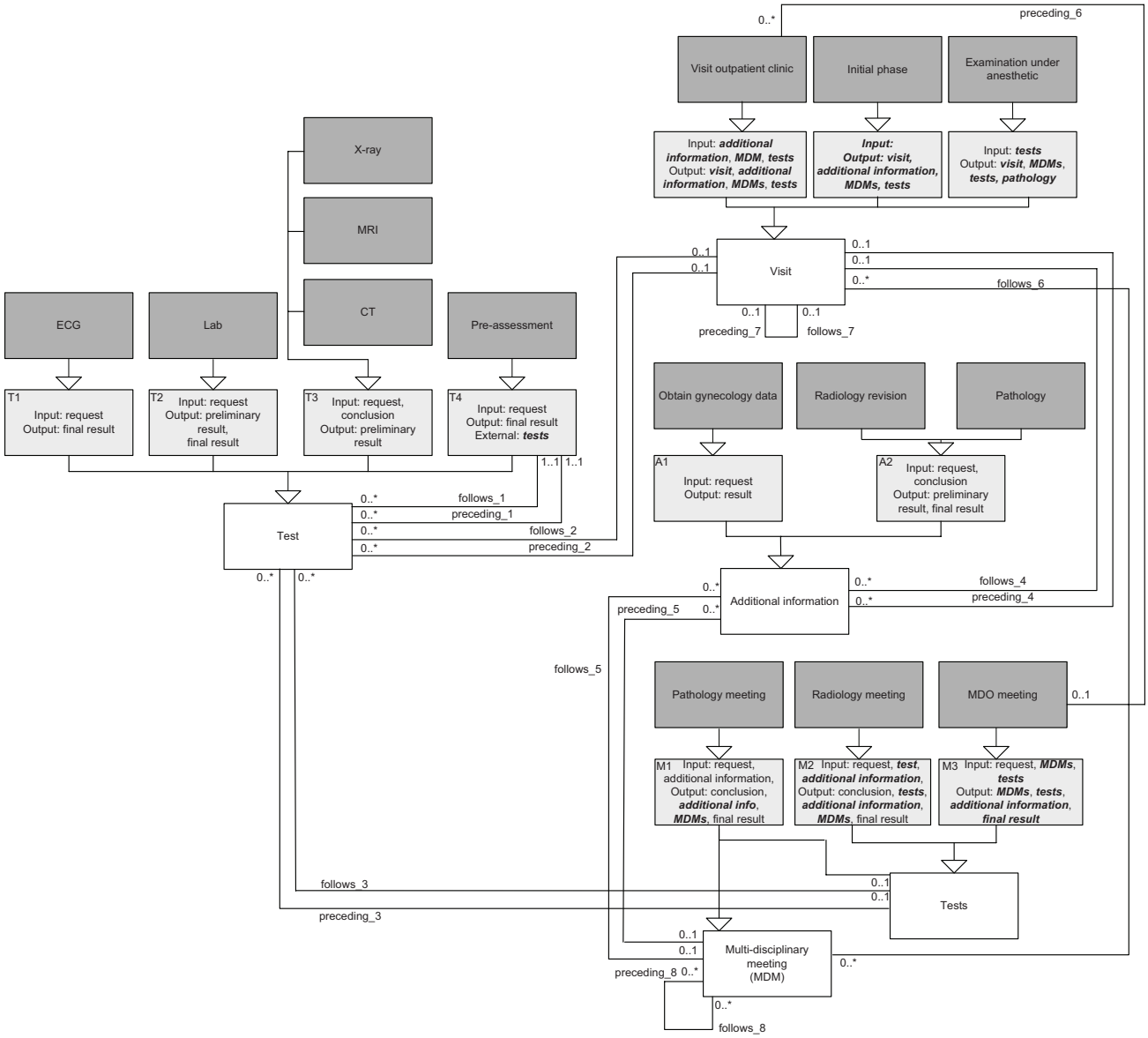

Figure 2: Class diagram outlining the concepts that exist within the healthcare process and their relationships.

The *lab visit* proclet continuously inspects this knowledge base and may decide to start analyzing the results to see if more tests are needed. If so, these are ordered in one go by the task *determine need for more tests*. Note that the cardinality of the output port of this task is $*$, i.e., in one step all relevant *lab test* proclets are triggered in order to perform any additional tests. After this the new results are sent from the various *lab test* proclets to the "parent" *lab visit* proclet. Finally, the task *finish lab visit* triggers the completion of all child *lab test* proclets that may have been initiated.

The example in Figure 1 is rather simplistic and hides many details, but at the same time it compactly illustrates the main features of proclets. For more details on the formalism we refer the reader to [1, 2]. Note that in Figure 1 interaction is modeled explicitly and there is no need to artificially flatten the process into a monolithic workflow, instead, the different levels of granularity are preserved. For more complex situations involving not only one-to-many relationships (as in Figure 1(b)) but also many-to-many relationships, it is still possible to model the overall process as a collection of intertwined loosely-coupled object life-cycles whilst it

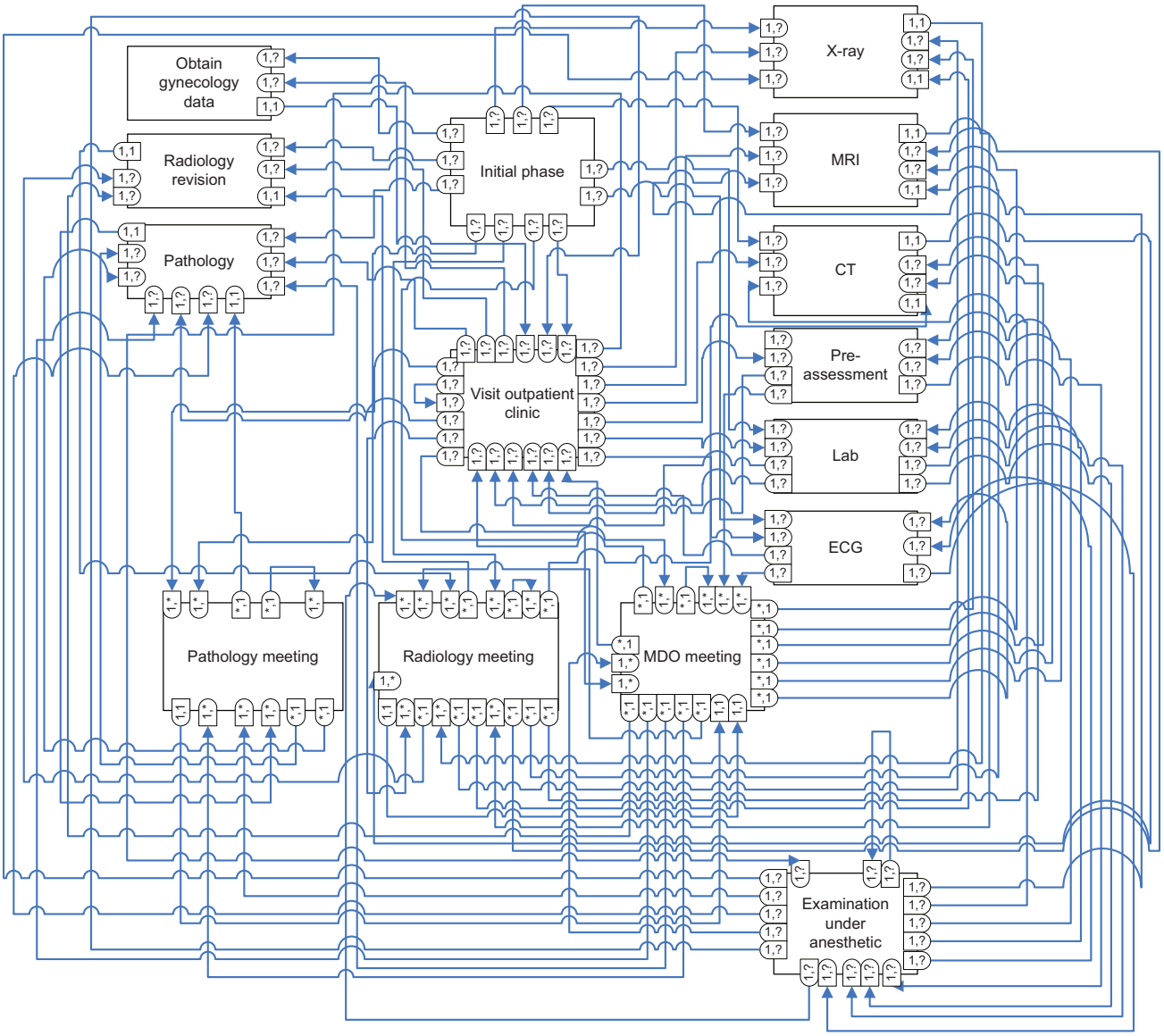

Figure 3: The proclets that are defined for the healthcare process and all the possible interactions between them.

would be virtually impossible to straightjacket the desired behavior into a monolithic workflow process.

Note that there is a strong correspondence between proclet classes and classes in a class diagram carrying the same name. A class in a class diagram outlines the data a proclet class carries with it and its relationship with other proclets. Via Object Constraint Language (OCL) expressions it is possible to access data of different proclets.

4 Application: Gynecological Oncology Workflow at the AMC

We have used proclets to model the gynecological oncology workflow at the Academic Medical Center (AMC) in Amsterdam. The AMC is the most prominent medical research center in the Netherlands and one of the largest hospitals in the country.

Given the complexity of the process and space limitations, we focus only on the main results. In total, 15

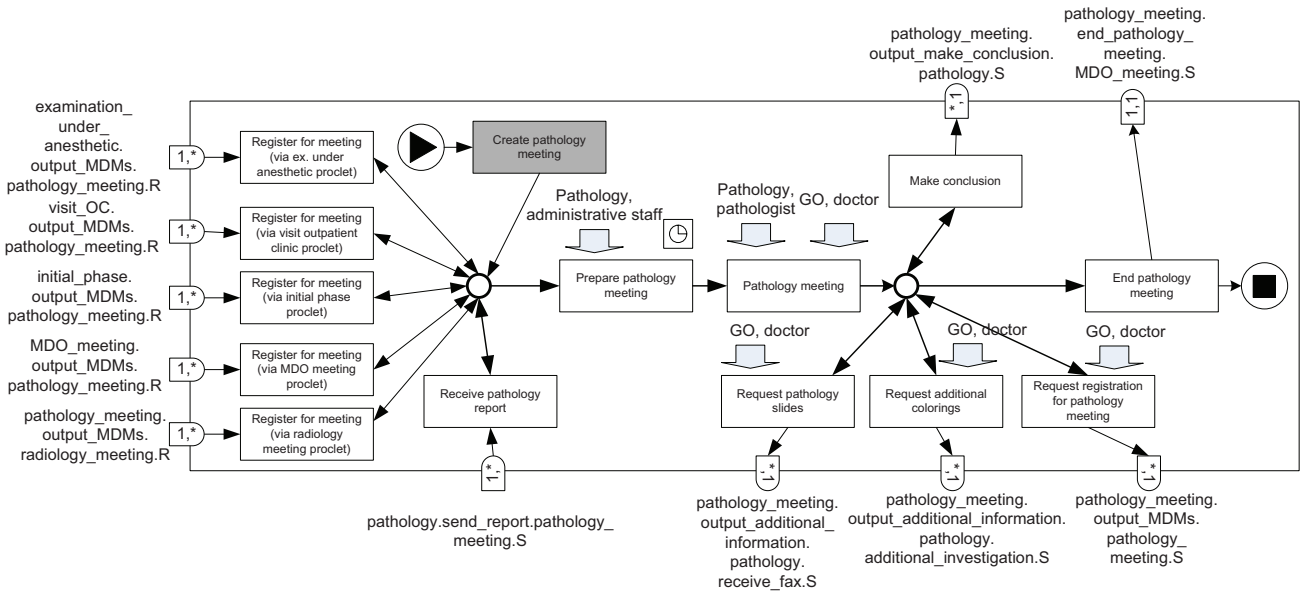

Figure 4: The *Pathology meeting* proclet.

proclet classes have been identified for the gynecological oncology workflow. Figure 2 shows a class diagram illustrating the relationships between the proclet classes. The dark rectangles correspond to concrete proclet classes. The inheritance relationships show which proclet classes have common features, i.e., the gray and white rectangles can be seen as abstract classes used to group and structure proclets. Moreover, as in Figure 1(b) the relationships between the various classes are depicted.

The 15 proclet classes identified in Figure 2 are connected to other proclet classes via the port and channel concepts. Figure 3 shows a high-level view of the interconnection structure. This diagram shows the complexity of the process. Given the different levels of granularity it is difficult (if not practically intractable) to flatten this structure into a monolithic workflow model.

Each of the rectangles in Figure 3 represents a proclet class and its ports. Figure 4 shows one example. Here the control-flow and the names of the ports and their cardinalities and multiplicities are shown. The proclet class models the weekly meeting in which the gynecological oncology doctors and a pathologist discuss the tissues examined by the pathologist that require further consideration. During this meeting, the tissues of multiple patients are discussed. For each weekly meeting, a separate proclet is created (*create pathology meeting*). In order to discuss a tissue of a patient, it first needs to be registered (*register for pathology meeting*). This can be done at different points in the process. However, as is indicated by the cardinality 1 and multiplicity $*$ of the associated ports, multiple patients can be registered using the same port. Note that after the weekly meeting (*Pathology meeting*), pathology examinations can be triggered for multiple patients. For example, as is indicated by the cardinality $*$ and multiplicity 1 of the associated port of task *Request additional colorings*, multiple tissues may be reinvestigated by a pathologist.

In this paper, it is impossible to give a more comprehensive description of the process and its 15 proclet classes. Instead, we refer the reader to [6] for an extensive description of the model.

5 Conclusion: Divide, Interact, and Conquer

In this paper we have advocated the use of proclets to overcome the problems related to monolithic workflows. Proclets are particularly suited to environments where processes are fragmented, interaction is important, and

tasks are done at different levels of granularity, e.g., healthcare processes where a visit to a doctor can trigger a wide range of tests and experiments. The next challenge is to provide advanced tool support for the design, analysis, and enactment of proclets. For example, proclets-based verification and process discovery pose interesting and challenging research questions.

References

[1] 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.

[2] 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.

[3] W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business Process Mining: An Industrial Application. *Information Systems*, 32(5):713–732, 2007.

[4] 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.

[5] R. Hull and J. Su. Research Challenges in Data-centric Workflow. Arlington, Virginia, 2009 (to appear).

[6] R.S. Mans, N.C. Russell, W.M.P. van der Aalst, A.J. Moleman, and P.J.M. Bakker. Proclets in Healthcare. BPM Center Report BPM-09-05, BPMcenter.org, 2009.