GDAŃSK UNIVERSITY OF TECHNOLOGY

TASK Quarterly 28 (2) 2024

Document coordination patterns

Magdalena Godlewska [1], **Bogdan Wiszniewski** [2]
Dept. of Intelligent Interactive Systems, Faculty of Electronics, Telecommunications and Informatics,
Gdańsk University of Technology, Narutowicza 11/12, 80-233, Gdańsk, Poland

https://doi.org/10.34808/tq2024/28.2/a

Abstract

Many organizations lack support for document workflow management, making it difficult to efficiently handle business processes. Implementing a workflow management service that encompasses all organizational tasks is both complex and costly. However, document workflows can be observed as structured sets of identifiable workflow patterns. These patterns have been extensively described in the literature based on real-world organizational processes. By adopting and integrating these patterns into document workflows, it is possible to enable documents to autonomously determine the processes they should execute. The paper explores the application of a well-known set of workflow patterns in email-based document workflows and outlines the necessary conditions for integrating workflow management directly into documents. Documents circulating within an organization can easily collect information about their workflow in the form of logs with minimal effort. By applying process mining techniques, it is possible to extract the actual processes they follow. The canonical set of document workflow patterns proposed in this paper enables a much faster implementation of a document management application compared to a top-down approach, where processes are theoretically defined first and then implemented.

Keywords:

Workflow management, Task coordination, Process mining

[1]Email: maggodle@pg.edu.pl
[2]Email: bogwiszn@pg.edu.pl

1. Introduction

Document workflow is crucial in organizations as it ensures efficiency, transparency, and proper record-keeping. However, the available technical support is often insufficient, leading to frustration and inefficiencies among employees. When technical issues arise, delays occur, disrupting operations and reducing productivity. Moreover, inadequate support can cause stress and dissatisfaction among staff, as they struggle with system errors, lack of training, or slow response times from IT teams. These emotional reactions can further impact workplace morale and overall organizational effectiveness, making it essential to improve and streamline document management systems.

Cloud solutions are very fashionable now, in this processing model document management systems can be found. Azmir and Wijayanti listed an overview of this type of solution [1] and summarized its main advantages and disadvantages. Thus, cloud-based electronic document management offers several advantages. It provides flexibility and accessibility, allowing users to access documents from anywhere with an internet connection. Additionally, cloud solutions often reduce costs by eliminating the need for extensive on-premise infrastructure and maintenance. They also enhance collaboration, enabling multiple users to work on the same documents in real time. However, there are also challenges. Security and data privacy remain significant concerns, as storing sensitive information in the cloud increases the risk of cyber threats. Dependence on internet connectivity can also be a drawback, as system downtimes or slow connections may hinder productivity. Moreover, some organizations face compliance issues, as cloud providers may store data in different jurisdictions with varying legal regulations.

Although cloud-based document management systems are continuously improving, many employees still prefer using email for document sharing and communication. This preference often stems from the familiarity and simplicity of email, as well as the perceived ease of managing files directly from inboxes. Despite the benefits of cloud solutions, such as improved collaboration and organization, employees may be reluctant to adopt new systems due to the need to learn new knowledge, non-intuitive interfaces, lack of trust in central solutions, and their own established work methods that they are unwilling to discontinue.

This may also be due to the fact that, aside from group editing, email aligns with the natural way humans work with documents. A person completes their task, finishes it, and forwards the document, having control over when they finish their work and when and to whom they send the email with the document. Additionally, they can include instructions, comments, etc., related to the document in the message. However, basing document flow on email brings a number of other problems, which we described in detail in the paper [2]. In the mentioned paper, we described a collaborative system implementing email-based document exchange and proposed for that purpose proactive email attachments with a built-in workflow module. Such documents constitute both information and interaction units [3], as they can combine passive content with active services to interact with collaborators and their devices. This paper offers a detailed analysis of workflow patterns (Section 2) as realized through email communication acting as a transport layer, and introduces a comprehensive, implementable set of Document Coordination Patterns (Section 3.2) for modeling the migration paths of proactive email attachments.

The question arises, however, whether documents with an integrated workflow module are capable of handling processes of the same complexity as centralized document management systems. Our research conducted to address this issue resulted in the development of document coordination patterns. The possibility of identifying them allowed us to define the assumptions for distributed work management. Both of these developments are completely independent of the implementation of the system, which will be the transport layer and the runtime environment for documents. That is why we have described them in this paper, separately from the description of the collaborative system.

Two perspectives of modeling dependencies of the control flow of knowledge processes involving email-based activities are discussed in Section 2; they refer to both the process communication and coordination levels, with the latter involving the concept of workflow patterns introduced by Aalst et al. in [4]. The solution proposed in Section 3 draws on this concept by defining *document coordination patterns* that can be implemented in email systems with proactive documents that have the ability to implement document coordination patterns. In Section 4, a case study of the knowledge process is presented. It shows the use of document coordination patterns in a specific example and was used to validate our systems implementing the described idea.

The main contribution of this paper is the mapping of established flow patterns from the literature — such as sequential, conditional, and parallel flows — to their realization within email-based communication, which functions as a transport medium for organizational processes. Despite its informal nature, email reflects structured interactions like delegation, escalation, and feedback loops. By abstracting email exchanges into formal patterns, we can reveal latent workflows embedded in everyday com-

munication. And the primary outcome of this study is that these workflow patterns are embedded into autonomous migrating documents — digital entities that carry content, context, and logic as they navigate an organization. Such documents interpret and adapt their built-in routing based on email-derived patterns, enabling dynamic workflows almost without centralized control. The document coordination patterns presented in Section 3.2 constitute a set of ready-to-use and implementable templates that enable document migration within email-based communication and cover all the ways in which a process organizes the execution of its activities, as defined in standard workflow patterns listed in Section 2.

The paper is primarily related to three topics: loosely-coupled systems [5], email-based communication [6], and workflow patterns [4]. A detailed reference to the literature along with a discussion can be found in Section 5. In Section 6 we summarize our research.

2. Workflow patterns

The basic activities performed by workers exchanging documents by email (it can also be other transport media) involve a few simple steps: opening an email, copying its attachments to the local filesystem, processing their content with available computer applications and finally attaching the resulting document or documents to a new message sent to another worker. Throughout this time, the worker remains the only user of his/her inbox, local file system and personal device used to read and send emails. The dependencies of the control flow between such email-based activities may be, however, less obvious to collaborating workers, especially when viewed from different levels. At the process *communication level*, workers perceive collaboration through matching the send and receive operations: based on who sent the message and what documents were produced, the recipient must decide who to send the message to next. At the process *coordination level*, however, more complex patterns of organizing activities into larger decision and synchronization structures may occur. For the workers to perceive these patterns, they must understand the broader context of the activities being performed, which may not be fully comprehensible from the communication level. Based on their process understanding, workers must be able to implement the related workflow patterns by receiving and sending messages appropriately. Throughout the rest of this section we will investigate whether such patterns can be implemented when exchanging documents as email attachments, and next in Section 3, what set of patterns would suffice to implement realistic document-centric collaboration processes.

2.1. Process communication level

Let's start with the canonical classification of communication events in distributed systems that use message passing for data exchange [7]:

- *one-to-one* (1-1), when one sender sends a single message in a single action to one recipient,
- *one-to-many* (1-n), when one sender sends a single message in a single action to many recipients, and
- *many-to-one* (n-1), when many senders send independently their own (but related to each other) messages to one recipient.

Implementation of 1-1 and 1-n communication events is fairly simple for email users. Sending an email to some known recipient (1-1) requires performing just one send operation, whereas for sending the same email to multiple recipients (1-n) in one operation their addresses must be listed explicitly in the `To` or `Cc` fields of the message. Alternatively, the sender may put there an *alias* pointing to many addresses, or use an electronic *mailing list*; the two respective types of 1-n events correspond to *multicast* and *broadcast* in distributed systems.

Implementation of n-1 communication events is a little bit more complicated with emails, as the typical email client has a limited capability to get all messages from its inbox that are sent from different addresses, but are somehow related in a global context. This relation may be established by senders denoting messages with commonly agreed identifiers in the `Subject` field, so the recipient could collect all so marked messages in a separate local folder. Nevertheless, this informal technique is prone to error, as senders may fail to edit their `Subject` fields properly when preparing their messages for sending.

Another important characteristic of communication events based on email messaging is their inherent asynchronism. Sending an email message is a *buffered send* action, because the sender puts the message into a communication buffer and continues their current activity without waiting for the message to be delivered to the recipient. On the other hand, receiving the email message is a *non-blocking receive* action, as the recipient does not wait for any specific message to be delivered.

2.2. Process coordination level

In order to assess implementability of a knowledge process with email messaging and the minimum scope of the worker's comprehension of the context of executing a given activity, let's review briefly the list of workflow patterns proposed originally by Aalst et al. in [4] and elaborated further by the same team in [8]. They capture all relevant decisions and synchronization process structures

in real organizations.

2.2.1 Basic Control Flow Patterns

A process arranges the execution of its activities in a specific order, constituting its control flow. It consists of specific structures, combining their subsets in a serial and/or parallel manner:

- *Sequence*; upon completion of one activity, the next one is enabled. This corresponds to one collaborator sending a single email message to another one. No choice is involved, i.e., there is no alternative recipient for the sender.
- *Parallel Split*; a single thread splits without any condition into multiple parallel threads. This implies one collaborator sending a single email message to many collaborators.
- *Synchronization*; multiple parallel threads converge into one thread after the preceding Parallel Split. At this point no further activity may be performed until all merging threads are completed. This implies one worker receiving emails from the expected number of collaborators. The recipient should not perform any activity related to the received messages until all messages are delivered.
- *Exclusive Choice*; a single thread is redirected to another one, selected under some condition associated with this point. This implies one collaborator sending a single email message to another one, selected by the sender from a set of possible recipients.
- *Simple Merge*; one out of many threads may become active after the preceding Exclusive Choice. This implies a collaborator receiving a single email message from one of many possible senders. The recipient should not perform any further activity related to that point if no message is delivered.

2.2.2 Advanced Branching and Synchronization Patterns

This class of patterns characterizes more complex branching and merging transitions that may occur in processes:

- *Multi-Choice*; a subset of threads is selected under some conditions to be executed in parallel. This generalizes Synchronization and Exclusive Choice patterns. For email messaging, this implies that one collaborator sends a single message to a number of collaborators after selecting them from a larger set of possible recipients.
- *Synchronizing Merge*; like the Synchronization pattern, except that for multiple incoming threads only a few may actually converge into one after some preceding Multi-Choice. For email messaging, this implies that one collaborator receives emails from a certain number of collaborators belonging to the larger group. The problem for the recipient is to determine the exact number of messages that should be received. Therefore, the original Synchronizing Merge has been divided further in [8] into three more specific Structured, Local and General Synchronizing Merge patterns, which differ in determining how many threads are required to synchronize.
- *Multi-Merge*; many threads converge without synchronization, i.e., each incoming thread requires the same reaction in the activity. This pattern is useful when one collaborator is counting the number of responding collaborators upon receiving their respective emails, and performs the same activity for each one received. It resembles the execution of a single definite 'for' loop in an imperative programming language.
- *Partial Join*; an activity waits for the specific subset of all incoming threads to complete. The k first completed threads converge into one subsequent activity, while all other threads are ignored, regardless of their completion status. For email messaging, this implies one collaborator waiting for emails from the unspecified group of collaborators, who upon receiving the first few emails perform the relevant activity and ignore all subsequent ones. The Partial Join pattern has been divided further in [8] into three more variants: Structured, Cancelling and Blocking, which differ in determining what to do with ignored threads. In email systems, it is difficult to cancel a message sent but not delivered yet to its recipient's inbox or one which prevents the sender from sending another message. Therefore, only the Structured variant is feasible in email systems.
- *Generalized AND-Join*; similar to the basic Synchronization pattern in that upon completion all incoming threads converge into one thread again, but provides for the possibility that incoming multiple threads may be completed repeatedly. In such a case, threads may have to synchronize with other threads on a repetitive basis and in the correct order, as incoming threads are independent of each other. For email-based communication, this implies the worker receiving periodic emails from his/her collaborators and performing a subsequent activity for each complete set (wave) of emails received in one cycle.
- *Thread Split* and *Thread Merge*; a single outgoing thread must be activated a specific number of times, and a single incoming thread must be completed a specific number of times. For email-based communication, this implies the worker sending a series of emails to his/her collaborator, and respectively one collaborator receiving a series of emails from a

single sender, e.g. multiple blocks of a large message partitioned into smaller chunks.

2.2.3 Multiple Instances Patterns

It is often required to perform a single piece of work in parallel by multiple instances of the same activity. This is the case of Multiple Instances patterns, which in an email system may involve sending a single email in one click to a group of recipients identified by some email alias, or a named group of workers subscribing to the relevant mailing list or shared mailbox. A common feature of the latter communication event is the anonymity of recipients, including their exact email addresses and the group size. Instances of the multiplied activity should return their results as if they were produced by a single instance, therefore some external entity is required to collect, combine and send them back as one email from a single address. Alternatively, upon multiplying instances the results may be loaded back directly into the local file system. The latter, however, is beyond the capability of a typical email client and would require manual implementation, e.g. by providing links in the broadcast message to the relevant file transfer service.

2.2.4 State-based Patterns

In this class of patterns, one thread depends on the state of another thread in the process or even outside of the process, as these patterns are interdependent. It is not a standard situation in email-based communication — most often the only way for signaling state by collaborators is to exchange messages. Some additional communication channels would be necessary to implement these patterns, for example phone calls or texting. The patterns are:

- *Deferred Choice*; one out of several alternative threads is chosen, based on information which is not necessarily available when this point is reached. i.e., alternatives are offered, but the choice between them is delayed until the occurrence of some future event. This is a situation where a collaborator does not know to whom an email should be sent and waits for some additional information to proceed further.
- *Milestone*; a given activity has finished and another activity following it has not yet started. This may be a point in the process at which an email was sent by a collaborator who waits for the acknowledgment of its delivery to proceed further.
- *Interleaved Routing*; a set of activities is executed in an arbitrary order. Each activity in the set is executed exactly once and no two activities in the set can ever be active at the same time. This pattern may correspond to sending a single email with a link to an external service providing mutual exclusion access to the shared resource needed to perform the related activity by each group member one at a time. A variant of this pattern — the *Interleaved Parallel Routing* pattern imposing a constraint of partial ordering on these activities — could be implemented by the aforementioned service if subsequent calls of individual recipients are kept in a priority queue. Although the above mentioned solutions can automate mutual exclusion of specific activities related to accessing a shared resource, a more general implementation of the Interleaved Routing pattern in an email system is costly and not intuitive, as processing the same message by a group of recipients in the mutually exclusive manner requires locking all but one member out repeatedly until all of them are done with completing the given activity.

2.2.5 Cancellation and Termination Patterns

Process conclusion may be either exceptional and caused by some external entity (cancellation) or planned and implied by the internal process structure (termination). Cancellation is normally beyond the capability of any email client, as a properly sent email message cannot be eliminated by any process participant until it has been delivered to its destination address.

Note that even providing collaborators with a separate communication channel, e.g. phone calls, would not be sufficient to implement the Cancellation pattern in an email-based system, as there is no mechanism for tracing the current locations of messages being sent. Of course, the respective 'cancellation' message could be sent to all potential collaborators involved in other threads, but this may not work if some collaborators have been added to the process after starting it, and thus may not be known to the sender of the 'cancellation' message.

A Termination pattern occurs when each active thread in the process has no remaining work to do (implicit termination), or when any of its active threads reaches a distinguished end activity (explicit termination). In an email-based communication, implicit termination implies receiving a message that does not need to be forwarded to any other collaborator, whereas explicit termination requires the worker receiving such a message to send a notification (cancellation) message to all other workers involved in the process.

2.2.6 Iteration Patterns

This class of patterns includes:

- *Arbitrary Cycle*; cycles in a process having more than one entry or exit point without any structure. This pattern could be implemented with the Exclusive

Choice and Simple Merge patterns.

- *Structured Loop*; repeated execution of some process part. The loop has either a pre-test or post-test condition and its structure is single-entry single-exit — in fact a combination of the Exclusive Choice and Simple Merge patterns with a condition at the beginning or at the end of the split-join region of the process.
- *Recursion*; an activity invokes itself during execution. In an email-based system, this implies a collaborator sending a message to himself/herself. This may seem rather unusual during normal work, but this technique is used sometimes by collaborators processing a working copy of the received document in separate stages, each time accessing his/her email from a different device. Sending a message to himself/herself keeps the working copy up-to-date, without the need to transfer it between stages by using some additional media, e.g. a memory stick.

2.3. Pattern implementability issues

The set of control flow patterns presented above is summarized in Table 1 as the reference set for our *document coordination* patterns defined further in the paper. The 'sender' and 'recipient' columns indicate for each respective pattern the elementary email operations that have to be performed by collaborators in order to implement it manually. Note that apart from basic sending, reading and acknowledging message delivery, these patterns are not directly supported by any popular email client.

A question to be asked is whether operations listed in Table 1 can be implemented with the built-in functionality of active documents attached to the email message in a nondisruptive manner to the standard email exchange protocols but fully conformant with the coordination patterns discussed above.

3. Document-centric collaboration

Traditional models of collaborative work based on document exchange assume each document to be a passive object, whose content represents a certain corpus of data in terms of bytes, syntax and structure, to be processed with some dedicated tool. The worker handling the document has to select and activate the tool individually, based on its syntax and the local system resources. If, however, the exchanged documents possess execution capability, they may automatically call the appropriate software function required to work with their content and interact with their human users. Such a document-centric computing paradigm could then involve documents implemented as autonomous agents, which render and use services interchangeably with their human counterparts. If the agents additionally include a workflow component that routes them to the appropriate workers, they may be able to migrate on their own in between various execution devices operated by people.

Such properties of documents can be integrated with the email system. This was our proposal for the application of a document-agent in a specific communication model within the organization. We described the details in our paper [2]. Here, we will focus on the definition of a document-agent itself.

3.1. Proactive document-agents

The key functionality of the document-agent enabling document-centric collaboration is its ability to execute and migrate. The former refers to activation of the incoming document-agent (object) and calling its services to process the related *content* brought to the device by the agent, whereas the latter refers to the specific operations to be performed on the document to forward it to collaborators operating other devices. This concept is presented in Figure 1.

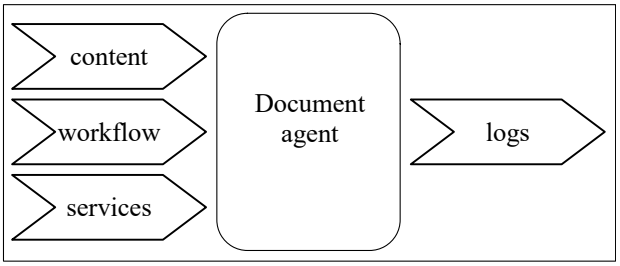


Figure 1: A document-agent concept

Services of the activated document-agent are called by the local client running on the receiving device and acting as the intermediary between the document and the worker. Three types of services are possible: an *embedded service*, implemented as a piece of code or script that may be executed directly in the local system of the receiving worker's device, a *local service*, implemented as a script that can test for availability and may also activate a specific tool installed on the worker's device, and an *external service*, implemented as a script requesting the local system to call a specified remote service.

For security reasons, execution of the aforementioned services is governed by preferences of the actual worker, who may decide whether to allow the document to activate its embedded code or just to open its content with his/her locally installed tools. Of course in either case the attached document is routinely scanned for viruses by the locally available anti-virus tool. A decision to activate a proactive document does not deviate from

Table 1: Implementability of control flow patterns in email systems

Workflow pattern	Email system			Impl.
	Event type	Sender	Recipient	
1. Sequence	1-1	A sender sends a message to a single recipient.	The recipient reads the message.	+
2. Parallel Split	1-n	A sender sends the same message to all members of the group of recipients.	Each group member reads the same message.	+
3. Synchronization	n-1	Each group member sends a message to the same recipient.	The recipient reads messages from all members of the group of senders.	+
4. Exclusive Choice	1-1	A sender selects a recipient from the group of recipients and sends him/her a message.	The recipient reads the message.	+
5. Simple Merge	1-1	A group member sends a message to a single recipient.	The recipient reads the message from one member of the group of senders.	+
6. Multi-Choice	1-n	A sender sends the same message to selected members of the group of recipients.	Each selected member of the group of recipients receives the same message.	+
7. Synchronizing Merge	n-1	Each group member sends a message to the same recipient.	The recipient reads messages from all members of the group of senders.	+
8. Multi-Merge	n-1	Some members of the group send a message to the same recipient.	The recipient reads messages from some members of the group of senders.	+
9. Partial Join	n-1	Each group member sends a message to the same recipient.	The recipient reads the first few messages from a larger group of senders and ignores all subsequent ones.	~
10. Generalized AND-Join	$(n\text{-}1)^+$	Each group member sends at least one message to the same recipient.	The recipient repeatedly reads messages from all members of the group of senders.	+
11. Thread Split/Merge	$(1\text{-}1)^+$	A sender sends a series of messages to a single recipient.	The recipient reads the series of messages.	+
12. Multiple Instances	1-n	A sender sends a message to some destination address.	The recipient forwards the message to a group of undisclosed recipients and waits for a response.	~
13. Deferred Choice	1-1	A sender waits for a destination address to send a message to.	The recipient reads the message.	+
14. Milestone	1-1	A sender waits before sending a message to some known destination until receiving another message.	The recipient reads the message.	+
15. Interleaved (Parallel) Routing	1-n	A sender sends the same message to all members of the group of recipients.	Exactly one group member reads the message one at a time.	~
16. Canceling and Termination	1-n	A sender identifies a group of recipients not known beforehand and sends them the same message.	A recipient deletes selected messages from his/her inbox.	~
17. Arbitrary Cycle	$(1\text{-}1)^+$	A group member sends a message to another member despite the fact that he/she might have gotten it before.	The recipient repeatedly reads each received message and sends it to another member of the group.	+
18. Structured Loop	$(1\text{-}1)^+$	One group member sends a message to another and waits to get it back again.	The recipient repeatedly reads each received message and sends it back to the sender.	+
19. Recursion	1-1	A sender sends a message to himself/herself.	The recipient reads the message from himself/herself.	+

Note:'+' in full, '~' in part (additional functionality of the email server required)

the typical situation when the recipient of a message has to decide whether to open a plain (passive) document attached to it. Usually, when he/she is sure that the document comes from a trusted source, the decision is to open the attachment. In the solution proposed in the paper, we expanded the range of possible decisions not only based on the personal preferences of the receiving worker, but also on the current location of the device on which the document will be opened, the device performance characteristics, the security parameters of its system software, the quality and security of the available network connection and several other attributes, whose values define the document *execution context*. Moreover, values of these attributes may be negotiated automatically by the document and its receiving worker's device. A preferences file, prepared by the worker, enables our smart email client to choose the appropriate sets of negotiable attribute values called *modes*: a *private mode* indicates what types of document computations would not be possible (local services/tools are preferred and no embedded services are allowed), a *travel mode* indicates that access to available networks is possible but unstable or not recommended for security reasons (thus external services are not allowed), a *business mode* assumes that documents come from the corporate, trusted network from inside of the worker's organization (both embedded and external services are allowed), an *airplane mode* with no access to any network (external services are not possible and the received document can further be processed only off-line) also exists, and so on.

Mobility of the document-agent on the other hand could utilize the whole range of data communication protocols used on the Internet for sending and retrieving email messages, including the lower layer TCP/IP or HTTP/HTTPS data streaming protocols, as well as protocols of higher layers, such as SMTP, POP3 and IMAP.

The *workflow* path is also an integral part of the document-agent and can be modified during its flow. This eliminates the need to know the entire process in advance. The details of document workflow in various business scenarios are presented in the subsection below. From the perspective of organizational management, it is crucial that every document workflow leaves a trace in the form of an event log. This enables the application of process mining techniques [9], which are discussed in more detail in Section 5.

3.2. Document Coordination Patterns

A knowledge process represents a logical sequence of activities to be performed by workers involved in it. Its specification is usually composed of several reusable patterns to enforce a certain choreographic order in document flows. Each pattern of that kind represents a specific rationale for coordinating document transfer between activities of various types and purposes. In this paper, we distinguish three classes of such patterns. One class includes *distributed state patterns*, when the document transitions depend only on the execution state of a single activity performed at some location in a system. Another class includes *coupled state patterns*, when the document transitions depend on execution states of two or more activities performed simultaneously by other workers at other locations in the system. Finally, *embedded state patterns* involve block activities implemented as subflows. These classes are specified further with BPMN diagrams shown in Figures 3a–5b. A subset of standard BPMN widgets [10] used in these diagrams is listed in Figure 2.

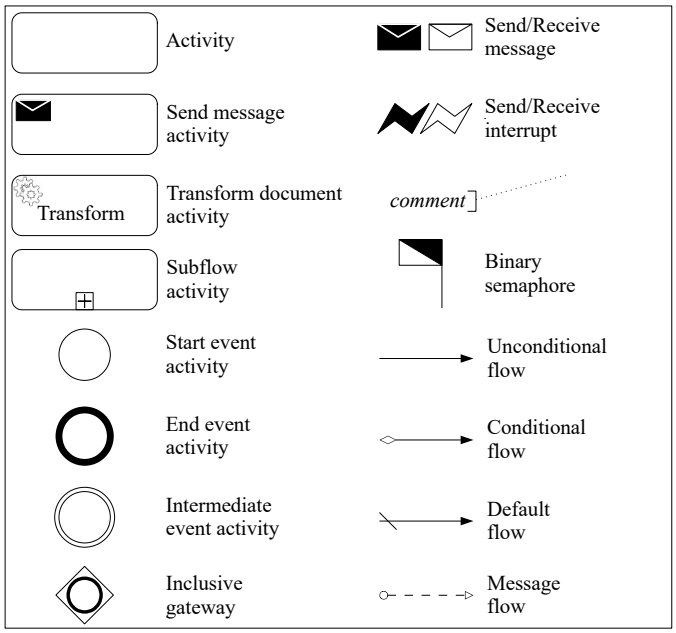


Figure 2: BPMN elements used to specify document coordination patterns

The specific functionality brought by a proactive document-agent to aid the receiving client is represented by a generic *transform* document activity. The *binary semaphore* symbol has been added to denote a semaphore enabling a document to wait for an asynchronous external event. The *inclusive gateway* widget, which allows one, several or all options to be taken respectively of the conditions controlling them is used to specify all situations when specific threads of the pattern diverge or converge.

3.2.1 Distributed State Patterns

The four patterns described below refer to situations in which the successive activity to be performed can be determined based solely on data brought by the received document to the execution device.

Sequencer involves a collaborator who upon completion of the activity sends to another collaborator one or more resulting documents in a sequence (see Figure 3a). Depending on the implementation of the Transform activity, this could range from manual operations performed by a worker to fully automated execution of the internal functionality of the document. Each outgoing document is labeled with a serial number based on the value of variable a counter embedded in its body. The total number of documents produced by Transform is not known in advance to workers KW1 and KW2 and the last document should be marked appropriately with the sentinel serial number. If for the last produced component counter>1, the document sequencer patterns correspond to the Thread Split/Merge pattern listed in row 11 of Table 1; otherwise it is just the Sequence pattern listed in row 1 of Table 1.

Even in this most basic pattern, the processing of the received document may be a serious job demand for the recipient, as well as for the sender, who often has to explain to the recipient what to do with the attached document. A proactive document can buffer that job demand by automatically checking for the availability of the appropriate local application tool to process the document content or by providing a dedicated service enabling that.

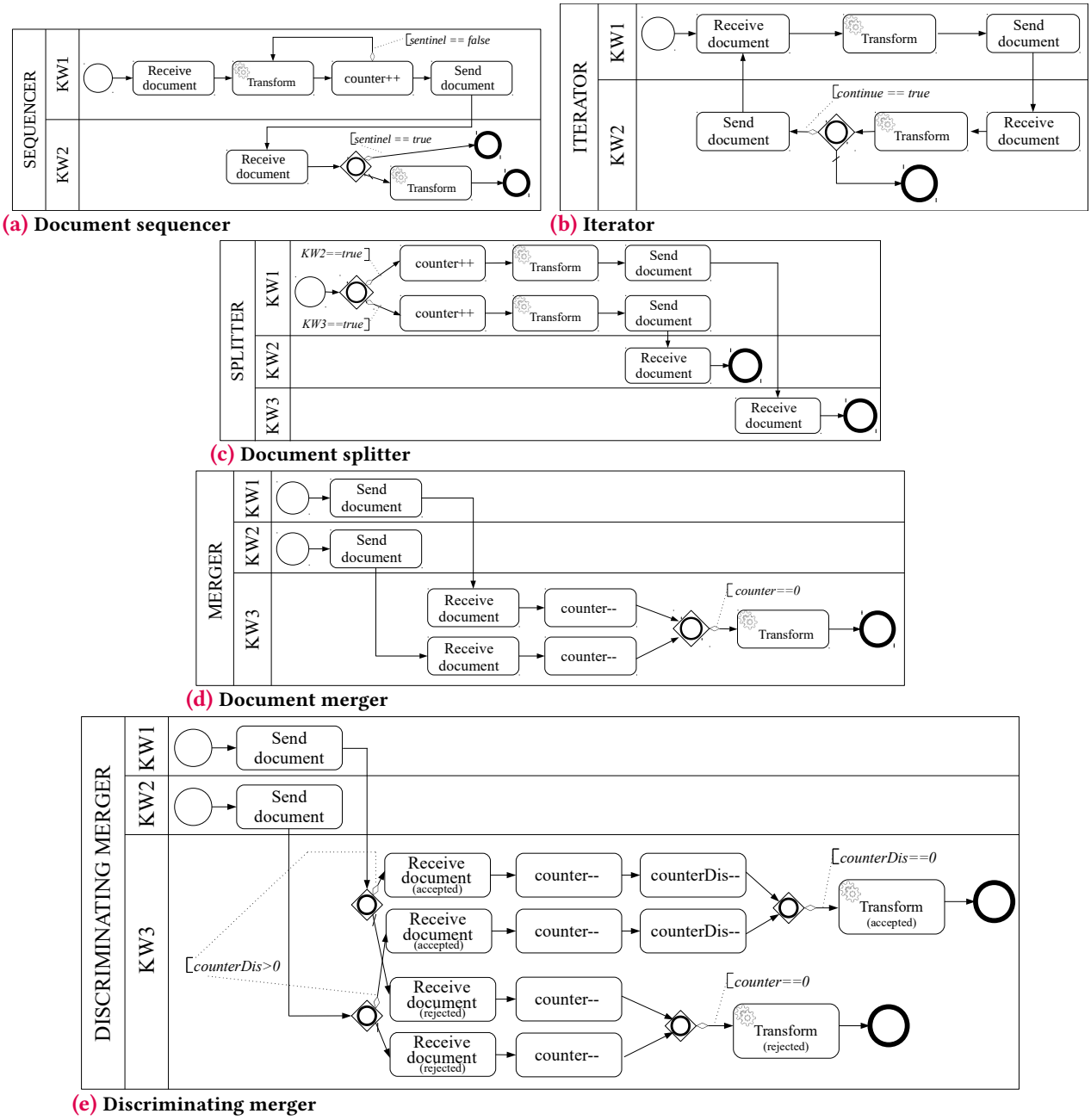
An active document is also able to calculate the urgency level of activity based on deadline or other process attributes rather than the individual sender's indications and can change this level over time.

Splitter may be *cloning* or *decomposing* (see Figure 3c). A cloning splitter creates multiple copies of each single received document, whereas its decomposing counterpart partitions the document into disjointed fragments. The resulting documents, either copies or fragments, are sent next to the respective collaborator specified in the received document body. The Transform activity in the cloning splitter merely copies a received document, while in the decomposing splitter it may perform more complex operations, as in the sequencer pattern described above.

Another distinction of this pattern is whether the number of receiving workers is constant or must be determined dynamically upon splitting of the received document. That number is required to properly merge all arriving documents and should be stored in the body of each outgoing document, as each one may win a possible race to the execution device where merging shall be performed. If all workers specified in the received document body are intended to receive the outgoing documents, the value of the counter is copied to each one of them accordingly and no gateway conditions or increment operations at each outgoing thread are required. This is the case of the Parallel Split pattern listed in row 2 of Table 1; note that neither gateway conditions nor incrementation activities at each outgoing thread are needed in this case. Conversely, if the number of receiving workers is determined dynamically upon splitting the received document, the increment operation counter++ and calculation of gateway conditions are performed by the respective Transform activity — corresponding to the Multi-Choice pattern listed in row 6 of Table 1. If only one condition is true each time the gateway is passed, then the document splitter corresponds to the Exclusive Choice pattern listed in row 4 of Table 1.

Sending a document to many recipients by email is easy (maybe sometimes too easy) and may often lead to neglect and abuse of the sender, such as sending emails with missing attachments, an ill-considered number of recipients or placing too many topics in one message — each of which may require separate processing. Any of these is a job demand for the recipient, which a proactive document can buffer easily: no missing attachments, a precisely determined number of recipients in its migration path and the ability to divide the message into independent threads. That would certainly be much easier to comprehend and process.

Merger complements the document splitter pattern (see Figure 3d). It may involve any functionality in its Transform activity brought to the receiving client, depending on whether a preceding splitter was cloning or decomposing. It may be a simple concatenation of document fragments or a more complex combination of multiple document fragments [11]. The value of the counter variable, set to the total number of documents sent by some preceding splitter, is copied by the receiving client from the first delivered document and decremented subsequently by operation counter-- with each next arriving one. Each received document component may be immediately processed and sent further, or saved in a buffer folder until all expected document components are received. Note that in either case the merger pattern is terminated when the value of counter reaches 0. The former corresponds to the Multi-Merge pattern listed in row 8 of Table 1, while the latter does so to both the Synchronization and Synchronizing Merge patterns, listed respectively in rows 3 and 7 Table 1. Merging document components stored in a buffer folder may involve various functionalities, depending on whether the preceding splitter was the cloning or decomposing type. It may be as simple as the concatenation of chunks of text, or quite complex if sophisticated content merging algorithms are required. If the arriving document brings counter=1, the document merger corresponds to the Simple Merge pattern listed in row 5 of Table 1. Parts of documents from various processes can be received in the inbox of one collaborator. However, each attached proactive document can have a unique ID, so the email



(a) Document sequencer **(b)** Iterator

(c) Document splitter

(d) Document merger

(e) Discriminating merger

Figure 3: Distributed State Patterns

system would know exactly which documents from the inbox are parts of the document to be merged. This corresponds to the Generalized AND-Join pattern listed in row 10 of Table 1.

In email systems, the majority of the effort to adopt this pattern lies with the recipient. A proactive document can indicate how many and exactly which attachments to various emails need to be integrated. Moreover, its services can aid the recipient in integrating them correctly. The basic services can collect cloned documents in one logical folder or automatically concatenate a previously decomposed document. More advanced services can perform automatic integration using various dedicated intelligent algorithms proposed by D'Angelo et al. in [12, 13]

Discriminating merger pattern shown in Figure 3e is a special case of the document merger. It waits for a number of documents before performing the subsequent `Transform` block activity. The respective value of this number is specified by the variable `counterDis`. The value of `counter` external attribute specifies the total number of all expected documents. Gateway condition `counterDis>0` determines for each received document if it can be accepted, while the condition `counter==0` is used to terminate the pattern. If for the first received document (upon starting the pattern) `counterDis<counter`, the discriminating merger corresponds to the Partial Join pattern listed in row 9 of Table 1; otherwise it corresponds to the Synchronizing Merge pattern listed in row 7 there.

In email messaging, this pattern is performed by the recipient, who needs to look up the inbox to select all relevant messages. An active document can make its handling smart email client automatically selects messages in the inbox; the condition for the pattern termination would be then reading all the required messages or exceeding a certain time limit.

Iterator enables repeated execution of the sequencer pattern controlled by the loop termination condition calculated by the respective document `Transform` activity (see Figure 3b). This is a powerful pattern for creating more complex patterns listed in Table 1. One of them is the Arbitrary Cycle pattern listed in row 17. It may be constructed by iterating a combination of the Exclusive Choice, Sequence and Simple Merge patterns listed respectively in rows 4, 1 and 5. Another is the Structured Loop pattern listed in row 18 with pre- or post-test conditions. The gateway in the iterator pattern can also be placed on the KW1 lane after the `Transform` activity. The value of the `continue` flag is set by `Transform` based on the predetermined loop counter or the ad-hoc worker's decision, making this pattern useful in many real situations. The Recursion pattern listed in row 19 corresponds to the situation where KW1 and KW2 lanes specify the same worker using different devices.

An iteration without an a priori determined counter is commonly used by electronic mail users. The sender can explicitly create loops by choosing the "reply" option or implicitly by forwarding a document to a recipient who has already processed it. Such loops can be created by the sender many times, without any restrictions enforcing their structure. An active document does not limit the possibility of creating ad-hoc loops of any structure (the Arbitrary Cycle pattern in row 17 of Table 1), but it can impose this structure if the process requires it. Examples of the latter include: after completing certain activities the document always returns to the worker responsible for the specific part of the process, a copy of the document automatically returns to the responsible worker, or the document has to be checked a specified number of times (e.g. after reviews or translations).

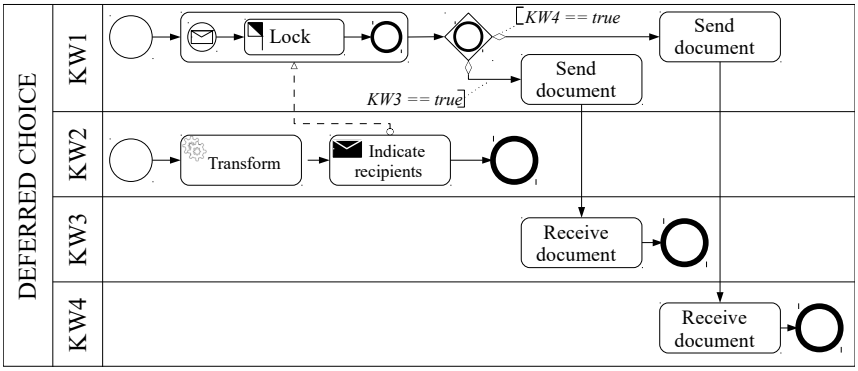
3.2.2 Coupled State Patterns

As indicated before, a principal source of information to determine the next activity to be performed is the arriving document itself. However, certain states reported by other devices may also be needed to make a decision. This is the case in coupled state patterns, which involve the notion of *asynchronous signals*, sent to the performer of a given activity by other performers, who participate in the same process and perform their respective activities in parallel to the former one. Three patterns of this kind are proposed below.

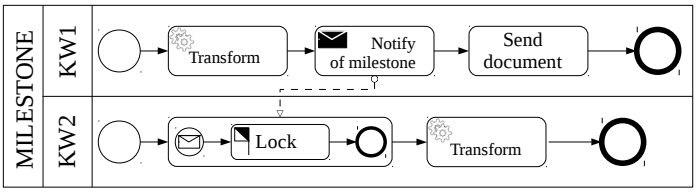
Deferred choice is used when sending a given document which has to be postponed until the respective execution device receives information as to whom it should be sent to (see Figure 4a). This corresponds directly to the Deferred choice pattern listed in row 13 of Table 1. The arriving document brings `Lock==closed` as the initial value of its semaphore, so if worker KW1 is ready to send a document before a signal from KW2 has been received, it is put aside and waits to enable the execution device to handle other arriving documents. Upon receiving a signal from worker KW2, processing of the document is resumed. If the signal has been received by KW1 before receiving a document, the relevant signal notification should be stored by the client internally to enable immediate transfer as soon as the document has been received and processed. The signal involved in deferred choice carries additional data, which are the email addresses of workers to whom the pending document will be sent; in Figure 4a they are workers KW3 and KW4.

Milestone is the simplified version of the deferred choice pattern, in which the related signal does not carry any specific data associated with it and whose only purpose is to block some activity of one collaborator by another (see Figure 4b). It corresponds directly to the Milestone pattern listed in row 14 of Table 1. Processing of the document arriving to worker KW2 may be postponed until a certain "milestone" activity has been completed by worker KW1. Again, the initial setting of the semaphore `Lock==closed`, brought by the arriving document to KW2, prevents it from proceeding any further until the signal notifying it of completion of the milestone activity has been received from worker KW1. Conversely, if a signal from KW1 has been received before document delivery to KW2, its notification should be stored by the client internally to enable immediate transfer as soon as the document is received and processed. A combination of the document splitter, merger, sequencers (if any) and a set of milestone patterns would suffice to implement the Interleaved Parallel Routing pattern listed in row 15 of Table 1. A document is sent to all workers of the group with the initial setting of the semaphore `Lock==closed`, whereas the unlocking signal can be sent by the proactive document to all members of the group to unlock their activities automatically.

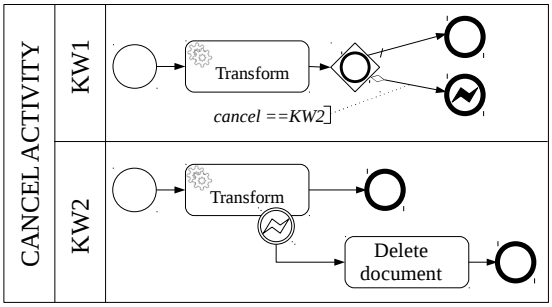
Cancel activity would be needed when the performer of one activity decides that some other activity of another performer should be canceled. Unfortunately, implementation of Canceling and Termination patterns listed jointly



(a) **Deferred choice**



(b) **Milestone**



(c) **Cancel activity**

Figure 4: Coupled State Patterns

in row 16 of Table 1 is not straightforward with email messaging for its inherent asynchronism.

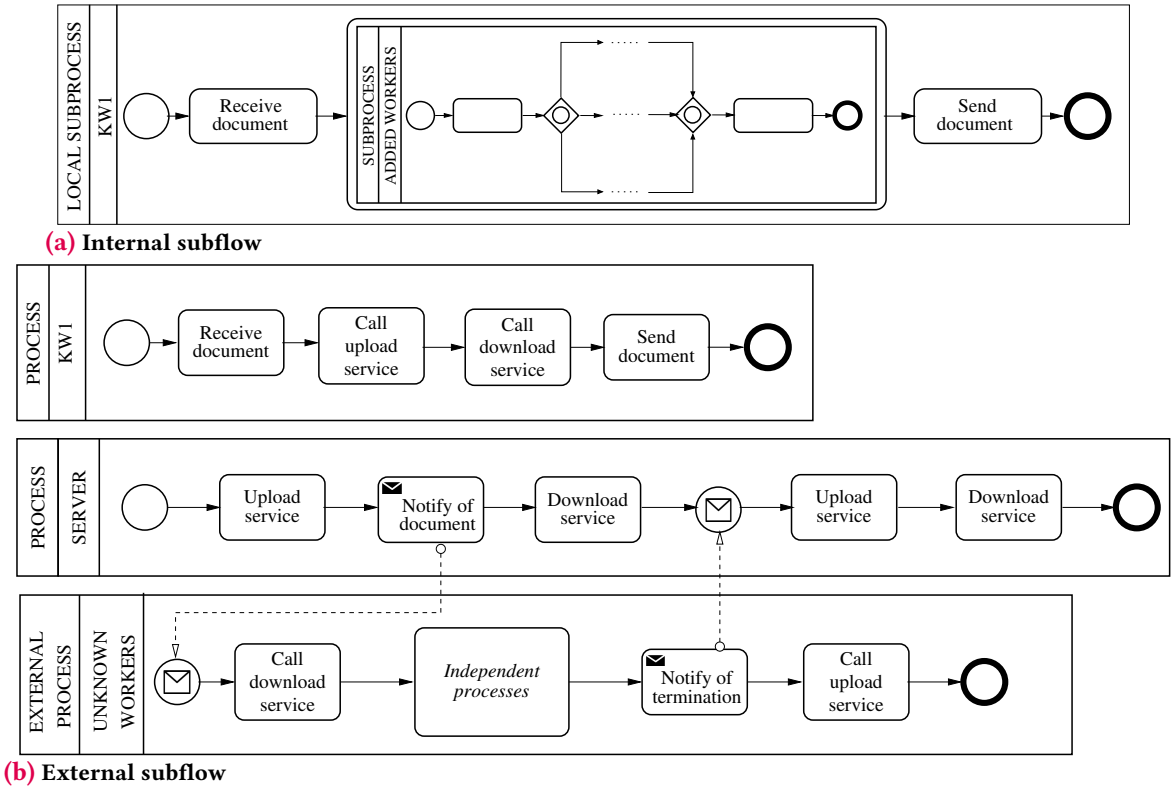
Consider the *cancel activity* pattern shown in Figure 4c. Three cases are possible when the cancellation signal or out-of-bound cancellation message has to be sent. One is when a document has not yet been delivered to another collaborator, so the cancellation signal should be registered internally by the receiving device to enable immediate deletion of the indicated document as soon as it has been delivered. Another is when the document is currently being processed, hence its deletion may be problematic if the performer has advanced to the point where some resulting documents have already been sent. Note that no semaphore is needed there, as a decision on canceling the activity and deleting the related document is immediate for the execution device. Finally, a document may have already been processed and sent to another worker, so canceling that particular activity would fail. Since there is no guarantee that the activity can be effectively canceled (and its related document deleted), a cancellation message should be sent to all performers indicated in the migration path. It would however implement a cancel case pattern rather than cancel activity or cancel region patterns defined originally in [8] — a feasible but expensive solution. Note that the cancellation signal should be sent to all activities in the relevant process and its subprocesses, some of which may be hard to identify if new workers have been added to the process after starting it. The pattern specified in Figure 4c also corresponds to the explicit termination pattern defined in [8], where the canceling signal is sent when the process reaches a specific end node along any of its active threads.

3.2.3 Embedded State Patterns

Sometimes the process thread may include a set of activities performed as a subprocess, with workers not specified originally in the document body. Two patterns of this kind are proposed below. They both incorporate the Multiple Instances pattern listed in row 12 of Table 1.

Internal subflow implies that the performer of the activity is authorized to extend the original migration path of a document by adding new activities of his/her choosing. Such an extension would constitute an internal subflow (see Figure 5a). Neither its structure nor the identity of added performers are assumed to be known in advance to other collaborators in the original process. This pattern makes it possible to modify the process during its execution. A graph of new activities is added as a subflow to the parent activity, which allows the process to be expanded, and saves the already defined activities structure.

External subflow may be initiated by the performer of the activity who calls the external service to do some work involving a separate subprocess structure (see Figure 5b). The identity of the thus "subcontracted" performers are not known to the performer and other collabora-



(a) Internal subflow

(b) External subflow

Figure 5: Embedded State Patterns

tors. Separation between the main process and the external subprocess is provided by an intermediary server, to which the relevant document is uploaded for extra processing and later downloaded upon completion. At the other side of the intermediary server, the subcontracted external process downloads that document for processing and upon completion, uploads it back to the intermediary server. Upload and download service calls are blocking and the external process may be implemented in an arbitrary manner, including manual operations. This allows the email system to be merged with many other systems operating on documents, e.g. cloud-based group editing systems [13].

4. Class Roster Case Study

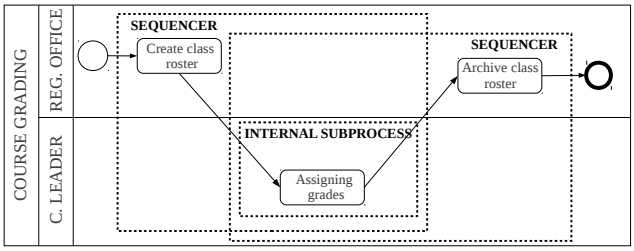
Several proof-of-concept prototypes of smart email clients were implemented by us to handle proactive attachments, that concept is presented in Figure **??**.

They were tested using the class roster application described below. Although this application implements relatively uncomplicated decision processes, related to grading student work in a typical academic setting, it is sufficient to demonstrate the non-algorithmic flavor of an interactive computer system. It also demonstrates the use of document coordination patterns introduced in Subsection 3.2.

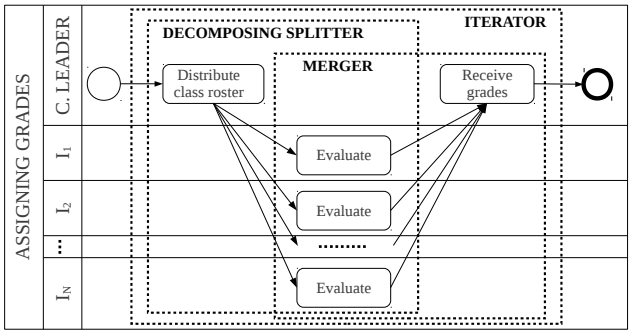
4.1. Class Roster Case Study

Consider the course grading knowledge process in which instructors judge the academic quality of students' work and assign grades as symbols of their evaluation. In doing so, they collaborate towards assigning a final (semester) grade for each student registered in the course. A Registrar's Office (RO) is the course grade roster document originator, whose collaborator is a Course Leader (CL). The CL runs his/her subprocess to successively collect credits from other instructors during all semester weeks. Structure and implementation of that subprocess are irrelevant to the RO. While the RO may use an online grading system for one-time roster submission and approval, the CL is free to implement the collection of credits on his/her own, in our case with email messages. The dynamics of the grading process involve *scheduled* events, such as assessment of lab assignments and *unscheduled* events, such as occasional grade corrections, which can both be readily implemented with the document coordination patterns defined before.

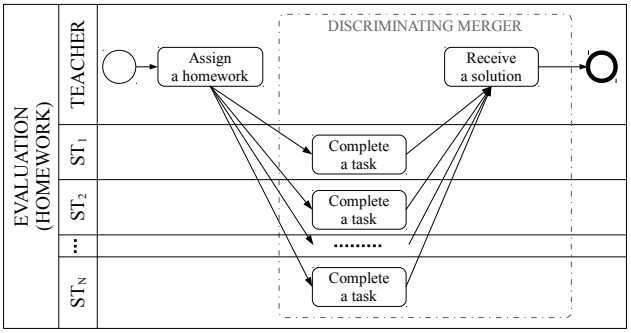
Figure 6a specifies the course management process from the perspective of the RO, to which the CL is the only collaborator. Note two *sequencer patterns* — one for sending the roster only with student names and IDs to the CL at the beginning of the semester and another for receiving it with final grades at the end of the semester. Details on collecting scores that correspond to assignments listed in



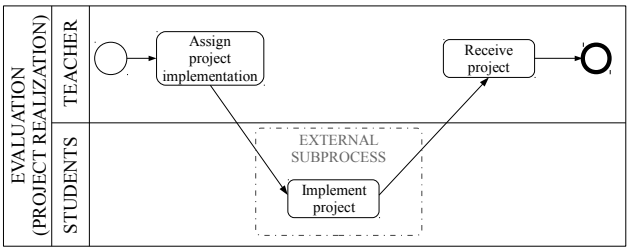
(a) The main grading process



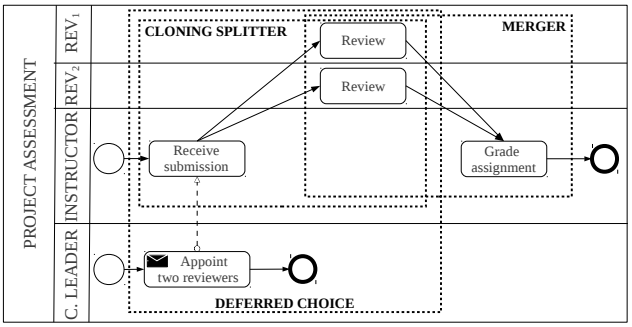
(b) Course leader subprocess



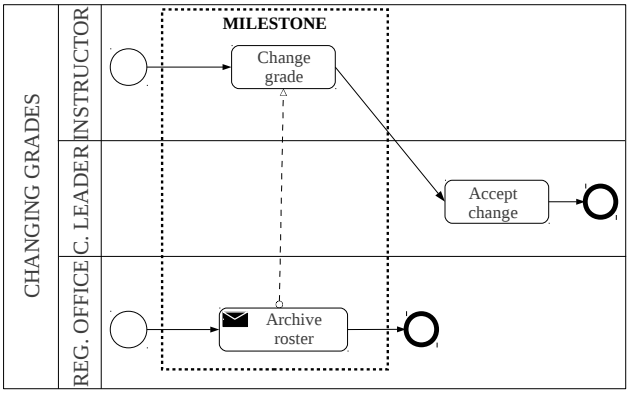
(c) Homework competition subprocess



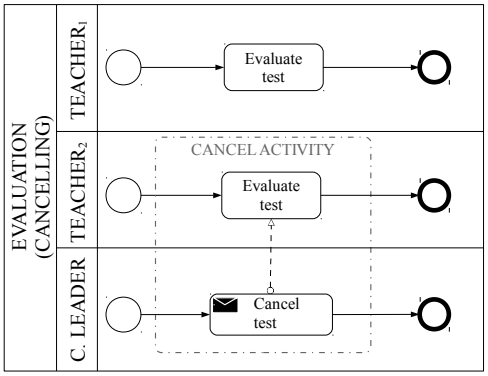
(d) Out-of-class project subprocess



(e) Project assessment subprocess



(f) Grade changing subprocess



(g) Handling of academic misconduct

Figure 6: Document migration patterns in a course management process

the course schedule, in particular their weights and timing, are irrelevant to the RO, who perceives grading as a single activity performed by the CL. Alternatively, the CL can implement that activity using the *internal subprocess* pattern to handle all relevant types of classes selected from the set of lectures, tutorials, seminars, projects or labs, engaging instructors to run them and monitor deadlines for each respective assignment. A generic structure of that subprocess is shown in Figure 6b.

The class roster may be partitioned at the CL's execution device into complementary group rosters, each one for the respective instructor and sent out according to the *decomposing splitter* pattern. Each instructor evaluates his/her group assignments and sends grades back to

the CL for merging, in accordance with the *merger* pattern. Group rosters, governed by the *iterator* pattern, return on their own to the CL, after collecting credits for each particular assignment.

Evaluating assignments by individual instructors ends by entering scores in the respective group rosters, but may also involve more complex internal subprocesses. For example, Figure 6c specifies a situation where students compete when submitting homework solutions to their instructor. If only a few first correct solutions are to be awarded a grade, the discriminating merger pattern should be used. Another situation is shown in Figure 6d, when students participate in an out-of-class project, assessed by workers external to the university, e.g. company employees approved by the CL. In such a case, the CL may not know details of the external evaluation process, which is the case in the external subprocess pattern.

If the paperwork prepared by students requires independent reviews, the CL may appoint reviewers to assess student's work in a formal way. This is the case in the *deferred choice* pattern, as shown in Figure 6e. Each reviewer receives from the instructor a copy of the project documentation, as provided by the *cloning splitter* pattern. Upon receiving the reviews, the instructor makes a decision on the final score for the project work and merges the reviews to provide justification for it. If during the grading process some grade requires a correction (for example a grading error has been detected by the instructor or the student's query has been accepted), the *milestone pattern* may be used as shown in Figure 6f — under the condition that corrections are still possible because the deadline set by the RO for returning the class roster has not yet elapsed. Some grades may also be directly revoked by the CL, if a case of academic misconduct like cheating or plagiarism was revealed. This can be handled by the cancel activity pattern, as shown in Figure 6g.

Although the aforementioned process has algorithmic elements, such as calculating for each student the final grade based on a formula combining all received scores with their corresponding weights, for the most part it remains non-algorithmic. This is because assessment of the student's work involves human decisions based on understanding what has been done. Examples include evaluation of the individual contribution of a student member to a team project, assessment of oral presentations, report reviews, etc. Moreover, the grading process must be able to handle any incident that may occur during the course. Due to the unexpectedness and large volume of events, document workflow designers must be able to predict and implement mechanisms handling situations not known to them in advance. This necessitates in turn provisions for dynamic modification of a document migration path *after* the process has started. Authorized knowledge workers may then add or remove arbitrary process activities when interpreting procedures in accordance with their experience, common sense, ethics, and other non-algorithmic aspects of decision making.

5. Discussion

Workflow patterns [14] are recurring structures that represent common control-flow scenarios in business processes. They serve as a conceptual framework to analyze and compare the capabilities of various process modeling approaches and business process management tools. In the context of process mining, workflow patterns play a crucial role in understanding and discovering the underlying structures of processes from event logs.

Process mining introduced by Aalst in [15] is a data-driven technique used to analyze, discover, monitor, and improve business processes based on event logs recorded by information systems. It bridges the gap between traditional process modeling and real-time process execution by extracting insights from actual data.

There are three main types of process mining: discovery, which generates process models from raw event logs; conformance checking, which compares real process execution with predefined models to detect deviations; and enhancement, which optimizes processes by incorporating additional insights.

By leveraging process mining, organizations can gain transparency into their workflows, identify inefficiencies, and make data-driven decisions to improve operational performance.

Process mining involves extracting process models from event logs to gain insights into actual process executions. By identifying workflow patterns within these logs, analysts can recognize standard behaviors and deviations, facilitating the discovery of accurate and comprehensible process models. Porouhan et al. introduced the importance of recognizing such patterns to bridge the gap between real-time workflow processes and their perceived models [16].

Moreover, workflow patterns assist in addressing data quality issues in event logs. The paper [17], Suriadi et al. discussed how patterns can be utilized to identify and manage imperfections in event data, ensuring more reliable process mining outcomes. Thus, the application of workflow patterns in process mining enhances the discovery and analysis of business processes by providing a structured approach to identify common behaviors and address data imperfections. By analogy, document coordination patterns (see Section 3.2), which are directly derived from workflow patterns (Section 2), have a similar application.

Moreover, as argued in Section 2, collaborative work based on document exchange involves communication of content, meaning effective exchange of portions of information between workers, and coordination of activities performed by them in the knowledge process. In a more general sense they may be viewed as *knowledge transactions*, defined as transportation of *knowledge objects* between two or more communicating workers [18]. This in turn affects the dependence of the conduct of the related knowledge process on various interconnected knowledge-intensive decisions made by process members; they concern the knowledge objects and transactions alike [19]. The solution proposed in this paper enables convenient separation of these two concerns: knowledge about the formal process synchronization structure remains in the system, whereas the knowledge on how to perform the particular activity and make informed decisions by interacting with the relevant documents is left to the worker. As illustrated by the running example in Section 4, this can introduce a reasonable level of process management to document-agents.

To verify the above claim let us refer to the spectrum of knowledge-intensive process classes proposed by Ciccio et al. [20]. Spontaneous exchange of e.g. emails with attached documents and without any coordination of knowledge transactions results in *unstructured processes*. Although such processes can exhibit a great level of flexibility to collaborators, the predictability of the results and repeatability of the order of execution of individual activities could be problematic, as decisions of process participants on these issues are solely based on their experience and implicit knowledge. If, however, messages with attached documents could bring the necessary information to guide or instruct collaborators on how to control their knowledge transactions, then implemented processes would exhibit a *loosely structured behavior*. Their scope would then be implicitly framed by indicating to the worker the undesired behavior. This is the case of variables carried by our documents, such as `counter` or `lock` specified in Section 3, whose values affect the execution of knowledge transactions implemented by the respective distributed state and coupled state coordination patterns. If all knowledge transactions are implemented with a fixed set of patterns, then according to [20] processes can be considered *unstructured with pre-defined segments*. The overall process logic would not have to be explicitly defined, but its structured, pre-defined fragments should introduce rules governing at least its essential knowledge transactions. It is worth noting that such rules can be implemented automatically for our documents. If collaborators need to make ad-hoc changes to the process at runtime they can add subprocesses whenever the actual course of action needs to deviate or expand beyond the particular activity; they can do so with our document-agents by using respectively the cancel activity pattern (Figure 4c) or any of the subflow patterns (Figures 5a and 5b). Thus according to [20], our document-agent-based process would become *structured with ad hoc exceptions*. Finally, with a migration path built in the document body as proposed in this paper, the relevant process would become structured — with all possible options and decisions that can be made during process enactment captured in a process model defined a priori; they can be repeatedly instantiated in a predictable and controlled manner.

Given the above it can be seen that proactive document-agents provide sufficient flexibility in modeling the whole spectrum of document-centric processes. In particular, the structured with ad-hoc exceptions class of processes is interesting, as due to the duality of our proactive document, recipients may access/open the document content directly with the local tool of their choice or let the document-agents service to point the proper tool; they can also modify the document's migration path by adding subprocesses or modifying workers/performers assigned to specific activities. In this way, both anticipated or unanticipated exceptions can be handled dynamically.

According to the criteria defined by Steinau et al. [21], the document-centric approach proposed in this paper conforms to the data-centric paradigm, whose common feature is that only the availability of specific data objects (documents) drives process execution. It involves two modeling aspects: *behavior* to describe how data values are acquired by a data object (document) to perform the activity, and *interactions* to describe how data objects (documents) communicate with one another during the process enactment. Two principal concepts of interaction models have been proposed in this regard in the world literature.

The first one assumes separate modeling of interactions and behavior with different notations. A representative of this concept is the *object-aware approach* (e.g., the PHILharmonicFlows framework [22]), where the behavior (lifecycle) of an object is described as a *micro-process* and interactions among different objects are described as a *macro-process*. Both are addressed by document coordination patterns specified in Figures 3a–5b.

The second concept encapsulates into a conceptual artifact object both information and lifecycle models. The information model includes both the application data (called data attributes) and process-relevant data (called status attributes). A representative of this concept is the *artifact-centric approach* (e.g., the BizArtifact framework [23]), which uses the Guard-Stage-Milestone (GSM) notation to define in a declarative manner the lifecycle of artifacts and changes to the information model that are caused by interactions with other artifacts [24].

GSM defines stages that group individual activities with *guards* and *milestones* representing entry conditions to a stage and their respective completion points in the specified artifact lifecycle. This metaphor may also be used to specify the business process based on proactive document-agents.

Based on the above two representatives of business modeling frameworks, it may be seen that mapping of their underlying models towards our solution is possible when implementing various process models with the document-agents exchange.

6. Summary

The document coordination patterns presented in this paper can contribute to the advancement of multiple domains of research and technological development, particularly in the fields of process mining, document management systems, and electronic communication.

Firstly, document coordination patterns can serve as a foundational input for process mining studies, enabling the systematic analysis of workflow execution and identifying deviations, inefficiencies, and optimization opportunities. By leveraging structured document coordination patterns, process mining algorithms can more effectively reconstruct process models from event logs, leading to enhanced accuracy in business process discovery and conformance checking in the field of document flow management.

Secondly, these patterns contribute to the design and development of document management systems by providing standardized frameworks for handling document workflows. By integrating document coordination patterns into workflow automation, organizations can achieve more structured and efficient document routing and access management, thereby improving overall operational efficiency and compliance with organizational policies.

Moreover, document coordination patterns can significantly impact the evolution of email systems and similar communication platforms. Given that email remains the dominant medium for document exchange, incorporating structured document coordination patterns into email systems could enhance functionalities such as automated categorization, knowledgeable routing, and collaborative editing. These advancements would bridge the gap between traditional email-based document handling and modern workflow automation solutions.

Finally, document coordination patterns offer a robust foundation for further research into the advancement of electronic documents and collaborative document processing. As the digital transformation continues to reshape document-centric workflows, exploring advanced document pattern applications such as smart contracts, blockchain-based document verification, and AI-driven document processing can lead to more intelligent and adaptive document ecosystems. Additionally, fostering interdisciplinary collaboration on document patterns can facilitate the development of innovative frameworks for knowledge sharing and workflow standardization across various industries.

In summary, document coordination patterns not only provide valuable insights for process mining but also enhance document management systems, improve electronic communication tools, and pave the way for future innovations in electronic document processing and collaboration.

Acknowledgements

The research was supported in part by project "Cloud Artificial Intelligence Service Engineering (CAISE) platform to create universal and smart services for various application areas", No. KPOD.05.10-IW.10-0005/24, as part of the European IPCEI-CIS program, financed by NRRP (National Recovery and Resilience Plan) funds.

References

[1] A. Azmir and L. Wijayanti, "Cloud computing opportunities and challenges in electronic document management," *Record and Library Journal*, vol. 8, pp. 248–258, 12 2022.

[2] M. Godlewska and B. Wiszniewski, "Serverless-edge computing with mobile documents agents," *TASK Quarterly*, 2025.

[3] R. J. Glushko and T. McGrath, *Document Engineering - Analyzing and Designing Documents for Business Informatics and Web Services.* MIT Press, 2008.

[4] W. M. van der Aalst, N. Russell, A. H. ter Hofstede, and N. Mulyar, "Workflow control-flow patterns: A revised view," Tech. Rep. BPM-06-22, BPM Center, `http://bpmcenter.org/wp-content/uploads/reports/2006/BPM-06-22.pdf`, 2006.

[5] L. Arango-Vasquez and M. Gentilin, "Organizational couplings: A literature review," *Innovar*, vol. 31, pp. 151–168, 03 2021.

[6] M. Fullman, T. Jackson, P. Chamakiotis, and E. Russell, "Getting on top of work-email: A systematic review of 25years of research to understand effective work-email activity," *Journal of Occupational and Organizational Psychology*, vol. 97(1), p. 74–103, 2024.

[7] H. Krawczyk and B. Wiszniewski, *Analysis and Testing of Distributed Software Applications.* John Wiley & Sons, Inc., 1998.

[8] N. Russell, W. M. van der Aalst, and A. H. ter Hofstede, *Workflow Patterns: The Definitive Guide.* Cambridge, MA, USA: MIT Press, 2016.

[9] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[10] OMG, "Business Process Model and Notation (BPMN), Version

2.0," Tech. Rep. formal/2011-01-03, Object Management Group, http://www.omg.org/spec/BPMN/2.0/PDF, 2011.

[11] J. Challenger, P. Dantzig, A. Iyengar, and K. Witting, "A fragment-based approach for efficiently creating dynamic Web content," *ACM Trans. Internet Technol.*, vol. 5, pp. 359–389, May 2005.

[12] G. D'Angelo, A. Di Iorio, and S. Zacchiroli, "Spacetime characterization of real-time collaborative editing," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, pp. 41:1–41:19, Nov. 2018.

[13] Y. Sun, D. Lambert, M. Uchida, and N. Remy, "Collaboration in the cloud at Google," in *Proc. 2014 ACM Conference on Web Science*, WebSci '14, (New York, NY, USA), pp. 239–240, ACM, 2014.

[14] N. Russell, A. Ter, D. Edmond, and W. Aalst, "Workflow data patterns: Identification, representation and tool support," vol. 3716, 10 2005.

[15] W. van der Aalst, *Process Mining: Data Science in Action*. Springer Publishing Company, Incorporated, 2nd ed., 2016.

[16] P. Porouhan, N. Jongsawat, and W. Premchaiswadi, "Workflow mining: Discovering process patterns data analysis from mxml logs," in *2013 Eleventh International Conference on ICT and Knowledge Engineering*, pp. 1–8, 2013.

[17] S. Suriadi, R. Andrews, A. ter Hofstede, and M. Wynn, "Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs," *Information Systems*, vol. 64, pp. 132–150, 2017.

[18] P. Dalmaris, E. Tsui, B. Hall, and B. Smith, "A framework for the improvement of knowledge-intensive business processes," *Business Proc. Manag. Journal*, vol. 13, no. 2, pp. 279–305, 2007.

[19] R. Vaculin, R. Hull, T. Heath, C. Cochran, A. Nigam, and P. Sukaviriya, "Declarative business artifact centric modeling of decision and knowledge intensive business processes," in *2011 IEEE 15th International Enterprise Distributed Object Computing Conference (EDOC 2011)*, pp. 151–160, Aug 2011.

[20] C. D. Ciccio, A. Marrella, and A. Russo, "Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches," *Journal on Data Semantics*, vol. 4, pp. 29–57, 2014.

[21] S. Steinau, A. Marrella, K. Andrews, F. Leotta, M. Mecella, and M. Reichert, "DALEC: A framework for the systematic evaluation of data-centric approaches to process management software," *Software & Systems Modeling*, September 2019.

[22] V. Künzle and M. Reichert, "PHILharmonicflows: towards a framework for object-aware process management," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 23, no. 4, pp. 205–244, 2011.

[23] D. Boaz, *Artifact-centric Business Process Management*. BizArtifact, 2016. [Software].

[24] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculin, "Introducing the guard-stage-milestone approach for specifying business entity lifecycles," in *Web Services and Formal Methods* (M. Bravetti and T. Bultan, eds.), (Berlin, Heidelberg), pp. 1–24, Springer Berlin Heidelberg, 2011.