1 Introduction

In this chapter we introduce the YAWL4Healthcare WfMS. This is a prototype implementation of a WfMS augmented with calendar-based scheduling support and inter-workflow support. It has been realized in combination with the YAWL WfMS (version 2.1 final). For this chapter we assume that the reader is already familiar with the YAWL WfMS. Extensive information about YAWL can be found on www.yawlfoundation.org and in the user manual [5]. Moreover, we also assume that the reader is already familiar with the theoretical foundations of the scheduling support and inter-workflow support features that are added to the YAWL WfMS. Extensive theoretical information regarding scheduling support and inter-workflow support can be found in Chapters 5 and 6 of [7].

The remainder of this tutorial is organized as follows. First, in Sections 2 and 3 the calendar-based scheduling support and inter-workflow support capabilities of the system are presented. Afterwards, in Section 4, the architecture of the realized system is presented. Finally, in Section 5 some links are provided.

2 Calendar-based Scheduling Support

In this section, the calendar-based scheduling support features of the YAWL4Healthcare WfMS are presented. These features will be presented in the context of the running example that is shown in Figure 1. Note that an elaborate discussion of the running example can be found in Chapter 5 of [7]. Next to that, note that for demonstration purposes the physical resources are not taken into account.

First, we show in Section 2.1 how a model can be defined in order to allow for scheduling support. Then, in Section 2.2, it is discussed how these models are enacted such that cases can be created and that tasks, which are either scheduled or unscheduled, can be performed. Next, in Section 2.3, it is explained how exceptions in the context of scheduled tasks can be handled.

2.1 Modeling Support

In order to make use of the calendar-based scheduling features of the YAWL4Healthcare WfMS, first a scheduling EWF-net (sEWF-net, for more details see Chapter 5 in [7]) needs to be defined.
As part of this sEWF-net it needs to be defined which resources exist in an organization and the roles that they have. Roles can be defined in the “Organizational Data” section of the YAWL Control Centre. Figure 2a shows the roles that are defined for the human resources in the running example. Moreover, it can be seen that “Jane” and “Fred” are members of the “assistant” role. Resources can be defined in the “Users” section of the YAWL Control Centre as shown in Figure 2b. For us it is only important that for each resource personal data, user identifier, password and the associated role are defined. So, no privileges need to be defined. For example, human resource “Jane” has “Edwards” as her last name and she is member of the “assistant” role.

The remaining details of the scheduling EWF-net need to be defined via the YAWL editor. Figure 3 shows how the remaining details in the running example are defined in the YAWL editor. In Figure 3a the control-flow logic is defined, i.e. tasks, conditions, the flow relation, and the split- and join-behavior of each task. Schedule tasks are indicated by a calendar icon whereas flow tasks are indicated by a single person icon. Note that these icons do not influence the task’s behavior.

For each task extended attributes can be defined (see Figures 3b and 3c). From these attributes, the “caseResource”, “duration”, “roles”, and “type” attributes are part of the scheduling EWF-net (indicated by a rectangle). The “type” attribute specifies whether a task is a schedule or a flow task. Via the “duration” attribute the average duration of a task is specified whereas via the “roles” attribute the associated roles are defined. Finally, the “caseResource” attribute specifies whether the human resource for whom the case is being performed is also required to be present. For example, for the “receive telephone
Fig. 2. Defining the organizational details of the scheduling EWF-net.

call” flow task it is specified that it is a flow task and requires 20 minutes on average to complete. For the “physical examination” schedule task, “assistant” and “nurse” have been defined as roles.

2.2 Enactment Support

In this section, we discuss how the model defined in the previous section is enacted.
Fig. 3. Defining the remaining details of the scheduling EWF-net.

Scenario The enactment support will be discussed in the context of the following scenario which is schematically depicted in Figure 4. At the top of the figure, the process model of the running example is shown while at the bottom the calendars of patient “John” and the human resources are depicted.

As can be seen on the left, the current time is 08:30. Now, the scenario commences by starting a case for patient “John”. When moving from the left to the right of the process model we see that subsequently the schedule tasks “physical examination” and “consultation” need to be scheduled. For the proper scheduling of these two tasks it is important that enough time is reserved for the preceding tasks to be completed, i.e., the time constraints of the entire workflow need to be taken into account. This can be achieved in the following
Fig. 4. Scenario in which a case is started for “John”. As a result, appointments need to be scheduled for the “physical examination” and the “consultation” tasks. For the scenario, only the calendars of patient “John” and the human resources need to be considered in order for making appointments. With regard to appointments in calendars, an appointment for a “physical examination” is abbreviated by “PE” whereas an appointment for a “consultation” is abbreviated by “CON”. Note that the start and end times of tasks are anticipated times.

In this way. Note that for the scenario we assume that a resource starts working on a flow task once it is enabled. For the first task at most 30 minutes are required to complete it. In this way, it can be started at 08:30 and at will be finished at 09:00 at the latest. In a similar fashion, the next task “check patient data” can be started at 09:00 and finished at 09:30 at the latest. Next, an appointment for the “physical examination” task can be scheduled starting from 09:30. As for the tasks, the “assistant” and “nurse” roles have been defined, the availability of “Jane” or “Fred” and “Sue” or “Rose” need to be taken into account. Moreover,
the availability of the patient needs to be checked too. Due to the unavailability of “Fred” and “Jane”, as can be seen in Figure 4, the first possible appointment can be scheduled for 10:00 to 11:00 and involves “John”, “Jane” and “Sue”.

Subsequently, for the booking of an appointment for the “consultation” task, it needs to be taken into account that the “physical examination” task finishes at 11:00 and the “make documents” task at 10:00 at the latest. As a result, the appointment can be booked from 11:00 on. Based on the availability of the patient “John” and either “Nick” or “Marc”, the appointment is booked from 11:00 to 12:00 in the calendars of “John” and “Nick”. No further appointments need to be booked which finishes the scheduling process.

**Enactment** Now the enactment of the scenario in our system will be illustrated. Here, the users working with a WfMS can only communicate with the system via the Workflow Client Application. As shown in Figure 22c, for the YAWL4Healthcare WfMS this is done via an Outlook 2003 client. Figure 5 illustrates the user interface that is shown when opening this client. In addition to the default Outlook folders, there are five folders which are important for the enactment of processes. These folders have been indicated by different callouts and are the “Active case”, “Allocated work”, “Available processes”, “Available work” and the “Calendar” folders. The roles of these folders will become clear when demonstrating the scenario.

![Fig. 5. Interface which is shown when opening the Outlook 2003 client.](image-url)
Start a Case In order to start a case for a process, the “available processes” folder needs to be selected which shows all the processes that are uploaded to the system. As can be seen at the right-side of Figure 5, there are two processes available. When selecting the “oncology” process, the form shown in Figure 6 is presented. The “Output Variables” panel shows the data elements that need to be filled in order to start a case. Here the “caseResource” data element has a special role as it allows for the selection of the calendar of the human resource for which the case needs to be performed. For the scenario, it can be seen that “John’s” calendar has been selected. Moreover, the messagebox indicates that a case has been started with identifier “63”.

![Form for starting a case](image)

**Fig. 6.** A case has been successfully started for patient “John”. The new case has “63” as its identifier.

As a result of starting a case, appointments are booked for the “physical examination” and “consultation” tasks. In Figure 7, the calendars of patient “John” and the human resources are shown before and after scheduling. For example, in the calendars before scheduling it can be seen that “John” is not available from 08:00 to 09:00 because he needs to travel and that “Sue”, “Rose”, “Marc”, and “Nick” are not available from 12:00 to 13:00 because of lunch. The calendars after scheduling are shown in Figure 7b. Here it can be seen that the “physical examination” has been scheduled for 10:00 to 11:00 in the calendars of patient “John”, assistant “Jane”, and nurse “Sue”. The “consultation” task has been scheduled for 11:00 to 12:00 in the calendars of patient “John” and doctor “Nick”. Remember that the YAWL4Healthcare WfMS only takes the average duration of a task into account for the scheduling. Therefore, for the
case which is started at 08:30, the earliest time that the “physical examination”
may be scheduled is 09:10. However, assistant “Anne” is only available after
10:00 and assistant “Fred” is not available at all, which impacts the scheduling.
Moreover, it is important that the “consultation” task is scheduled after the
“physical examination task”, which is also consistent with the corresponding
process definition.

Fig. 7. State of the calendars before and after scheduling. The calendars of “John”,
“Fred”, “Jane”, “Sue”, “Rose”, “Marc”, and “Nick” are shown respectively. For an
appointment related to a schedule workitem the process identifier, instance identifier,
and task identifier are shown respectively.

Perform a Workitem For selecting and performing workitems for flow tasks,
the “Available work” and the “Allocated work” folders are important (see Fig-
ure 5). In particular, by selecting the “Available work” folder, a resource can
see all the workitems that he is allowed to allocate to himself. For example, in
Figure 8a, the content of this folder is shown for “Sue”. As only one case is run-
ning, she can only select the workitem for the “register patient” task. Moreover,
a messagebox shows that a workitem has been successfully allocated to “Sue”. Note that additional information for the workitem is provided in the “Enablement Time” column which shows the time at which the workitem was enabled. Moreover, if there is a question mark in the “!” column, then limited time is left in which to perform the workitem (not shown in the figure).

By selecting the “Allocated work” folder, a resource can see all the workitems that have been allocated to them and that may be completed. For example, in Figure 8b, the content of this folder is shown for “Sue”. Not surprisingly, the workitem for the “register patient” is the only one which is present in the list. When clicking on the item, the associated form for completing the workitem is shown. As part of this, some data elements need to be filled in (e.g. “John Wales” has been filled in for the name of the patient). As can be seen in the figure, “Sue” has successfully completed the workitem. In addition to completing a workitem there is also the option to deallocate the workitem.

Note that additional information for the workitem is shown in the “!” column, “Enablement Time”, and “Start Time” columns. Here, the “Start Time” column shows the time the workitem is allocated.

The form for completing a workitem for a schedule task looks similar to the form shown in Figure 8b. Such a form can be obtained by clicking on an appointment in the calendar for which a workitem exists. An example is shown in Figure 9.

2.3 Exception Handling

In this section, the exception handling capabilities, in the context of appointments that are scheduled, are discussed.

Automatic Rescheduling For the YAWL4Healthcare WFMS, an exceptional situation occurs if, based on the average duration of tasks, too little time is left in which to perform workitems of tasks preceding an appointment. In that case, the appointment itself is rescheduled. Moreover, subsequent appointments are also rescheduled, if needed. Remember that for a case after checking in a workitem and at regular intervals it is checked whether tasks need to be (re)scheduled. Also, the rescheduling is done automatically, i.e. no user involvement is necessary.

Rescheduling on Request The rescheduling of an appointment can also be requested by a resource itself. As can be seen in Figure 9, the “Calendar” folder can be clicked in order to see all the appointments in the calendars of one or more resources. In particular, the leftmost column shows the calendar of the resource itself. Additionally, for this user, a partially white line at the left side of an appointment indicates that a workitem exists and that it can be completed. For example, for the “physical examination” a workitem exists.

Moreover, the associated form of the “physical examination” workitem is shown. Next to the fields for filling in or displaying the value of data elements for the workitem, three buttons are grouped together. Each of these three buttons
a) Allocating a workitem. The client for Sue is shown.

The workitem has been allocated to Sue.

Column for indicating whether limited time is left in which to perform the workitem.

Time at which the workitem was enabled.

b) Completing a workitem. The client for Sue is shown.

Deallocating the workitem.

The workitem has been completed.

Column for indicating whether limited time is left in which to perform the workitem.

Time at which the workitem was enabled.

Fig. 8. Executing a workitem for a flow task.
allows for manipulation of the appointment in own specific ways. Their specific purpose is as follows.

- reschedule: the resource which requested the rescheduling of the appointment will stay involved.
- reject: the resource which requested the rescheduling of the appointment will not be involved in the appointment anymore.
- specific time: the appointment is moved to a specified date and time.

In order to illustrate the rescheduling of an appointment, below we consider the rescheduling of an appointment to a specific time. So, let us assume that “John” is suddenly not available till 12:00. In order to also incorporate some time for traveling it is decided that the appointment should be moved to 13:00. The effect of the specific rescheduling request can be seen in Figure 9. In this figure, the message box indicates that the “physical examination” has been successfully rescheduled to the requested time. Moreover, we can also see that it was necessary to reschedule the appointment with doctor “Nick” which will now take place from 14:00 to 15:00. As can be checked in Figure 1, this rescheduling step is necessary as the task “consultation” occurs after the “physical examination” task.

Cancelation of a Case  Looking back at Figure 5, we can see that there is a folder called “Active cases”. When clicking on the folder, a list of cases which are currently running in the system can be seen (see Figure 10). For each item on the list, the process identifier and the case identifier can be seen. Moreover, from this list, a case can be selected for cancelation. As a result, all appointments for the case are removed from the calendars of the resources. In this way, we can cater for situations in which a case for a patient is not continued anymore.

For example, if patient “John” suddenly dies then all appointments and the case itself needs to be canceled. This can be achieved by double-clicking on the “63, oncology” item (Figure 10). Afterwards, the form shown in the figure is presented. When clicking on the “Cancel case!” button in the form, the case will be canceled. This is illustrated by the messagebox which indicates that the case with identifier “63” has been successfully canceled.

3 Inter-Workflow Support

In this section, the inter-workflow support features of the YAWL4Healthcare WfMS are presented. These features are presented in the context of the scenario that is shown in Figure 11. Note that an elaborate discussion of the scenario can be found in Chapter 6 of [7]. In Figure 11a the patient processes of both “Sue” and “Anne” are shown. Next, in Figure 11b the corresponding Proclet classes are shown together with their connections. Remember that for “Sue” during the first visit it was decided that a second visit is needed, and that she needs to be discussed during a multidisciplinary meeting. Moreover, the result of the multidisciplinary meeting is required as input for the second visit. For “Anne” the process is the same.
Rescheduled appointments
Form for completing a schedule workitem
Manipulation of an appointment
Original appointments
Data elements

A workitem exists for the physical examination

Fig. 9. Rescheduling of the “physical examination” to 13:00. Note that the calendars of “Jane”, “John”, “Sue”, “Nick”, “Fred”, “Rose”, and “Marc” are shown respectively.
The case with identifier '63' has been canceled.

Identifier of the case that is running
Identifier of the process of the case that is running

Fig. 10. Viewing a list of all cases that are running in the system.

Note that in comparison to the scenario presented in Chapter 6 of [7], an exception interaction point has now been defined for the “MDM” Proclet class. When an exception occurs for the “MDM” Proclet class, this exception interaction point allows that for an entity affected by the exception, an interaction with the “register” task of an existing or future “MDM” Proclet instance can be defined, i.e. a patient can be registered for another multidisciplinary meeting.

First, we discuss in Section 3.1 how a model can be defined in order to allow for inter-workflow support. Then, in Section 3.2, it is explained how these models are enacted such that cases can be created and that interactions between Proclet instances can be realized. Next, in Section 3.3, it is elaborated upon how exceptions can be handled.

3.1 Modeling Support

In order to make use of the inter-workflow support features of the YAWL4-Healthcare WfMS, first Proclet classes and the external interactions between them need to be defined. Note that process definitions are defined at the engine side (YAWL) and that the extensions on top of these definitions are defined at the Inter-Workflow Service side (via the Interaction Definition Editor).

YAWL Editor Figure 12 shows how the process definition of the “visit” Proclet class is defined via the YAWL editor. Every task in the process definition is a flow task as they do not need to be scheduled. However, tasks for which interactions may be necessary are indicated by a plug-in icon. The execution of the corresponding workitem for them is delegated to the Inter-Workflow Service.
This needs to be defined via the “Task Composition Details” of the task. Flow tasks which need to be performed via a work tray are indicated by a single person icon.

For both the “initial preparations” and “decide” task the corresponding task decomposition details are shown. In Section 4.2 it has been indicated that the Inter-Workflow Service has been implemented as a YAWL Custom Service. This means that the execution of a workitem for a task can be delegated to a service if needed. The “YAWL Registered Service Detail” field of the decomposition details for the “decide” task shows that the execution of the workitem is delegated to the Inter-Workflow Service. For the “initial preparations” task it is defined that it can be performed via the Workflow Client Application. Moreover, it can be seen that for both tasks, an “entities” data variable has been defined which is a complex data type. The reason for defining the variable is as follows.
In order to create an interaction graph, a user needs to provide at run-time the names of the entities for which such a graph needs to be created. Then, the graphs can be created if a workitem for a configuration interaction point is performed, i.e. in this case if a workitem for the “decide” task is performed. As the Inter-Workflow Service has been implemented as a YAWL Custom Service this has the consequence that, at run-time, the names of the entities can only be passed on to the service via a workitem whose execution is delegated to the service. More precisely, this can only be achieved via a data variable that has been defined for the task of the workitem.

Consequently, the names of the entities need to be provided if the workitem for the “initial preparations” task is performed. As the workitem is performed via the Workflow Client Application, the names of the entities can easily be filled in via the form that is automatically generated. Afterwards, these names are passed on the service via the “entities” variable if the workitem for the “decide” task is performed.

Fig. 12. Defining the process definition of a Proclet class in the YAWL editor.
Interaction Definition Editor  Figure 13a shows the GUI of the Interaction Definition Editor that allows for the definition of interaction points, ports, and internal interactions for an existing process definition. More specifically, at the
right side of the GUI, these details are defined whereas at the left side they are visualized.

In order to illustrate the definition of these details, Figure 13a shows the interaction points, ports, and internal interactions that are defined for the “visit” Proclet class (this is defined in the “Name Model” field at the top right). Note that at the top of Figure 13a the “visit” Proclet class is shown. Via dotted arcs, interaction points and ports are linked with the associated interaction points and ports that are shown in the GUI. At the left side, interaction points are visualized by a black dot together with the type of the interaction point and its identifier. As a special case of this, an exception interaction point is visualized as a grey dot. Ports are visualized by a white dot together with its identifier and the associated cardinality and multiplicity. Moreover, via an arc it is indicated to which interaction point a port belongs to. If an arc is leading from an interaction point to a port then we are dealing with an output port whereas for an input port this is exactly the opposite. Note that an internal interaction has not been defined for the “visit” Proclet class. Internal interactions are visualized via a dotted arc that leads from the source to the destination interaction point.

Interaction points, ports, and internal interactions can be manipulated via the “Interaction Points”, “Ports”, and “Internal Interactions” panels respectively. The associated panels for editing the details of an interaction point, a port, and an internal interaction are shown in Figures 13b to 13d respectively. In particular, in Figure 13b it is defined that the “receive” interaction point is an inbox interaction point, that no instance of the Proclet class needs to be created once it is triggered, and that the timeout value is 12.000 milliseconds. In Figure 13c it is defined that the “lab_visit_in” port is an input port and attached to the “receive” interaction point. Moreover, it has cardinality “1” and multiplicity
"*". Finally, in Figure 13d, the source and destination interaction point of an internal interaction can be selected.

Once Proclet classes have been defined, external interactions can be defined. The corresponding GUI is shown in Figure 14. At the right side of the GUI external interactions can be specified or removed whereas at the left side they are visualized.

In order to illustrate the definition of these details, Figure 14 shows a part of the external interactions that are defined for the example in Figure 11b. At the left side, ports are visualized by a green dot together with the associated identifier. Moreover, the direction of the arc between two ports indicates the source (tail) and the destination (head) of the external interaction. For example, the “lab_visit_out” output port is connected with the “lab_visit_in” input port. Via this connection, the result of a lab test is sent to the “receive” task of a “visit” Proclet.

3.2 Enactment Support

In this section, we demonstrate how Proclet classes can be enacted in the YAWL4Healthcare WfMS. This is done in the context of the running example that is discussed at the beginning of Section 3. For the demonstration we assume that all the Proclet classes of Figure 11b and their relationships have been configured in the system.

The demonstration is started by “Sue” for whom a visit to the outpatient clinic is required. So, for “Sue” an instance of the “visit” Proclet class exists which has “64” as instance identifier. However, in order to be able to have interactions with other Proclets, first an entity identifier should be created. As discussed before, this can be done during the “initial preparations” task. The corresponding form in the Outlook client is shown in Figure 15a. Via the “entities” data variable, data for entities can be filled in. In the “entity_id” field, the entity identifier can be filled in. Moreover, some additional data can be filled in via the “name value pair” elements. As the value of the “entities” data variable has a complex data type, in Figure 15a this is represented as XML code. However, in the Workflow Client Application of the YAWL WfMS, a form is automatically created for any (complex) data type which is more user friendly. The corresponding form is shown in Figure 15b.

For both forms, we see for “Sue” that “Sue” has been filled in as the entity identifier, she has age 65, and she is 1.75 m long. Remember that the additional data that has been filled in is included in the performatives that are sent from this Proclet instance. In this way, the data becomes available to the Proclet instances to which the performative is sent.

Afterwards, during the “decide” task, it can be defined what needs to be done next. As the “decide” task is associated with a “configuration” interaction point, the interaction graph for “Sue” can be extended. Below we demonstrate via several screenshots how this is supported in our system. Note that we assume that the reader is familiar with how an interaction graph can be extended. More details can be found in Chapter 6 of [7].
Fig. 15. Form in which the identifier for an entity can be filled in. Moreover, some additional data can be filled in. If later the workitem for the “decide” task is executed, the information can be used for creating an interaction graph.

First, in the Interaction Definition Editor, “Sue” is selected as the entity (see Figure 16a). Afterwards, the panel shown in Figure 16b is presented. At the left
Fig. 16. Extending the interaction graph for Sue.
a) Updated graph after selecting the creation of an instance for the 'visit' Proclet class. Moreover, Sue is registered for the MDO meeting.

Interactions can be selected for the first visit and multidisciplinary meeting.

No interactions can be selected for the visit.

Selection of interactions for the multidisciplinary meeting.

Ultimately, the 'send report' task is performed.

b) Possible interactions for the multidisciplinary meeting.

Fig. 17. Extending the interaction graph of Sue with an internal interaction.

side, the interaction graph that has been defined so far is shown. In particular, for nodes that have been colored white, new interactions can be selected. For nodes that have been colored black this is not possible. Following on, at the right side, one of these white colored nodes can be selected in order to define new interactions. For example, for “Sue” we see at the left side a node in the graph for the “decide” workitem. At the left, this node can be chosen for selecting new interactions. Note that for the arc states in an interaction graph, abbreviations will be used. So, for the arc states “unproduced”, “consumed”, “sent”, “executed
Fig. 18. Final graph for Sue. Moreover, as a result of the graph, the workitems that need to be performed are shown.

none”, “executed single”, “executed both”, and “failed”, the abbreviations “U”, “C”, “S”, “EN”, “ES”, “EB”, and “F” will be used respectively.

For a node that is selected, a new screen is presented which is shown in Figure 16c. At the left side, the interaction graph that has been defined so far is shown. In particular, the node that has been selected for defining new interactions is colored white. The other ones are colored black. At the right side,
for the selected node, the possible interactions are presented via three different panels. In the “Instantiate Proclet Instance” panel the interactions are shown 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 by checking the checkbox of the respective interaction. Note that future Proclet instances have a negative case identifier. In the “Internal Interaction” panel, internal interactions can be selected by checking the checkbox of the respective internal interaction. For example, for “Sue” one instance of the “visit” Proclet class will be created. Moreover, for the multidisciplinary meeting an instance exists which has “65” as the instance identifier. As a result, she is also registered for this meeting.

After selection of the interactions, the panel is shown again which allows for the selection of a node for which interactions can be selected. So, the process can be repeated until no more interactions are required. An example of this can be seen in Figure 17 which is a follow up to Figure 16. In particular, in Figure 17a a node is shown for the future instance of the “visit” Proclet class. Note that this node has a temporary instance identifier which is a negative number. Moreover, via the “mdo_meeting,65,register” node, the patient is registered for the multidisciplinary meeting. For the “mdo_meeting,65,register” node that has been selected, the next screen is shown in Figure 17b. As can be seen on the right side of this figure, there is only an internal interaction that can be selected. By selecting this interaction, it will be assumed that after registering “Sue” for the multidisciplinary meeting, the “send report” task is performed for her.

The resultant graph for “Sue” is shown in Figure 18a. Via rectangles it is indicated which nodes are related. For example, for the multidisciplinary meeting for which “Sue” is registered, the result will serve as input for the next visit (“visit,-40,receive” node). Note that the graph is in line with the example shown in Figure 11a. Also, at the top, the resultant interaction graph for “Sue” is shown. Via dotted arcs, corresponding interaction nodes and interaction arcs are connected. For example, the “(MDM,05/02,register)” interaction node is connected with the “mdo_meeting,65,register” interaction node of the scenario that is executed in the YAWL4Healthcare system.

Moreover, as a result of the interactions that are specified, one new workitem needs to be performed. This is shown in the worklist presented in Figure 18b. Here, the “66:initial_preparations_8” workitem relates to the second visit that is required.

In a similar fashion as for “Sue”, an interaction graph is defined for “Anne” which is shown in Figure 19. Also here, via rectangles it is indicated which nodes are related. Moreover, at the top, the resultant interaction graph for “Anne” is shown. Also here, via dotted arcs, corresponding interaction nodes and interaction arcs are connected. Note the graph is in line with the example shown in Figure 11.
3.3 Exception Handling

In the previous section, we have illustrated the operation of the system under normal circumstances. In this case, we will illustrate the operation of the system where exceptions occur. Therefore, we first demonstrate how an exception at case level is handled followed by the manner in which an exception at workitem level is handled.

**Exception at Case Level** For the demonstration assume that for both “Sue” and “Anne” we have just finished defining the interactions as a result of the “decide” task during the first visit. So, the interaction graphs shown in Figures 18a and 19 apply.

Moreover, we assume that the multidisciplinary meeting with instance identifier “65” can not take place anymore because some of the doctors involved need to attend a conference. As a result, the patients that are discussed during the meeting now need to be discussed during the next multidisciplinary meeting which has instance identifier “69”. In the system this is supported in the following way.

As a result of the cancelation of the multidisciplinary meeting with instance identifier “65” an exception occurred. This is because both for “Anne” and “Sue” it has been defined that the result of their multidisciplinary examination need to
be used as input for the second visit for each of them. This is now not possible anymore.

In Figure 20a, the panel is presented which shows exceptions that occurred at case level and how they can be handled. In the top panel, via the drop down box, instances are shown for which the cancelation or completion resulted in an exception. As a follow up to that, for a selected case, the affected entities
Fig. 21. Handling of the exception which involves a workitem for which not all required performatives are received.

are shown in the middle panel. Finally, in the bottom panel it can be decided how the exception needs to be handled. Either the exception is ignored (“Ignore Exception” button) or for the affected entities the interaction graphs may be extended (“Handle Exception” button).

For the canceled multidisciplinary meeting we see that both “Sue” and “Anne” are affected. For them we decide to click on the “Handle Exception” button. As an example, in Figure 20b it is shown how the interaction graph for “Sue” is extended. That is, using the exception interaction point of the canceled multidisciplinary meeting (node “mdo\_meeting,65,exception”), “Sue” is registered for the multidisciplinary meeting which has “69” as its instance identifier (node “mdo\_meeting,69,register”). Moreover, the result of the meeting is used as input for the second visit. Note that for the interactions that relate to the canceled multidisciplinary meeting it is indicated that they do not take place anymore, i.e. the arcs have state “failed”.

The interaction graph of “Anne” is extended in a similar way. Therefore, it is not shown here.

**Exception at Workitem Level** In this section, we demonstrate how an exception at workitem level is handled by the system. For the demonstration assume that for both “Sue” and “Anne” we have just finished defining the interactions as a result of the “decide” task during the first visit. So, the interaction graphs shown in Figures 18b and 19 apply.

Moreover, assume that now for “Sue” the second visit takes place. As a result, the “initial preparations” task is performed. Next, the result of the mul-
tidisciplinary meeting needs to be received at the “receive” task. However, as the multidisciplinary meeting has not been performed yet, an exception is raised.

In Figure 21, the panel is presented which shows exceptions that occurred on workitem level and how they can be handled. In the top panel, via the drop down box, workitems are shown for which the required interactions have not taken place in time. As a follow up to that, for a selected workitem, the affected entities are shown in the middle panel. Finally, in the bottom panel it can be decided how the exception needs to be handled. Either more time is granted for receiving missing performatives (“Grant More Time” button) or for the affected entities the interaction graphs may be extended (“Handle Exception” button).

For example, for the workitem exception of the “receive” task of the second visit for “Sue”, we see that indeed only “Sue” is affected and not “Anne”. In case it is decided to extend their interaction graphs, this would proceed in a similar way as for the exception at case level. Therefore, it is not shown here.

4 System Architecture

4.1 Overview

In Figure 22a, a conceptual model of the architecture of the YAWL4Healthcare WFMS is shown. The architecture of our YAWL4Healthcare WFMS is based upon the architecture of the conceptual model. This architecture can be found in Figure 22b. Based on these architectures it can be seen how the YAWL4Healthcare WFMS has been realized. In particular, in order to implement the functionality contained in the conceptual model it has been incrementally mapped to an operational system based on widely available open-source and commercial-off-the-shelf (COTS) software. Here, we choose an approach based on the reuse of existing software. In Section 4.2, we explain how each component has been realized. Finally, as the system is based on the YAWL WFMS, we end by discussing in detail which facilities of YAWL are used in our system in Section 4.3.

4.2 Components

Workflow Engine The workflow component is realized using the open-source WFMS YAWL [6, 4] (version 2.1 beta) and a service which acts as an adaptor in-between YAWL and the Workflow Client Application. The adaptor service communicates with YAWL via “Interface B” [6, 3]. The adaptor also communicates with the Scheduling Service using SOAP messages. The adaptor and the YAWL system are tightly coupled as large volumes of work-item and process related information are exchanged.

Calendars For the Calendars component we selected Microsoft Exchange Server 2007 as the system for storing the calendars of users. This system provides several advantages including widespread usage and the fact that it offers several interfaces for viewing and manipulating calendars.
**Workflow Client Application** Once the Exchange Server was in place we could easily use the Microsoft Outlook 2003 client to obtain a view of a user’s calendar. Furthermore, the Outlook client can be configured in such a way that it can act as a full Workflow Client Application which can communicate with the WfMS via an adaptor service through the exchange of SOAP messages.

**Scheduling Service** The Scheduling Service is implemented in Java as a service which communicates with the WfMS via SOAP messages. However, in order to get a view of and to manipulate the calendar, the service also communicates via a Java interface with the Exchange Server which in turn exchanges information via SOAP messages.

**Inter-Workflow Service** As can be seen in Figure 22b, the Inter-Workflow Service consists of an Interaction Service and an Interaction Definition Editor.
Figure 23 shows the interface between YAWL and a Custom Service. Being a YAWL Custom Service means that YAWL delegates a task to the service if this is defined for the task in the YAWL model (cf. task marked with letter “S” in Figure 23). In our case this would mean that for a task for which interactions may occur, the execution of a workitem for it, is delegated to the Interaction Service. Moreover, a custom service can request the launching of a new case in YAWL or get an overview of the cases that are running in the system. Also, the service is notified about the cancelation or completion of a case. Note that the interface that is offered by being a Custom Service is in line with the interface that is defined between the Workflow Engine and the Inter-Workflow Service in Figure 22b.

4.3 Usage of the YAWL WfMS

In Figure 22c it is shown that the YAWL WfMS is used for a part of the workflow engine. In this section, it is discussed which particular components of the YAWL WfMS are used and in which way. Therefore, Figure 24 again shows the architecture of the concrete implementation of the system. However, instead of the YAWL rectangle in the engine component we see now the architecture of the YAWL WfMS of which the grey colored components are (partly) used by our system.

In the picture of the architecture of the YAWL WfMS five components are shown. The YAWL Workflow Engine is used for managing the execution of cases. Moreover, the engine takes care of determining which workitems should be offered for processing and which events should be announced to the environment. Next, the YAWL Process Editor is a design environment for the creation and verification of YAWL process specifications. In our case, the process editor is used for partly defining scheduling EWF-nets (sEWF-nets). In particular, via
Fig. 24. Architecture outlining which components of the YAWL WfMS have been used for the Workflow Engine of our system.

the editor, the control-flow logic can be defined and for each task, the associated roles and its average duration can be defined. Moreover, for each schedule task it can be defined whether the human resource for whom the case is being performed is also required to be present. In order to define the organizational details of an sEWF-net (the resources and the mapping of these resources to roles), the Resource Service needs to be used. The basic role of the Resource Service is to allocate enabled workitems to resources so that they can be processed.
Note that in our system there are some slight deviations from the definition of a schedule EWF-net in Chapter 5 of [7]. For reasons of simplicity we only define an average duration of a task and assume that it is defined in minutes. So, for example, the maximal or minimal duration of a task is not taken into account. In Section 2.1 it will be visualized how an sEWF-net is defined.

As already indicated, the Interaction Service of the Inter-Workflow Service has been implemented as a YAWL Custom Service. Here, Proclet classes have been defined as an extension of EWF-nets (e.g. interaction points, ports, cardinality, and multiplicity). As can be seen in Figure 24, these extensions are stored at the Inter-Workflow Service side. In particular, for an EWF-net these extensions can be defined via the Interaction Definition Editor. Moreover, they can be accessed by both the Interaction Definition Editor and the Interaction Service. More details about defining these extensions can be found in Section 3.1.

Finally, the Worklet Service enables dynamic flexibility for process instances via the Selection Service. Moreover, facilities to handle both expected and unexpected process exceptions at run-time are provided by the Exception Service. Note that the Worklet Service is not used by our system.

5 Links

For the YAWL4Healthcare system an environment to test and play with the system is provided via SHARE [1]. The environment includes the system itself, a tutorial, two screencasts, and several input models. The environment can be accessed from [2] after registration. Furthermore, additional information including screencasts can be found on [2].

References

1. SHARE Website. http://is.tm.tue.nl/staff/pvgorp/share/.