Lightweight interacting patient treatment processes

Lightweight Interacting Patient Treatment Processes

Ronny Mans (*) received his M.Sc. degree in Computer Science from Eindhoven University of Technology (TU/e) in 2006. At present he is a postdoctoral researcher at the Department of Industrial Engineering and Innovation Sciences at the same university. In his PhD studies he focused on the application of workflow management technology in the healthcare domain. His research interests include workflow management, process mining, and discrete event simulation.

Ronny Mans
Department of Industrial Engineering & Innovation Sciences, Paviljoen K03
Eindhoven University of Technology
Den Dolech 2
P.O. Box 513, NL-5600 MB, Eindhoven
the Netherlands
tel: +31 40 2473205
r.s.mans@tue.nl

Wil van der Aalst is a full professor of Information Systems at Eindhoven University of Technology. He is also an adjunct professor at Queensland University of Technology (QUT) and coinitiated the ProM and YAWL initiatives.

Wil van der Aalst
Department of Mathematics and Computer Science, HG 7.74
Eindhoven University of Technology
Den Dolech 2
P.O. Box 513, NL-5600 MB, Eindhoven
the Netherlands
tel: +31 40 2474295
w.m.p.v.d.aalst@tue.nl

Nick Russell has over 20 years experience in IT, both in Australia and the Netherlands, in a variety of senior management, technical, and research roles. He is currently Chief Technology Officer for a major Australian tools distributor. Prior to this, he was a postdoctoral researcher at Eindhoven University of Technology. He completed his masters and doctoral degrees at Queensland University of Technology. Over the past 7 years, he has been the driving force for the extension of the workflow patterns to the data, resource, and exception handling perspectives and the development of the newYAWL business process modeling reference language.

Nick Russell
Carba-Tec Pty Ltd
tel: +
nrussell@carbatec.com.au

Piet Bakker is a full professor of Process Innovation in Healthcare at the Faculty of Medicine at the University of Amsterdam (AMC-UvA) where he is also director of the Quality and Process Innovation Department. His research interests include patient logistics, patient care, evidence-based practice and patient security.

Piet Bakker
Department of Quality and Process Innovation
Academic Medical Center, University of Amsterdam
P.O. Box 22700, NL-1100 DE, Amsterdam
The Netherlands
tel: +31 20 5666294
p.j.bakker@amc.uva.nl

Arnold Moleman is working as lead architect at the Academic Medical Center (AMC), a university hospital in Amsterdam. His main field of interest is the application architecture for patient care applications, with a personal interest in patient care logistics.

Arnold Moleman
Department of ADICT
Academic Medical Center, University of Amsterdam
P.O. Box 22700, NL-1100 DE, Amsterdam
The Netherlands
tel: +31 20 5663315
a.j.moleman@amc.uva.nl

Lightweight Interacting Patient Treatment Processes

Ronny Mans, Eindhoven University of Technology, The Netherlands
Wil van der Aalst, Eindhoven University of Technology, The Netherlands
Nick Russell, Carba-Tec Pty Ltd, Australia
Piet Bakker, Academic Medical Center, The Netherlands
Arnold Moleman, Academic Medical Center, The Netherlands

ABSTRACT

Processes concerning the diagnosis and treatment of patients cannot be straightjacketed into traditional production-like workflows. They can be best characterized as weakly-connected interacting light-weight workflows where tasks reside at different levels of granularity. Moreover, for each individual patient a doctor proceeds in a step-by-step way deciding about the next steps to be taken. Classical workflow notations fall short in supporting these patient processes as they have been designed to support monolithic processes. Classical notations (WF-nets, BPMN, EPCs, etc.) assume that a workflow process can be modeled by specifying the life-cycle of a single case in isolation. To address these problems, we present an extension of the *Proclets framework* which allows for dividing complex entangled processes into simple autonomous fragments. Additionally, increased emphasis is placed on interaction related aspects such that fragment instances for individual patients can cooperate in any desired way. Finally, we describe an implementation of the Proclets framework. Proclets have been added to the open-source Workflow Management System YAWL to better support inter-workflow support functionalities.

Keywords: Workflow Management, Workflow Interactions, Lightweight Workflow Processes

INTRODUCTION

In healthcare organizations, such as hospitals, many complex processes are performed which are lengthy in duration. These are concerned with the diagnosis and treatment of patients. Workflow Management Systems (WfMSs) present an attractive vehicle in order to provide support and monitoring for patient processes. Based on process definitions, WfMSs are able to manage the flow of work in these processes such that individual work items are done at the right time by the proper person (van der Aalst & van Hee, 2002; Weske, 2007).

However, a number of difficulties commonly arise when hospitals attempt to automate patient processes. This is because the entire process for an individual patient typically consists of a number of workflow fragments that need to cope with different levels of granularity and that run at different speeds. Moreover, the doctor proceeds in a step-by-step way when deciding about the steps to be taken next. This can be illustrated by the example that is depicted in Figure 1.

InsertFout! Verwijzingsbron niet gevonden. Figure 1 here. (Caption: Interacting workflow fragments. The arcs show the interactions that need to take place between fragment instances.)

Figure 1 shows two possible instances of a patient process. For patient

‘Sue’ the process is as follows. First, she visits the outpatient clinic (the process instance named ‘visit:Sue 25/01’). Here, a doctor decides during the ‘decide’ task about the next step(s) that need to be taken. As indicated by the outgoing arcs, a second visit is necessary (‘visit:Sue 10/02’), a lab test needs to be taken (‘lab:Sue 25/01’), and Sue's case needs to be discussed during a multidisciplinary meeting (MDM) (‘MDM:05/02’). For the lab test the resulting report is needed as input for the second visit. This is indicated by the arc leading from the ‘send report’ task to the ‘receive’ task of the second visit. At the multidisciplinary meeting, multiple patients are discussed individually. For this meeting, patients need to be registered via the ‘registration’ task. Finally, reports are sent out (task ‘send reports’). Note that Sue is registered for the meeting and that the resulting report is necessary as input for the second visit.

For patient ‘Anne’ a similar process is followed. However, note that Anne and Sue are discussed at the same multidisciplinary meeting. So, process instance ‘MDM 05/02’ operates at a different level of granularity (a group of patients rather than a single patient).

Clearly, the entire patient process should be seen as a *cloud of standardized workflow fragments* for which the ultimate *selection of these fragments and the interactions between them is patient specific*. Current workflow languages require that the complete workflow is described as one monolithic overarching workflow (van der Aalst, Mans & Russell, 2009). Consequently, it is hard to describe such a cloud of loosely coupled workflow fragments and all the possible interactions between them.

For delivering the desired support, the Proclets framework (van der Aalst, Barthelmes, Ellis & Wainer, 2000, 2001) provides an interesting means in order to model and execute patient processes. 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. Moreover, fragments may reside at different levels of granularity.

The application of the original Proclets framework to care processes in the Academic Medical Center (AMC) hospital revealed several limitations. Based on a careful analysis of the original Proclets framework, we developed important extensions. These are presented in this paper. Via ‘interaction points’, ‘internal interactions’, and ‘external interactions’, at design time, possible interactions between Proclet classes can be modeled without the need to define complex pre- and postconditions. At run-time they allow users to select interactions between existing and future Proclet instances. As a result, users can define the next steps that need to be taken in the treatment process (e.g. a lab test). Although the framework has been extended based on experiences from the healthcare domain, its application is not limited to the healthcare domain only.

Proclets have formal semantics and therefore can be enacted using workflow technology. An additional contribution of this paper is that we show how WfMSs can be fully extended with facilities for inter-workflow support. As part of this, we elaborate on a concrete system which has been realized by significant extensions to the open-source YAWL WfMS (van der Aalst & ter Hofstede, 2005; ter Hofstede & van der Aalst & Adams & Russell; Russell & ter Hofstede, 2009).

The remainder of this paper is organized as follows. First, the Proclet framework together with our contributions will be discussed. Second, the design and

implementation of a WfMS extended with inter-workflow support is presented. Third, related work is discussed. Afterwards, an evaluation of our approach is provided together with an outlook. Finally, our conclusions are given.

PROCLETS FRAMEWORK

In this section, the Proclets framework will be discussed. However, due to space limitations only the most important aspects can be discussed. For full details we refer to (Mans, Russell, van der Aalst, Moleman & Bakker, 2010).

Proclets provide a framework for modeling and executing workflows (van der Aalst, Barthelmess, Ellis, & Wainer, 2000; van der Aalst, Barthelmess, Ellis, & Wainer, 2001). A *Proclet* can be seen as a lightweight workflow process able to interact with other Proclets that may reside at different levels of aggregation. A *Proclet class* specifies which tasks need to be executed and in which order, i.e, the Proclet class defines the process followed by individual Proclets. One instance is called a *Proclet instance*. For the definition of a Proclet class, a selection can be made between multiple graphical languages (e.g. BPMN (White 2004), EPCs (Aalst, 1999)). In this paper, we use a graphical language based on the YAWL language (van der Aalst & ter Hofstede, 2005). Note that Figure 2 provides an overview of all the constructs that are used in the YAWL language.

Insert Figure 2 here. (Caption: YAWL lexicon.)

Proclet instances interact via channels. A *channel* can be used to send a performative to one Proclet instance or a group of Proclet instances (i.e. multicast). A *performative* is a specific kind of message with several attributes. The 'sender' attribute contains the identifier of the Proclet instance creating the performative and the 'set of receivers' attribute contains the identifiers of the Proclet instances receiving the performative.

Proclet classes are connected to channels via ports. Each *port* has two attributes: (a) the *cardinality* specifies the number of recipients of performatives exchanged via the port, and (b) the *multiplicity* specifies the number of performatives exchanged via the port during the lifetime of any instance of the class. Moreover, a port is either an incoming or an outgoing port. Each outgoing port is connected with exactly one incoming port called an *external interaction*. Furthermore, every port is connected to one interaction point. An *interaction point* represents a specific point in a Proclet class at which interactions with other Proclet classes may take place.

Furthermore, for an interaction point having only incoming ports, it may be desired that the receipt of an individual performative is followed by the subsequent sending of a performative at a later point in the execution of that Proclet instance. For example, for a lab test that is requested, its subsequent report needs to be used as input for another visit. Therefore, such an interaction point may be connected with an interaction point which has only outgoing ports and is called an *internal interaction*.

Insert Figure 3 here. (Caption**:** Proclet classes for the scenario illustrated in Figure 1.)

Figure **Fout! Verwijzingsbron niet gevonden.**3 shows three Proclet classes which are modeled based on the scenario illustrated in Figure 1**Fout! Verwijzingsbron niet gevonden.**. Proclet class 'visit' has four tasks, three interaction points, six ports, and describes a visit to the outpatient clinic.

Proclet classes ‘lab’ and ‘MDM’ describe respectively a lab test that is performed and a multidisciplinary meeting to discuss multiple patients. As can be seen in the class diagram, multiple lab tests and multiple registrations for a multidisciplinary meeting can be triggered starting from a visit of a patient. This also becomes clear from the ports that are connected to the interaction point of the ‘decide’ task in the ‘visit’ Proclet class. For example, one port is leading to the input condition of the ‘lab’ Proclet class. The cardinality ‘*’ and multiplicity ‘?’ show that is optional to create multiple instances of this class. Finally, the cardinality ‘1’ and multiplicity ‘?’ of the outgoing port of the ‘send report’ task (‘lab’ Proclet class) indicate that it is optional to send the result of a lab test to exactly one ‘visit’ Proclet instance. Note that for the ‘lab’ Proclet class an internal interaction is defined from the interaction point that is connected to the input condition to the interaction point connected to the ‘send report’ task. This allows that an instance of this class is created and that the result of the lab test is forwarded to a certain ‘visit’ Proclet instance. Finally, note that the ‘MDM’ Proclet instance operates at another aggregation level (a group of patients) than the other Proclet instances (a single patient).

However, in the scenario of Figure 1 it is indicated that for both ‘Sue’ and ‘Anne’ the result of the lab test is used as input for the second visit, i.e. for both a performative needs to be sent from the corresponding ‘lab’ Proclet instance to the corresponding Proclet instance for the second visit. This decision is made as part of executing the ‘decide’ task of the first visit. Note that at this point in time some Proclet instances do not exist yet (e.g. the instances for the lab and the second visit). As a consequence, it is important to record to which Proclet instances a performative was sent. Moreover, for the interactions that are taking place between (future) Proclet instances, it is important whether they have already taken place. For example, to ensure that the ‘receive’ task of the second visit may be executed for ‘Sue’, it is important to know whether the performative from the ‘send report’ task has already been received. Obviously, a global overview is required about the performatives that need to be sent around. Below, we introduce two different concepts in order to achieve this.

Entities and Interaction Graphs

The first concept is called an *entity*, which is an object that exists next to existing and future Proclet instances. Examples of an entity are a patient or a lab test. So, both ‘Sue’ and ‘Anne’ can be an entity. We use an interaction graph to ensure that for an entity multiple Proclet instances are performed, that interactions between them take place in the desired order, and that the state of these interactions are saved. Such an *interaction graph* belongs to a specific entity and consists of interaction nodes and interaction arcs. An *interaction node* refers to an interaction point of a Proclet instance for which one or more internal or external interactions will take place, i.e. an interaction node is a triple consisting of the identifier of the Proclet class, the identifier of the Proclet instance, and the identifier of the interaction point itself. An *interaction arc* refers to an interaction, either internal or external, that needs to occur between two interaction nodes. In that way, the direction of the arc in the graph is the same as the direction of the arc for the associated interaction. Furthermore, in order to save the state of interactions, every arc has an interaction identifier and an interaction state. The *interaction state* stores the specific state of the interaction. The *interaction identifier* is a unique identifier of which the first value is the entity itself and the second is a unique identifier for the interaction. When sending

a performative for the interaction, the interaction identifier is added to the 'set of interaction identifiers' attribute. In this way, it will be known whether external interactions have occurred or not.

Insert Figure 4 here. (Caption: For both 'Sue' and 'Anne' it is shown how existing and future Proclet instances need to interact. Moreover, the interaction graphs that are created during the execution of the 'decide' task are shown.*)*

In Figure 4a and 4c, the interaction graphs are shown for entities 'Sue' and 'Anne'. These graphs have been created during the execution of the 'decide' task for both of them. The corresponding (future) instances for both are shown in Figure 4b together with the desired interactions. Tasks that have already been finished are marked with a check mark. For 'Sue', via dotted arcs, nodes of the interaction graph are linked with the corresponding interaction points in a Proclet instance. Similarly, arcs of the interaction graph are linked with the corresponding interactions. For example, after executing the 'decide' task for 'Sue', the arc from the '(visit,Sue 25/01,decide)' node to the '(MDM, 05/02,register)' node means that a performative needs to be sent in order to register 'Sue' for the multidisciplinary meeting. Subsequently, the arcs from the '(MDM, 05/02,register)' node to the '(MDM,05/02,send report)' node and from the '(MDM,05/02,send report)' node to the '(visit,T1,receive)' node mean that after the registration the 'send reports' task needs to be executed and that the outcome of this task, which is a report, is used as input for the 'decide' task of the second visit.

For 'Anne' the interaction graph is comparable. As both are discussed during the multidisciplinary meeting, similar interaction nodes and arcs for the 'MDM:05/02' Proclet instance appear in both graphs. As no performatives have been sent, all arcs referring to an external interaction have state 'unproduced'. This means that both the source and destination interaction node have not occurred yet (e.g. for 'Sue' the 'lab' Proclet instance has not been created yet).

In order to illustrate the different states an interaction arc can have and to see Proclets in action, several tasks will be executed for both 'Sue' and 'Anne'. Therefore, for Figure 4 we assume that for both 'Sue' and 'Anne' the 'decide' task of the first visit is executed and that performatives are sent for it. This situation is depicted in Figure 5.

Insert Figure 5 here. (Caption: Performatives are sent for both 'Sue' and 'Anne' as a result of executing their 'decide' tasks.)

As can be seen all outgoing arcs of the '(visit,Sue 25/01,decide)' node and the '(visit,Anne 26/01,decide)' node have now state 'sent'. This means that a performative has been produced for the interaction represented by the arc. In particular, the interaction identifier of the arc is contained in the 'interaction identifiers' attribute of the performative that has been sent. For example, we see the performative that is sent for 'Sue' to the 'register' task of the multidisciplinary meeting. As interaction identifier '(Sue,4)' is added.

Insert Figure 6 here. (Caption: The 'register' task of the multidisciplinary meeting is executed.)

As a next step we assume that the 'register' task of the multidisciplinary meeting is executed. The new situation is depicted in Figure 6. As performatives have been sent containing the corresponding interaction identifiers, instances of the 'lab'

and the ‘visit’ Proclet class have been created both for ‘Sue’ and ‘Anne’. Moreover, the ‘register’ task of the multidisciplinary meeting is allowed to be executed as all incoming arcs of the ‘(MDM,05/02,register)’ node had state ‘sent’ in the graphs of both ‘Sue’ and ‘Anne’ (see Figure 5). For all these arcs, the state has been changed to ‘consumed’. Here it can be imagined that a performative has been ‘consumed’ in order to create either an instance of a Proclet class or to perform a task. Note that interaction nodes, which relate to an instance that has been created, have been updated with the identifier of that instance (e.g. the ‘lab’ instance for ‘Sue’ has now ‘Sue 25/01’ as instance identifier instead of ‘T3’).

Moreover, several arcs have state ‘executed source’ as the source interaction node of an internal interaction has been performed. For example, for the ‘(MDM,05/02,register)’ node the ‘register’ task has been performed. Here, it is important to see that this action has impact on the graphs of both ‘Sue’ and ‘Anne’. Actually, arcs between interaction nodes that can be found in multiple graphs are updated simultaneously and always have the same state.

Insert Figure 7 here. (Caption: The ‘send reports’ task of the multidisciplinary meeting is executed.)

Finally, as a subsequent step we assume that the ‘send reports’ task is performed (Figure 7). As can be seen in the two interaction graphs, the arc from the ‘(MDM,05/02,register)’ node to the ‘(MDM,05/02,send report)’ node has state ‘executed both’. This represents that both the source and destination interaction node of an internal interaction have been performed.

For the interactions that are defined in an interaction graph several exceptions may occur in which they cannot take place anymore (e.g. a Proclet instance may be canceled). Also, interactions may occur in the context of loops. Furthermore, as part of the sending of performatives, data is exchanged between Proclet instances. Above mentioned aspects are discussed in detail in (Mans, Russell, van der Aalst, Moleman & Bakker, 2010).

Extending an Interaction Graph

So far, we only considered interaction graphs that were already defined. We briefly illustrate how an interaction graph is extended for an entity. The extension of an interaction graph is based on the current graph and on the interaction points that exist for Proclet classes and how they are connected.

Assume that for the scenario shown in Figure 1 we are currently in the process of performing the ‘decide’ task during the first visit of ‘Sue’ (instance ‘visit:Sue 25/01’). Moreover, for the ‘MDM’ Proclet class currently an instance exists with instance identifier ‘05/02’. However, for the ‘decide’ task it is allowed to extend interaction graphs of entities during its execution. As no interaction graph exists for ‘Sue’, it is created first. The result can be seen in Figure 8. The ‘(visit,Sue 25/01,decide)’ node refers to the ‘decide’ task that is executed.

Insert Figure 8 here. (Caption: Defining an interaction graph for ‘Sue’. Possible interactions for an interaction node are indicated by dotted arcs.)

In Figure 8a we see the Proclet classes that have been defined and how they are connected. Starting from the ‘(visit,Sue 25/01,decide)’ node, which has three

outgoing ports, several interactions may be nominated. That is:

- One outgoing port is connected with the *input condition of a Proclet class*. For this class multiple instances may be created.
- A similar remark can be made for the 'visit' Proclet class. So, also for the 'visit' Proclet class multiple instances may be created.
- One outgoing port is connected with a *task of a Proclet class*. In the figure this is the 'register' task of the 'MDM' Proclet class. As currently an instance of the 'MDM' Proclet class exists with instance identifier '05/02', it is possible to have an interaction with the 'register' task of that Proclet instance. Note that we abstract from the current state of the 'MDM:05/02' Proclet instance.

In Figure 8b, the interactions that may be nominated are indicated by dotted arcs. Here, it is decided to create one instance of the 'visit' Proclet class, representing the second visit, and to have an interaction for the 'register' task representing that 'Sue' is discussed at the multidisciplinary meeting.

The updated graph can be seen in Figure 8c. As a result of the nominated interactions, there are arcs leading from the '(visit,Sue 25/01,decide)' node to the '(visit,T1,create)' node and to the '(MDM,05/02,send reports)' node. Note that interaction nodes, which relate to a future Proclet instance, have currently a temporary identifier (e.g. 'T1' for the second visit).

Based on the interaction nodes for which currently a Proclet instance exists or for which an instance will exist in the future, it is allowed to define subsequent interactions. So, for the '(visit,Sue 25/01,decide)' node again interactions may be defined. However, for the '(visit,T1,create)' node no subsequent interactions may be defined as for the associated interaction point no outgoing external and internal interactions have been defined. For the associated interaction point of the '(MDM,05/02,register)' node an internal interaction is defined which has the 'send report' task as its destination (Figure 8a). So, as indicated by the dotted arc in Figure 8c, it is possible to have an internal interaction from the '(MDM,05/02,register)' node to the '(MDM,05/02,send report)' node.

In Figure 8d, the updated graph is shown. Here, the interaction from the (MDM,05/02,register)' node to the '(MDM,05/02,send report)' node has been nominated. As can be seen in Figure 8a, the 'send report' task has one outgoing port which is connected with a *task of a Proclet class*. This is the 'receive' task of the 'visit' Proclet class. As an instance of the 'visit' Proclet class exists with instance identifier 'Sue 25/01', it is possible to have an interaction with the 'register' task of that Proclet instance. Moreover, in the graph of 'Sue' there is a node for a 'visit' instance with temporary instance identifier 'T1'. As it will exist in the future, interactions may be defined for it. Consequently, an interaction with the 'receive' task of this future instance may be nominated (node '(visit,T1,receive)').

Above, we have given an illustration of the different ways an interaction graph can be extended. It allows for defining interactions with future and existing Proclet instances in many different ways.

DESIGN AND IMPLEMENTATION OF A WFMS AUGMENTED WITH INTER-WORKFLOW SUPPORT

In this section we discuss how existing WfMSs can be extended with facilities for inter-workflow support. In particular, we focus on a concrete WfMS (YAWL) that has been extended with inter-workflow support based on our extended Proclet framework.

Approach

In order to determine how inter-workflow support facilities can be added to existing WfMS, we have first developed a detailed conceptual model which allows for identifying and formalizing the behavior of such a system. The conceptual model was created using CPN Tools (cpntools.org). This allowed us to simulate the behavior of the implemented system at an early stage. This conceptual model served as specification for the subsequent implementation of the system. Moreover, the CPN model provides a test environment of the real system.

In Figure 9a, the main components of the implemented system are shown.

- The **Workflow Engine** is the 'core' of the system and takes care of the routing of cases. The engine is realized by using the workflow engine of the open source WfMS YAWL (van der Aalst & ter Hofstede, 2005). Moreover, the YAWL Process Definition Editor allows for defining the process definition of a Proclet class.
- For a task that becomes available for execution, the corresponding work item is communicated to users via the **Workflow Client Application**. This component is realized using the Resource Service of the YAWL WfMS.
- Finally, the **Inter-Workflow Service** adds the desired inter-workflow support. For this component, the **Interaction Service** is responsible for managing interactions between Proclet instances at runtime. The Interaction Service has been set-up as a YAWL Custom Service. For tasks for which interactions may be required, the execution of the associated work item is delegated to the service.

  The **Interaction Definition Editor** offers tools that allow for defining the remaining aspects of Proclet classes (e.g. interaction points, internal interactions, external interactions). By keeping these remaining aspects separate from process definitions that are used as part of a WfMS, we can add inter-workflow support to any WfMS. Additionally, tools are offered such that at instance level human actors can define necessary interactions between Proclet instances. This subcomponent has been implemented as a Java application.

Insert Figure 9 here. (Caption: Architecture of the realized system. Moreover, several screenshots of the realized system are shown.)

Modeling and Enactment support

Below we give a brief impression of the modeling and enactment support provided by the system. We start by showing how a Proclet class is modeled in our system. This is illustrated for the 'visit' Proclet class in Figure 9b. The corresponding process definition is modeled using the YAWL Process Definition Editor. Tasks for which interactions are required are indicated by a plug-in icon (e.g. the 'receive' task). The execution of the corresponding work item for them is delegated to the Inter-Workflow Service.

Next, the interaction points, ports, internal interactions, and external interactions are defined via the Graphical User Interface (GUI) of the Interaction Definition Editor. For the GUI shown at the bottom of Figure 9b, interaction points are visualized by a red dot together with its associated identifier. Ports are visualized by

a green dot together with its identifier and its cardinality and multiplicity. Via dotted arcs, interaction points and ports in the GUI and interaction points and ports of the 'visit' Proclet class are linked with each other.

In order to give an illustration of the enactment support provided by the system we give a screenshot of how an interaction graph is extended in Figure 9c. For 'Sue' an instance of the 'visit' Proclet class exists which has '71' as instance identifier and that an instance of the 'MDM' Proclet class exists which has '72' as instance identifier. Furthermore, the interaction graph of 'Sue' is currently extended as part of executing the 'decide' task of the 'visit' Proclet instance. Here, the interaction node of the 'decide' task ((visit,71,decide)) has been selected in order to define new interactions.

At the right side, for the selected node, the interactions that may be nominated are presented. In the 'Instantiate Proclet Instance' panel interactions may be selected which lead to the instantiation of a Proclet class. For each of them, it can be indicated how many instances need to be instantiated. In the Existing or Temporary Proclet Instance' panel, interactions with existing or future Proclet instances can be selected. In the 'Internal Interaction' panel, internal interactions can be selected. In Figure 9c it has been indicated that one instance of the 'visit' Proclet class will be created, no instances of the 'lab' Proclet will be created, and that an interaction with the 'register' task of the 'MDM' instance is required.

Afterwards, the graph can be further extended till all desired interactions are defined.

RELATED WORK

Contemporary workflow languages and systems provide limited support for the modeling and execution of loosely coupled interacting workflows. Instead, most systems and approaches force the designer to squeeze real-life processes into a single 'monolithic overarching workflow' which describes how an individual case is handled in isolation. This issue has been recognized in the literature (van der Aalst, Barthelmes, Ellis & Wainer, 2001; van der Aalst, Mans & Russell, 2009; Bhattacharya, Gerede, Hull, Liu & Su, 2007; Künzle & Reichert, 2009; Müller & Reichert, 2007,2008). As a consequence, current WfMSs do not provide an adequate means for inter-workflow coordination (van der Aalst, Mans & Russell, 2009; Heinlein 2002)

One of the earliest formalisms which acknowledged the above mentioned problems is the Proclets framework presented by van der Aalst et al. (2000, 2001). Using Proclets increased emphasis is placed on interaction related aspects of workflow. Moreover, different levels of granularity, one-to-many and many-to-many relationships that exist between entities in a workflow, and batch-oriented execution of tasks is supported.

In comparison to the Proclets framework there are a limited range of alternate approaches that deal with the same issues (Künzle & Reichert, 2009). Müller et al. (2007, 2008) have worked on the Corepro framework which allows automatic generation and coordination of individual processes based on their underlying data structure. For a specific component in the data structure, the corresponding structure of the process is described by the data used during the life cycle of the component. Relationships between the components in the data structure, which can capture one-to-many and many-to-many relationships, indicate process dependencies.

Browne et al. present a two-tier, goal-driven model for workflow processes in the healthcare domain (Browne, Schrefl, Warren, 2003, 2004, 2004). 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 tasks such that each of the (sub)processes is designed in a way that achieves one of the upper level goals. However, direct interactions between subprocesses are not possible.

Bhattacharya et al. take the so-called business artifacts of a process as a starting point (Bhattacharya, Caswell, Kumaran, Nigam, Wu, 2007; Bhattacharya et al., 2007, Nigam, Caswell, 2003). A business artifact is an identifiable, self-describing unit of information. Based on this an artifact-centric process model can be constructed which consists of the business artifacts itself, business tasks, and repositories. However, the content of one or more artifacts can only be changed by the execution of a task. Consequently, aggregation issues are only handled at the task level.

Batch-oriented tasks are tasks that are based on groupings of lower aggregation elements. The concept of a batch-oriented task has been discussed in (Barthelmess, Wainer, 1995; Sadiq, Orlowska, Sadiq, Schulz, 2005) in order to allow for a task that is executed for multiple instances at the same time.

Finally, Heinlein (2000, 2001, 2002) focuses on the synchronization of concurrent workflows. An interaction graph is proposed which specifies how multiple workflows need to be synchronized. However, all involved workflows operate at the same level of aggregation and no data is exchanged between workflows.

DISCUSSION

The extended Proclets framework provides new ways of dealing with the complexity of processes encountered in the healthcare domain. Nevertheless, there are some limitations that need to be addressed in the future.

First of all, the Proclets framework allows for modeling complex workflows and their interactions. The increased emphasis on the interaction side of workflows requires a different modeling approach, and, hence modelers trained in this new style of modeling. Here we would like to stress that the complexity shown in the diagrams is real and simply ignored in classical notations. Using conventional approaches (e.g., BPMN models with swim-lanes) this is abstracted away. However, the behavior is essential for the process as a whole and needs to be implemented when realizing the information system.

A user has complete freedom in nominating interactions that need to take place between Proclet instances. A direct consequence of this freedom is that many errors can be introduced. This requires that advanced analysis algorithms are needed at both design-time and run-time. At design-time one would like to identify models that have obvious structural problems. At run-time one would like to forecast whether interactions can take place or not. Such diagnostics allow for immediate intervention such that undesired situations are prevented. Note that such analysis is far from trivial. Data plays a prominent role in our framework and cannot be abstracted away easily. Moreover, there is the problem of considering multiple instances at the same time. There can be many-to-many relationships between Proclet classes. This is known to be a difficult verification problem. In fact, without constraints the problem can be shown to be undecidable.

Finally, although the Proclet framework has been extended based on our experiences from the healthcare domain, no evaluation has taken place yet. Therefore,

we propose a study in which a group of medical professionals use our system following a realistic scenario. Based on their evaluation, the extended framework and the developed system can be improved. Afterwards, we plan to perform a pilot study in which one of AMC's healthcare processes is supported using our system.

CONCLUSIONS

Patient processes can best be seen as a cloud of standardized workflow fragments for which the ultimate selection of these fragments and the interactions between them is patient specific. Moreover, these fragments may cope with different levels of granularity. Current workflow languages require that a patient process is described as one monolithic overarching workflow.

In order to optimally support patient processes we have discussed an extension of the original Proclets framework. In particular, using the concepts of an entity and an interaction point, at design time, possible interactions between Proclet classes can be modeled without the need to define complex pre- and postconditions. Furthermore, at run-time, users can select interactions between existing and future Proclet instances. Consequently, it is possible to easily define the next steps within a patient treatment process. Finally, in order to demonstrate that Proclets can be enacted, we have briefly elaborated on a design and an implementation of a WfMS augmented with inter-workflow support.

REFERENCES

Aalst, W.M.P. van der (1999). Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, *41*(10), 639-650.

Aalst, W.M.P. van der, & Barthelmess, P. & Ellis, C.A. & Wainer, J. (2000). Workflow modeling using proclets. In O. Etzion & P. Scheuermann (Ed.), *Lecture Notes in Computer Science: 1901. 7th International Conference on Cooperative Information Systems* (pp. 198-209). Berlin: Springer-Verlag.

Aalst, W.M.P. van der, & Barthelmess, P. & Ellis, C.A. & Wainer, J. (2001). Proclets: A Framework for Lightweight Interacting Workflow Processes. *International Journal of Cooperative Information Systems*, *10*(4), 443-482.

Aalst, W.M.P. van der, & Hee, K.M. van. (2002). Workflow Management: Models, Methods, and Systems. Cambridge, MA: MIT Press.

Aalst, W.M.P. van der, & Hofstede, A.H.M. ter. (2005). YAWL: Yet Another Workflow Language. *Information Systems*, *30*(4), 245-275.

Aalst, W.M.P. van der, & Mans, R.S. & Russell, N.C. (2009).Workflow Support using Proclets: Divide, Interact, and Conquer. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, *32*(3), 16-22.

Barthelmess, P., & Wainer, J. (1995). Workflow systems: a few definitions and a few suggestions. In N. Comstock and C.A. Ellis (Ed.), *Proceedings of the Conference on Organizational Computing Systems - COOCS'95* (pp. 138-147). New York:ACM Press.

Bhattacharya, K. & Caswell, N.S. & Kumaran, S. & Nigam, A. (2007). Artifact-Centered Operational Modeling: Lessons from Customer Engagements. *IBM Systems Journal*, *46*(4), 703-721.

Bhattacharya, K., & Gerede, C. & Hull, R. & Liu, R. & Su, J. (2007). Towards formal analysis of artifact-centric business process models. In G. Alonso & P. Dadam & M. Rosemann (Ed.), *Lecture Notes in Computer Science: 4714. International Conference on Business Process Management (BPM 2007)* (pp. 288-304). Berlin: Springer-Verlag.

Browne, E.D., & Schrefl, M. & Warren, J.R. (2003). A two tier, goal-driven workflow model for the healthcare domain. *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS 2003)* (pp. 32-39).

Browne, E.D., & Schrefl, M. & Warren, J.R. (2004). Activity crediting in distributed workflow environments. *Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS 2004)*.

Browne, E.D., & Schrefl, M. & Warren, J.R. (2004). Goal-focused self-modifying workflow in the healthcare domain. *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS-37 2004)* (pp. 197-210). Los Alamitos: IEEE Computer Society Press.

Heinlein, C. (2001). Workflow and process synchronization with interaction expressions and graphs. *Proceedings of the 17th International Conference on Data Engineering* (pp. 243-252). Los Alamitos: IEEE Computer Society.

Heinlein, C. (2002). Synchronization of concurrent workflows using interaction expressions and coordination protocols. In R. Meersman and Z. Tari (Ed.), *Lecture Notes in Computer Science: 2519. On the Move to Meaningful Internet Systems 2002* (pp. 54-71). Berlin: Springer-Verlag.

Heinlein, C. (2000). Workflow and process synchronization with interaction expressions and graphs. Doctoral dissertation, University of Ulm, Ulm, Germany.

Hofstede, A.H.M. ter, & Aalst, W.M.P. van der, & Adams, M.J., & Russell, N.C. (2010) Modern Business Process Automation: YAWL and its Support Environment. Berlin: Springer-Verlag.

Künzle, V., & Reichert, M. (2009). Towards object-aware process management systems: issues, challenges, benefits. In G T. Halpin and J. Krogstie and S. Nurcan and E. Proper and R. Schmidt and P. Soffer and R. Ukor (Ed.), *Lecture Notes in Business Information Processing: 29. Proc. 10th Int'l Workshop on Business Process Modeling, Development, and Support (BPMDS'09)* (pp. 197-210). Berlin: Springer-Verlag.

Mans, R.S. & Russell, N.C. & Aalst, W.M.P. van der, & Moleman, A.J. & Bakker, P.J.M. (2010). Inter-workflow support (Rep. BPM-10-10). BPM Center, BPMcenter.org.

Müller, D., & Reichert, M. & Herbst, J. (2008). A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In Z. Bellahsène and M. Léonardi (Ed.), *Lecture Notes in Computer Science: 5074. Information Systems Engineering* (pp. 48-63). Berlin: Springer-Verlag.

Müller, D., & Reichert, M. & Herbst, J. (2007). Data-driven modeling and coordination of large process structures. In R. Meersman and Z. Tari (Ed.), *Lecture Notes in Computer Science: 4803. On the Move to Meaningful Internet Systems 2007* (pp. 131-149). Berlin: Springer-Verlag.

Nigam, A. & Caswell, N.S. (2003). Business Artifacts: an Approach to Operational Specification. *IBM Systems Journal*, *42*(3), 428-445.

Russel, N.C., & ter Hofstede, A.H.M. (2009). newYAWL: Towards Workflow 2.0. In. K. Jensen and W.M.P. van der Aalst (Ed.), *Lecture Notes in Computer Science: 5460. Transactions on Petri Nets and Other Models of Concurrency*, 79-97.

Sadiq, S., & Orlowska, M. & Sadiq, W. & Schulz, K. (2005). When workflows will not deliver: the case of contradicting work practice. In W. Abramowicz (Ed.), *Proceedings BIS'05* (pp. 69-84). Poznan: Wydawnictwo Akademii Ekonomicznej w Poznaniu.

Weske, M. (2007). Business Process Management: Concepts, Languages, Architectures. Berlin: Springer-Verlag.
S.A. White (2004). Business process modeling notation (BPML), Version 1.0. BPMI.org.

Figures

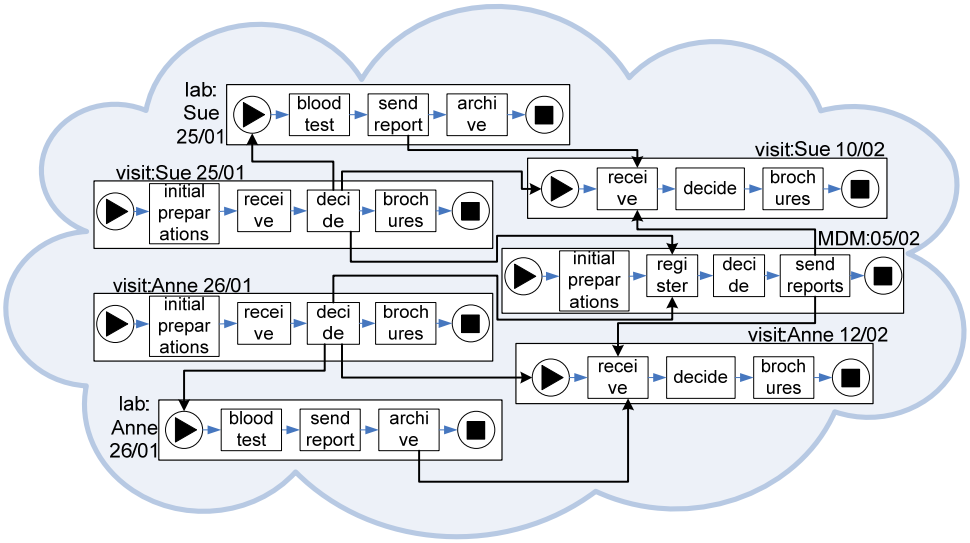


Figure 1: Interacting workflow fragments. The arcs show the interactions that need to take place between fragment instances.

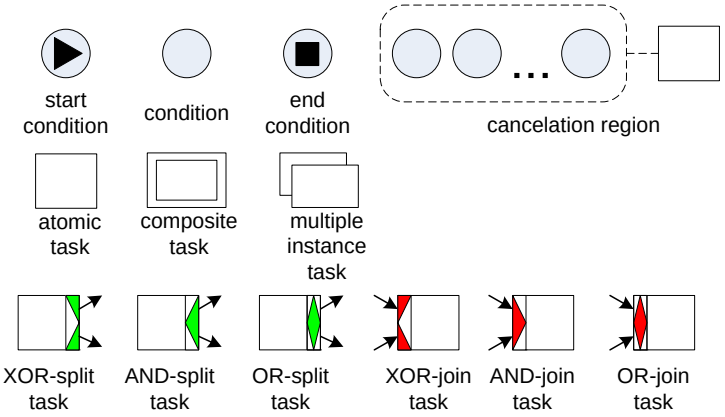
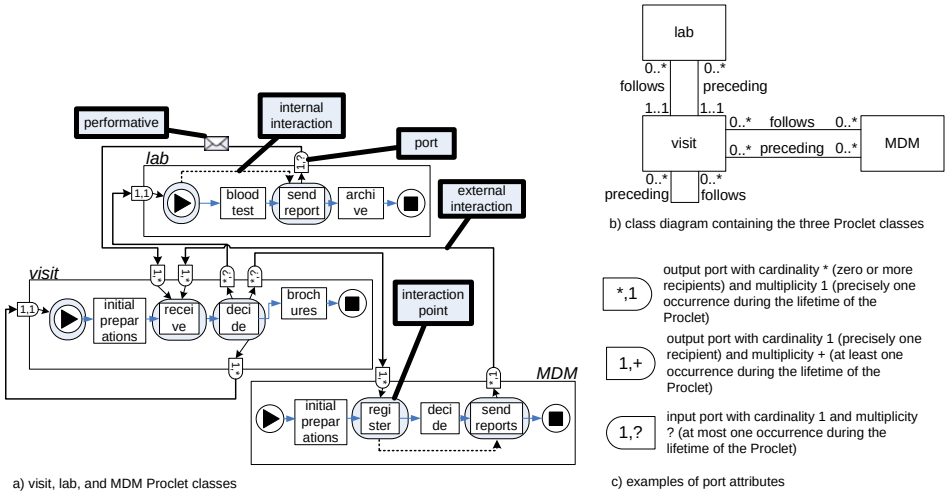


Figure 2: YAWL lexicon.

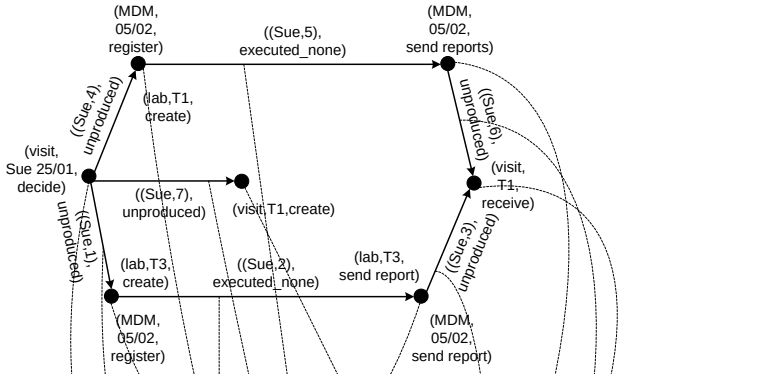


a) visit, lab, and MDM Proclet classes

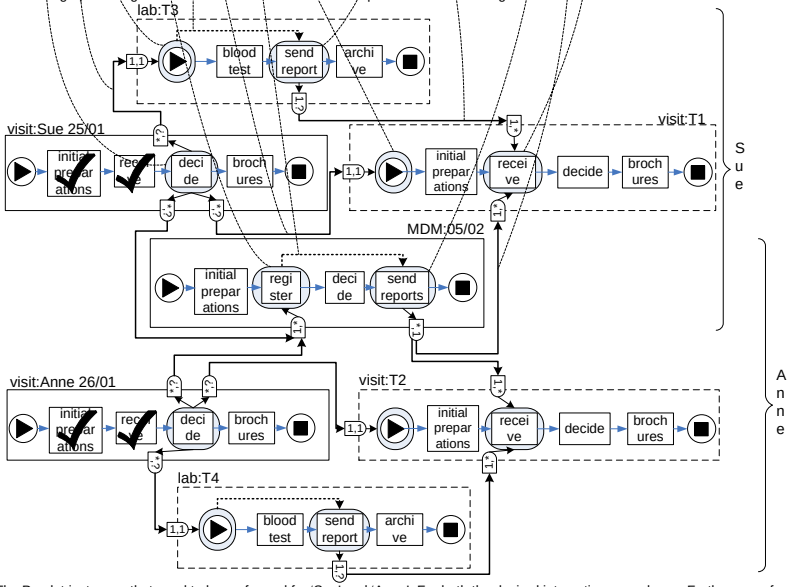
b) class diagram containing the three Proclet classes

c) examples of port attributes

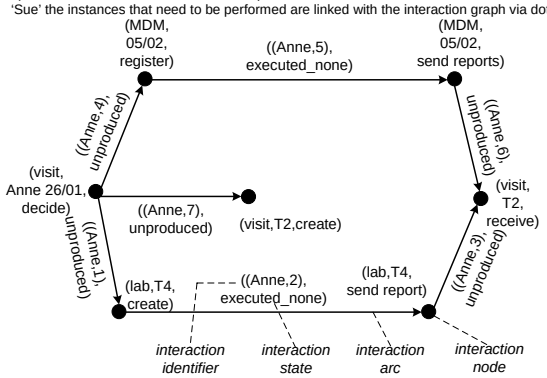
Figure 3: Proclet classes for the scenario illustrated in Figure 1.



a) Interaction graph saving the Proclet instances that need to be performed for 'Sue' together with the desired interactions



b) The Proclet instances that need to be performed for 'Sue' and 'Anne'. For both the desired interactions are shown. Furthermore, for 'Sue' the instances that need to be performed are linked with the interaction graph via dotted arcs



c) Interaction graph saving the Proclet instances that need to be performed for 'Anne' together with the desired interactions

Figure 4: For both ‘Sue’ and ‘Anne’ it is shown how existing and future Proclet instances need to interact. Moreover, the interaction graphs that are created during the execution of the ‘decide’ task are shown.

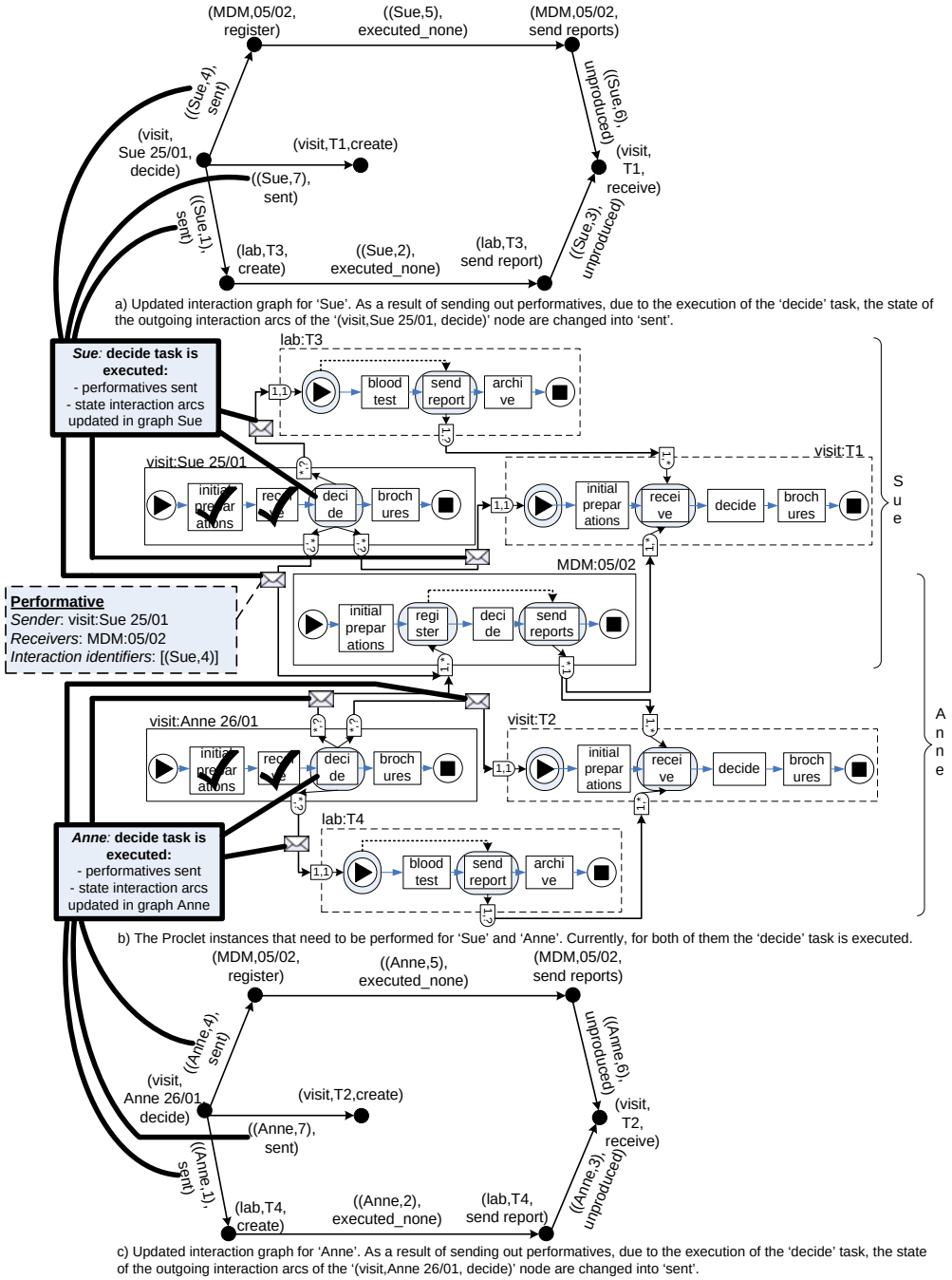
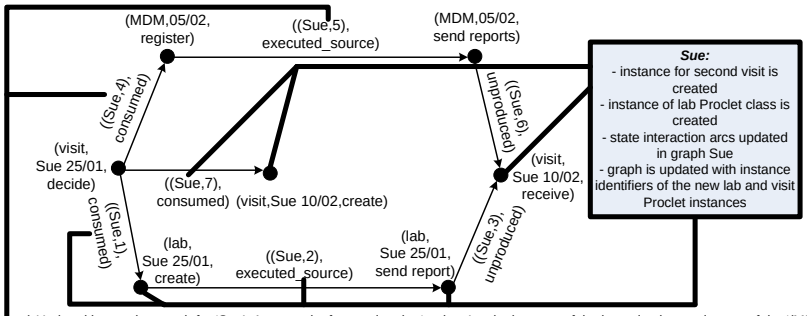
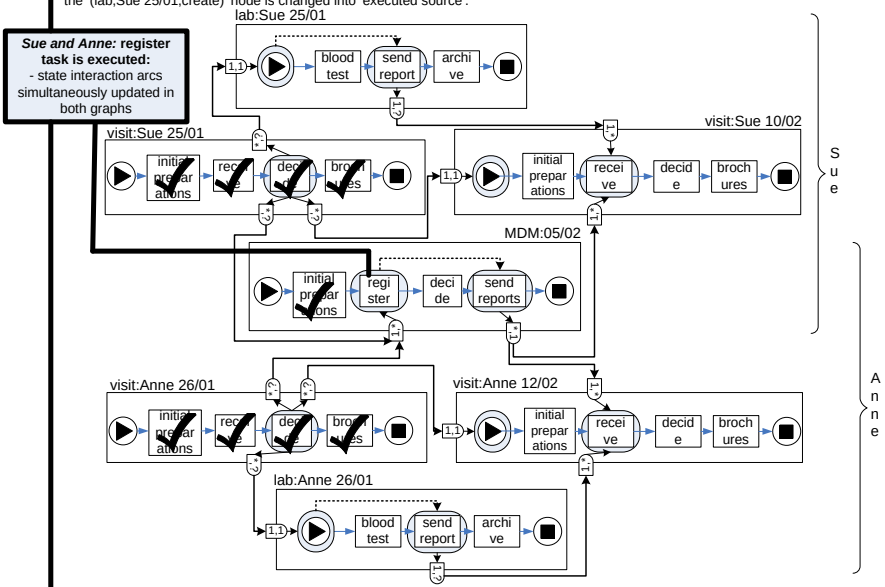


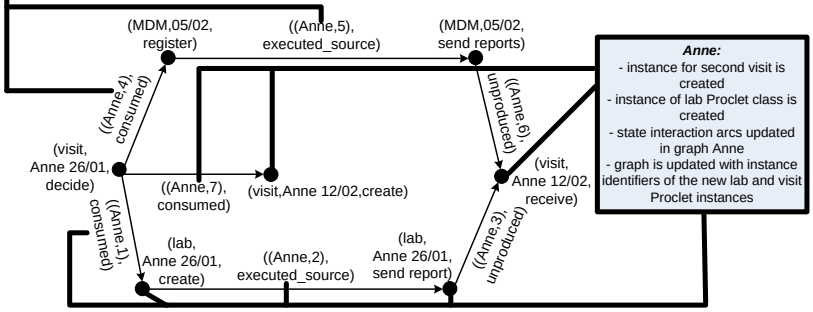
Figure 5: Performatives are sent for both 'Sue' and 'Anne' as a result of executing their 'decide' tasks.



a) Updated interaction graph for 'Sue'. As a result of executing the 'register' task, the state of the incoming interaction arc of the '(MDM,05/02,register)' node is changed into 'consumed' and the state of the outgoing arc of the '(MDM,05/02,register)' node is changed into 'executed source'. Moreover as a result of creating instances for the 'visit' and 'lab' Proclet classes, the state of the incoming arcs of the '(visit,Sue 10/02,create)' and the '(lab,Sue 25/01,create)' nodes are changed into 'consumed'. Additionally, the state of the outgoing arc of the '(lab,Sue 25/01,create)' node is changed into 'executed source'.

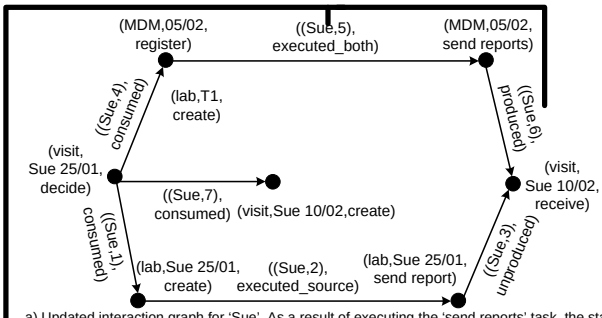


b) The Proclet instances that need to be performed for 'Sue' and 'Anne'. For both the desired interactions are shown

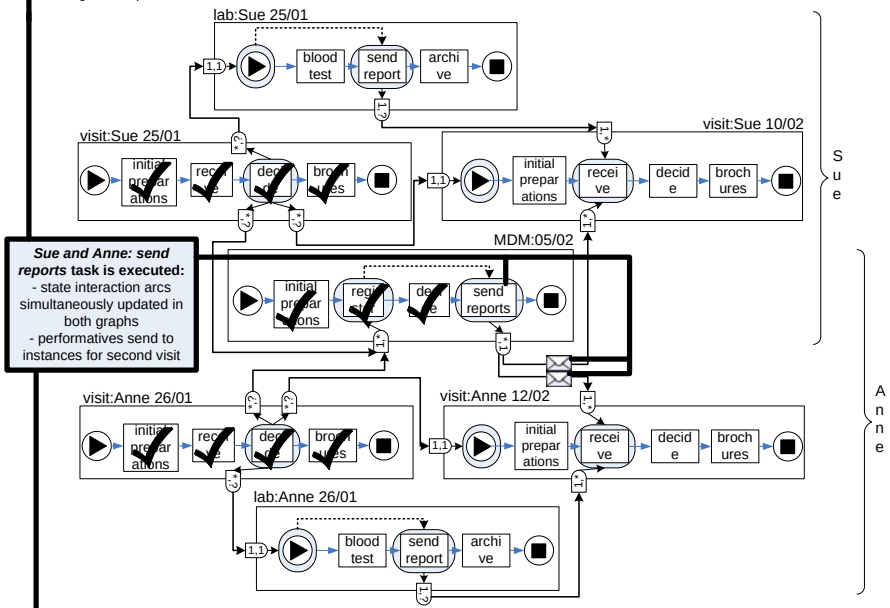


c) Updated interaction graph for 'Anne'. As a result of executing the 'register' task, the state of the incoming interaction arc of the '(MDM,05/02,register)' node is changed into 'consumed' and the state of the outgoing arc of the '(MDM,05/02,register)' node is changed into 'executed source'. Moreover as a result of creating instances for the 'visit' and 'lab' Proclet classes, the state of the incoming arcs of the '(visit,Anne 12/02,create)' and the '(lab,Anne 26/01,create)' nodes are changed into 'consumed'. Additionally, the state of the outgoing arc of the '(lab,Anne 26/01,create)' node is changed into 'executed source'.

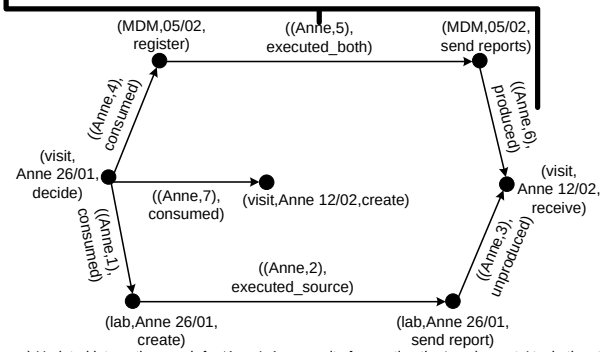
Figure 6: The ‘register’ task of the multidisciplinary meeting is executed.



a) Updated interaction graph for 'Sue'. As a result of executing the 'send reports' task, the state of the outgoing interaction arc of the '(MDM,05/02, register)' node is changed into 'executed both' and the state of the outgoing arc of the '(MDM,05/02,send reports)' is changed into 'produced'.

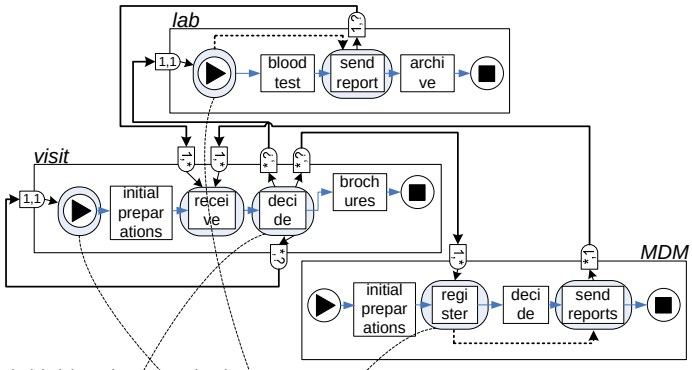


b) The Proclet instances that need to be performed for 'Sue' and 'Anne'. For both the desired interactions are shown

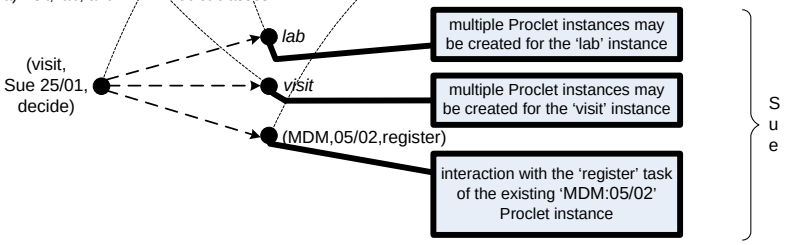


c) Updated interaction graph for 'Anne'. As a result of executing the 'send reports' task, the state of the outgoing interaction arc of the '(MDM,05/02, register)' node is changed into 'executed both' and the state of the outgoing arc of the '(MDM,05/02,send reports)' is changed into 'produced'.

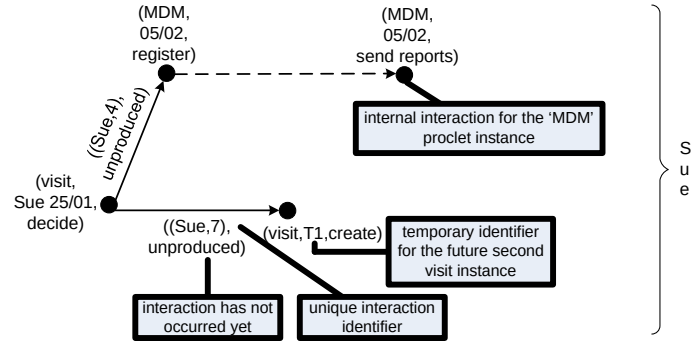
Figure 7: The ‘send reports’ task of the multidisciplinary meeting is executed.



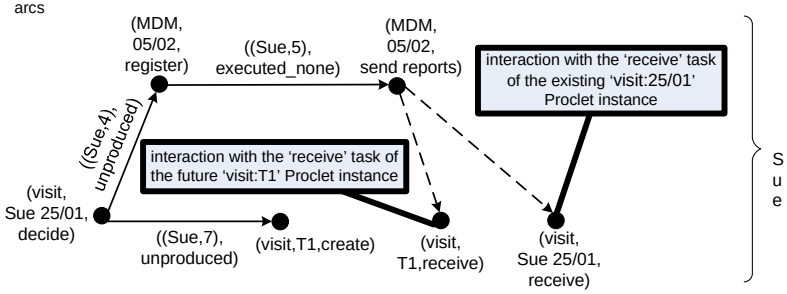
a) visit, lab, and MDM Proclet classes



b) Interaction graph that has been defined for the entity 'Sue' so far. Additionally, by the dotted arcs the next interactions that can be nominated are visualized

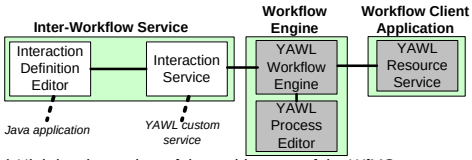


c) For the '(MDM,05/02,register)' node the next interactions that can be nominated are visualized by dotted arcs

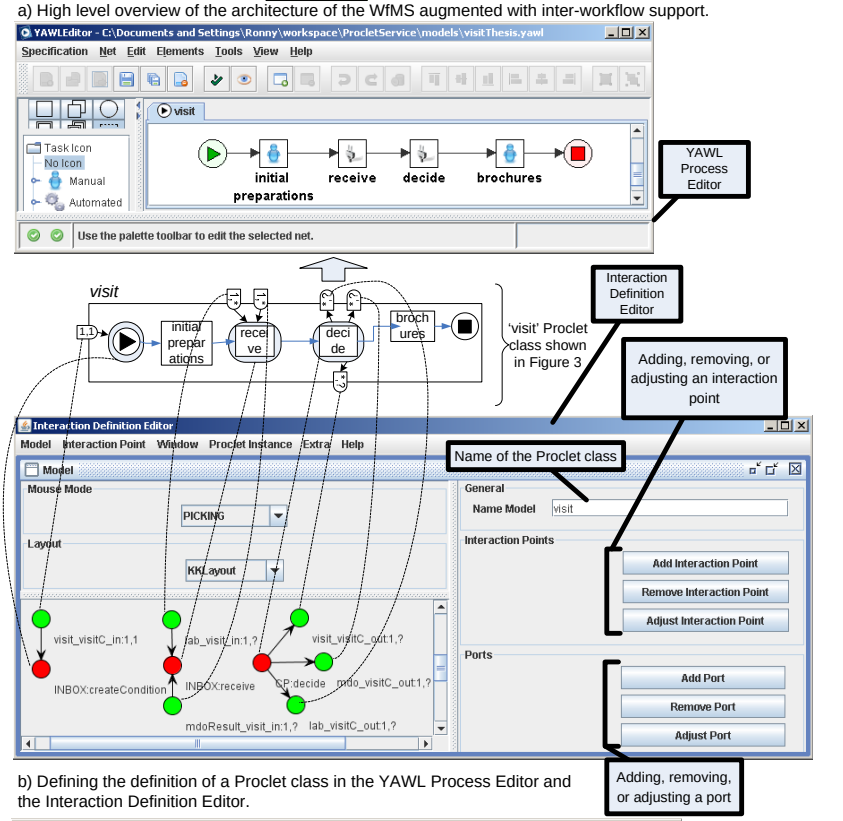


d) For the '(MDM,05/02,send reports)' node the next interactions that can be nominated are visualized by dotted arcs

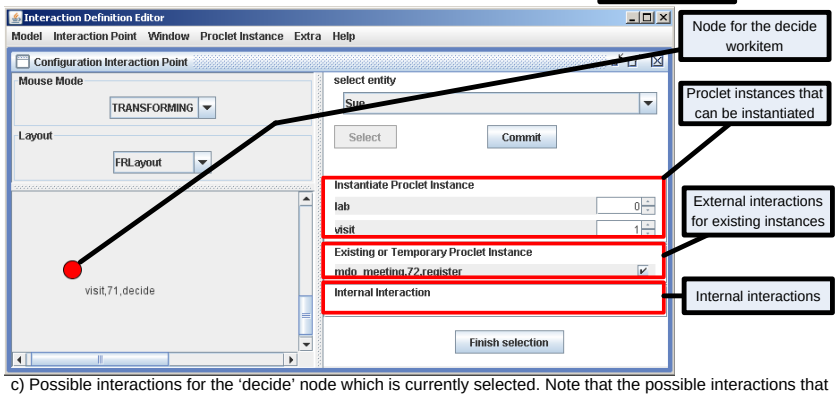
Figure 8: Defining an interaction graph for ‘Sue’. Possible interactions for an interaction node are indicated by dotted arcs.



a) High level overview of the architecture of the WfMS augmented with inter-workflow support.



b) Defining the definition of a Proclet class in the YAWL Process Editor and the Interaction Definition Editor.



c) Possible interactions for the 'decide' node which is currently selected. Note that the possible interactions that can be nominated are the same as in Figure 8b.

Figure 9: Architecture of the realized system. Moreover, several screenshots of the realized system are shown.