Workflow Modeling using Proclets

W.M.P. van der Aalst[1,2], P. Barthelmess[2], C.A. Ellis[2], and J. Wainer[2,3]

[1] Department of Technology Management, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands. w.m.p.v.d.aalst@tm.tue.nl
[2] Department of Computer Science, University of Colorado at Boulder, Campus Box 430, Boulder, CO 80309-0430, USA. {barthelm,skip}@colorado.edu
[3] Department of Computer Science, State University of Campinas, Caixa Postal 6176, 13083-970, Campinas - SP, Brazil wainer@dcc.unicamp.br

Abstract. The focus of traditional workflow management systems is on control flow *within one* process definition, that describes how a single case (i.e., workflow instance) is handled in isolation. For many applications this paradigm is inadequate. Interaction between cases is at least as important. This paper introduces and advocates the use of interacting *proclets*, i.e., light-weight workflow processes. By promoting interactions to first-class citizens, it is possible to model complex workflows in a more natural manner, with improved expressive power and flexibility.

1 Introduction

Workflow Management Systems allow for the explicit representation and support of business processes. Available workflow management systems have difficulties dealing with the dynamic and inter-organizational nature of today's business processes [26]. We will argue that one of the core problems of current workflow languages is the focus on isolated case-based processes.

In traditional workflow management systems, the control-flow of a workflow is described by one *workflow process definition*, that specifies which tasks need to be executed and in what order. Workflow process definitions are instantiated for specific *cases*. Examples of cases are an insurance claim, or an order.

Today's workflow management systems assume that a workflow process can be modeled by specifying the *life-cycle of a single case in isolation*. For many real-life applications this assumption is too restrictive. As a result, the workflow process is changed to accommodate the workflow management system, the control-flow of several cases is artificially squeezed into one process definition, or the coordination amongst cases is hidden inside custom built applications. Consider for example an engineering process of a product consisting of multiple components. Some of the tasks in this engineering process are executed for the whole product, e.g., the task to specify product requirements. Other tasks are executed at the level of components, e.g., determine the power consumption of a component. Since a product can have a variable number of components and the components are engineered concurrently, it is typically not possible to squeeze this workflow into one process definition. In most workflow management systems, it is not possible to concurrently instantiate selected parts of the workflow process a variable number of times.

To solve these problems, we propose an approach based on *proclets*, *performatives* and *channels*. Proclets are light-weight processes. The interaction between proclets is modeled explicitly, i.e., proclets can exchange structured messages, called *performatives*, through *channels*. By adopting this approach the problems related to purely case-based processes can be avoided.

The remainder of this paper is organized as follows. First, we motivate our approach by identifying the problems encountered when modeling the reviewing process of a conference. Then we present the framework, which is based on Petri nets [22, 23] and inspired by concepts originating from object-orientation [9, 24], agent-orientation [19], and the language/action perspective [15, 30–32]. Finally, we compare the framework with existing approaches and conclude with our plans for future research.

2 Motivating Example: Organizing a Conference

The process of selecting papers for a conference presents features that challenge existing modeling languages. The goal of this process is to select papers out of a larger set, based, e.g., on quality, minimum and maximum number of papers, and so on. After a set of people accepts to act as program committee members, a call for papers is issued. Authors submit papers that are then reviewed by peers (invited by pc members) and finally a selection is made. Such process is complicated by a series of factors:

- Prospective PC members and reviewers may accept or reject the invitation to join the committee and to review one or more papers, respectively. Replacements for those that rejected need to be found.
- Reviewers can fail to return reviews on time. As a result, some of the papers may lack enough reviews to allow their fair evaluation.
- For effective distribution, classification and matching, the set of papers needs to be considered as a whole, i.e., distribution can not be done only considering individual papers in isolation.
- For selection, paper quality needs to be gauged against the quality of remaining papers. Again, this requires that the set of papers be considered as a whole.

A modeler faces many problems translating these requirements. A first basic question is what is to be considered the case[1] - the submission, the review, the “empty slot” in the conference, that one wants to fill with a good quality paper, or is the case the whole set of slots?

The class diagram (Figure 1) shows that different tasks rely on information that is at different levels of aggregation - some of the tasks operate at the conference level, that groups all papers, others at the paper level, and others yet at the lower level of individual reviews. A major obstacle is, therefore, how to conciliate these multiple perspectives into one model.

Lacking the power to express differences in aggregation, most workflow management systems force one to depict the process at an arbitrarily chosen level. Important shortcomings result:

[1] Workflow instance.

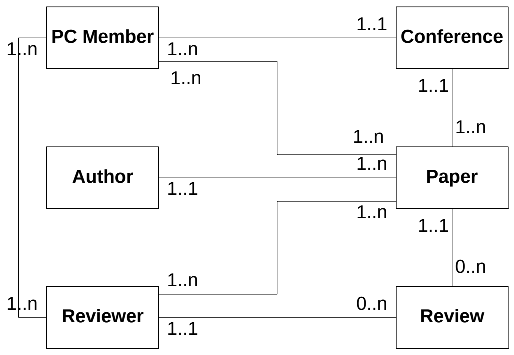

Fig. 1. Review process class diagram.

– Models are artificially flattened, being unable to account for the mix of different perspectives of the real process.
– Batch tasks are usually not supported. Batch tasks are those that require grouping sets of elements, e.g., the set of papers during distribution and selection.
– Launching and synchronizing variable numbers of tasks is also usually a problem, e.g., launching and synchronizing reviews from a paper centered case.
– Actors sometimes interact in complex ways. These interactions are usually not incorporated in process models.

Conference review is not an atypical example, in the sense that one encounters similar problems very frequently in other areas as well, for example:

– In software development: changes to one module may impact a variable number of other modules, making necessary the instantiation of a variable number of cascading tasks.
– Processing of insurance claims: some claims may refer to the same accident. At some point in time it is desirable that all related claims be merged so that a uniform decision can be reached.
– Hiring new people: candidates have to be evaluated and ranked with respect to each other. Again, the interactions between the applications are most relevant.

3 Framework

The examples given in the previous section show that today's workflow management systems typically have problems dealing with workflow processes that are not purely *case-oriented*.

Inspired by these problems, we have developed a new framework for modeling workflows. This framework is based on *proclets*. A proclet can be seen as a lightweight workflow process equipped with a knowledge base containing information on previous interactions. One can think of proclets as objects equipped with an explicit life-cycle (in the object-oriented sense) [9, 24]. Proclets interact via *channels*. A channel is the medium to transport messages from one proclet to another. The channel can be used to send a message to a specific proclet or a group of proclets (i.e., multicast). Based on the properties of the channel, different kinds of interaction are supported, e.g., push/pull, synchronous/asynchronous, and verbal/non-verbal. In order for proclets to find each other, there is a *naming service*, that keeps track of registered proclets. The concepts *proclet*, *channel* and *naming service* constitute a framework for modeling workflow processes (see Figure 2).

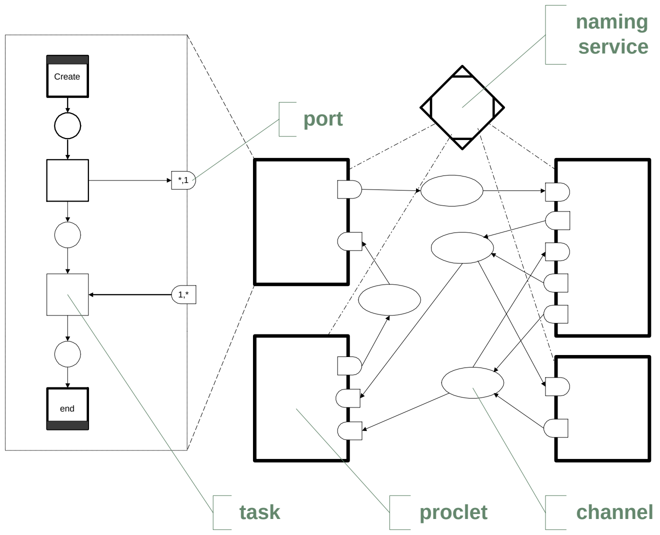

Fig. 2. Graphical representation of the framework.

Compared to existing workflow modeling languages, complex case-based workflow definitions describing the control flow of an entire process are broken up into smaller interacting proclets, i.e., there is a shift from control to communication. The framework is based on a solid process modeling technique (Petri nets [22, 23]) extended with concepts originating from object-orientation [9, 24], agent-orientation [19], and the language/action perspective [15, 30–32].

In the remainder of this section we present the four main components of our framework: *proclets*, *channels*, *naming service*, and *actors*.

3.1 Proclets

A *proclet class* describes the life-cycle of *proclet instances*. A proclet class can be compared to an ordinary workflow process definition or workflow type [18]. The class describes the order in which tasks can or need to be executed for individual instances of the class, i.e., it is the specification of a generic process. Proclet instances can be created and destroyed, and are executed according to a class specification. Proclet instances have a state.

To specify proclet classes, we use a graphical language based on *Petri nets*, an established tool for modeling workflow processes [1, 2, 6, 12, 13]. Powerful analysis techniques can be used to verify the correctness of Petri net models [22, 23].

In this paper, we use a specific subclass of Petri nets, the so-called class of sound WF-nets [1, 2].[2] A WF-net has source and sink transitions: A source transition has no input places and a sink transition has no output places. Every node (i.e., place or transition) is on a path from some source transition to some sink transition. Moreover, any WF-net is connected, i.e., the network structure cannot be partitioned in two unconnected parts. A WF-net becomes activated if one of the source transitions fires. In the remainder we assume that a WF-net becomes activated only once (single activation), and furthermore, that it is *sound* (see [1, 2] for a discussion of soundness).

Most workflow modeling languages primarily focus on control flow inside one process definition and (partly) abstract from the interaction between process definitions, i.e., coordination is limited to the scope of the process definition and communication and collaboration are treated as second-class citizens. Our framework explicitly models interactions between proclets. The explicit representation of interaction is inspired by the *language/action perspective* [32, 31] which is rooted in *speech act* theory [25]. The language/action perspective emphasizes how coordination is brought about by communication. The need for treating interaction as first-class citizens is also recognized in the agent community [19]. Emerging agent communication languages such as KQML [14] demonstrate this need.

Inspired by these different perspectives on interaction, we use *performatives* to specify communication and collaboration among proclets. A performative is a message exchanged between one sender proclet and one or more receiver proclets. A performative has the following attributes:

(1) *time*: the moment the performative was created/received.
(2) *channel*: the medium used to exchange the performative.
(3) *sender*: the identifier of the proclet creating the performative.
(4) *set of receivers*: list of identifiers of the proclets receiving the performative.
(5) *action*: the type of the performative.
(6) *content*: the actual information that is being exchanged.

The role of these attributes will be explained later. At this point, it is important to note the action attribute. This attribute can be used to specify the illocutionary point of the performative. Examples of typed performatives are request, offer, acknowledge, promise, decline, counter-offer or commit-to-commit [32]. In this paper, we do not restrict our model to any single classification of performatives (i.e., a fixed set of types). At the same time we stress the importance of using the experience and results reported by researchers working on the language/action perspective.

Proclets combine performatives and sound WF-nets. A *proclet class* PC is defined as follows:

[2] For the readers familiar with WF-nets: For notational convenience we omit the unique source and sink place used in [1, 2].

(1) PC has a *unique name*. This name serves as a unique identification of the class - the *class_id*.
(2) PC has a *process definition* defined in terms of a *sound WF-net*. The transitions correspond to *tasks* and the places correspond to *state conditions*.
(3) PC has *ports*. Ports are used to interact with other proclets. Every port is connected to one transition.
(4) Transitions can send and receive *performatives* via ports. Each port has two attributes: (a) its *cardinality* and (b) its *multiplicity*. The cardinality specifies the number of recipients of performatives exchanged via the port. The multiplicity specifies the number of performatives exchanged via the port during the lifetime of any instance of the class.
(5) PC has a *knowledge base* for storing these performatives: Every performative sent or received is stored in the knowledge base.
(6) Tasks can query the knowledge base. A task may have a *precondition* based on the knowledge base. A task is enabled if (a) the corresponding transition in the WF-net is enabled, (b) the precondition evaluates to true, and (c) each input port contains a performative.
(7) Tasks connected to ports have *post conditions*. The post condition specifies the outcome of the task in terms of performatives generated for its output ports.

A proclet class is a generic definition. Proclet instances are created by instantiating the proclet class and have a *unique identification* - the *proc_id*. Tokens in the WF-net specifying the process definition refer to one proclet instance, i.e., tokens of different proclet instances are *not* merged into one WF-net. Moreover, each proclet instance has its own knowledge base.

A performative has by definition one sender, but can have multiple recipients. The sender is always represented by a proc_id, i.e., by its identifier. The list of recipients can be a mixture of proc_id's and class_id's, i.e., one can send performatives to both proclet instances and proclet classes. A performative sent to a proclet class is received by all proclet instances of that class.

To illustrate the framework we use the example shown in Figure 3. There are two proclet classes, used to organize meetings. Proclet class *Meeting* is instantiated once per meeting. Proclet class *Personal entry* is instantiated for every potential participant of a specific meeting. The instance of class *Meeting* first multicasts an invitation to all potential participants. Note that the cardinality of the port connected to task *Invite for meeting* is denoted by a star $*$. This star indicates that the invitation is sent to an arbitrary number of potential participants, i.e., the performative has multiple recipients. We will use $*$ to denote an arbitrary number of recipients, $+$ to denote at least one recipient, 1 to denote precisely one recipient, and *?* to denote no or just a single recipient. Performatives with no recipients are considered not to have occurred, and are not registered in the knowledge base. The multiplicity of the output port connected to task *Invite for meeting* is denoted by the number 1. This means that during the lifetime of an instance of class *Meeting* exactly one performative is sent via this port. The invitation performative is sent though the channel *E-mail* (details in Section 3.2). The performative creates a proclet for each recipient, i.e., creation task *Create entry* is triggered. Creation tasks are depicted by squares with a black top. The input port connected

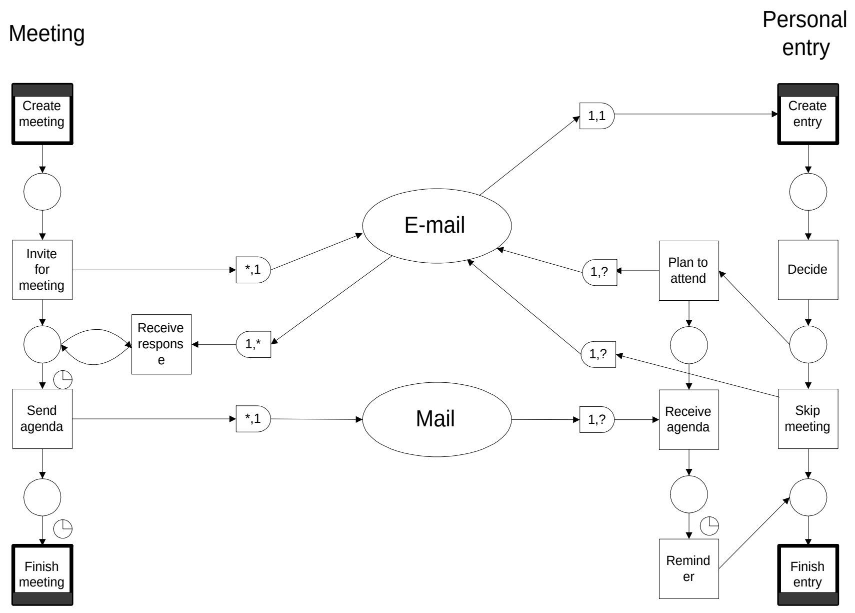

Fig. 3. Example of two proclet classes: *Meeting* and *Personal entry*.

to *Create entry* has cardinality 1 and multiplicity 1. Every input port has by definition cardinality 1, i.e., from the perspective of the receiving proclet there is only one proclet receiving the performative. Input ports connected to a creation task (i.e., a source transition) have by definition a multiplicity of 1 or ?: An instance can be created only once. Since there is just one creation task in *Personal entry*, the multiplicity is 1. After an instance of the class *Personal entry* is created, a decision is made (task *Decide*). Based on this decision either task *Skip meeting* or *Plan to attend* is executed. In both cases a performative is sent to the instance of the proclet class *Meeting*. The performative is either a confirmation (*Plan to attend*) or a notification of absence (*Skip meeting*). Note that each instance of the class *Personal entry* sends such a performative. These performatives are sent through channel *E-mail*. Note that the ports connected to *Plan to attend* and *Skip meeting* both have cardinality 1 (i.e., one recipient) and multiplicity ? (one performative is sent via one of the two ports). Task *Receive response* is executed once for every “confirmation/notification of absence” performative. After some time, as indicated by the clock symbol [2], task *Send agenda* is executed. *Send agenda* generates one performative: the agenda of the meeting. This performative is sent to all proclets that confirmed the invitation (this information is in the knowledge base of the *Meeting* proclet). The proclets that confirmed the invitation receive the agenda (task *Receive agenda*) and a timer for the task *Reminder* is set. Finally, all proclets are destroyed by executing the finishing tasks *Finish meeting* and *Finish entry*. The finishing tasks (i.e., sink transitions) are depicted by squares with a black bottom.

3.2 Communication Channels

Communication channels are used to link proclets. Channels transmit messages containing performatives. There are many different categories of channels defined by channel properties such as medium type, reliability, security, synchronicity, closure, and formality. These properties are briefly explained:

- *Medium Type*
  This can be point-to-point or broadcast, or some form of limited multicast. Recall that performatives can be sent to an individual proclet instance (point-to-point), a set of proclets (multicast), or an entire proclet class (broadcast). Common media include postal mail, telephone, and electronic mail.
- *Reliability*
  Some channels are very reliable; some are unreliable. For some electronic channels, we assume that the technology is robust, and that error detection and retransmission are implemented at lower layers of the communication protocols. Sometimes channels are inherently unreliable (such as in data channels in some lesser developed countries).
- *Security*
  At times the content of a performative is considered to be quite valuable and secret. In such cases, the transmission should be via secure channels.
- *Synchronicity*
  Some channels are used for real time communications in which each party expects to get rather immediate feedback from recipient parties. This requires synchronous channels. Face-to-face spoken conversation falls into this category. In the case of an asynchronous channel, the sender usually is not waiting for an immediate response.
- *Closure*
  Channels can be classified as open or closed. When a channel is open, the sender does not know exactly who, and how many recipients are connected. When a channel is closed, the exact identity of all recipients is specified in advance. A radio broadcast, and a notice posted on a bulletin board are examples of open medium communications, in which the senders do not exactly know who are the recipients.
- *Formality*
  Some channels convey much more formality in the messages delivered than others. Performatives can be very formally specified, or can be informal and flexible. Generally, business letters are much more formal than chat rooms. A careful record is kept of formal channel transmissions, whereas informal channels are usually not recorded.

Clearly, channel properties and performative types are closely related, i.e., for a given performative certain properties are appropriate, others are not. For example, for the performative "You are fired!" a point-to-point, reliable, secure, synchronous, closed, and formal channel is most appropriate.

3.3 Naming service

All interaction is based on proclet identifiers (proc_id's) and class identifiers (class_id's). These identifiers provide the handles to route performatives. By sending a performative using a class_id, all instances of the corresponding class receive the performative. In many situations the sending proclet does not know the proc_id's of all receiving proclets. The naming service keeps track of all proclets and can be queried to obtain proc_id's. There are many ways to implement such a naming service. Consider, e.g., object request brokers developed in the context of CORBA. In this paper, we only consider the desired functionality and abstract from implementation details (e.g., distribution of the naming service over multiple domains).

The naming service provides the following main primitives: *register*, *unregister*, *update*, and *query*.

The function *register* is called by the proclet the moment it is created. The execution of one of the create tasks (i.e., source transitions) coincides with the execution of the *register* primitive.

The naming service stores a set of attributes for each proclet. These attributes are not fixed and may vary from one class to another. The function *update* with parameters *proc_id* and *attributes* can be used to change existing or add new attributes.

Based on the attributes, proclets can query the naming service using the function *query*. The function has one parameter describing a Boolean expression in terms of attributes and returns a set of proc_id's, i.e., all proclets satisfying the expression.

Entries in the naming service can be removed using the function *unregister*. Executing a finish task (i.e., a sink transition in the WF-net) results in a call to *unregister*.

3.4 Actors

Proclets have *owners*. Owners are the actors responsible for the proclet. Actors can be automated components, persons, organizations (e.g., shipping department), or even whole companies. Owners are specified at proclet registration time and this information is kept by the naming service (see Section 3.3). Ownership can be transferred by updating the naming service information.

Owners will sometimes be the executors of proclet tasks themselves - in the example of Figure 3, for instance, owners of each personal entry will most probably be the ones that will perform the tasks, essentially the decision of attending or skipping the meeting. Roles may be specified for each task, in which case the executor can be different from the owner. We assume then that the usual role resolution mechanisms [34] are employed.

We propose to model as *external proclets* those actors (in the broad sense of the word) that interact with proclets in a more complex way. *External proclets* are useful to model those interactions that go beyond the simple model assumed by the usual role mechanism, e.g., when a request for service may be either accepted, rejected or counter-proposed. *External proclets*, as the name implies, represent entities that are outside of the scope of the process proper, whereas *internal proclets* are those under the control of the workflow system's enactment service. Both types of proclets are modeled in a similar way - by describing expected interactions with other proclets. For more extensive examples of both *internal* and *external proclets*, see [5]. This technical report also describes the application of the approach to the example described in Section 2 (i.e., the workflow of organizing a conference).

4 Related work

Petri nets have been proposed for modeling workflow process definitions long before the term “workflow management” was coined and workflow management systems became readily available [12, 13]. Workflow models described in the literature focus on various aspects (cf. [26]) such as transactional concepts [16], flexibility [21], analysis [1, 2], and cross-organizational workflows [3, 4], etc. Any attempt to give a complete overview of these models is destined to fail. Therefore, we only acknowledge the work that extended workflow models to accommodate the problems identified in Section 2.

Zisman [33] presents a paper refereeing example that involves Petri-nets and allows multiple instantiation of the reviewer net.

Batch-oriented tasks were discussed in [8]. Creation of multiple instances of tasks have been proposed by some, e.g., Casati et al. [10] (multi-tasks); Regatta system by Fujitsu [27] (multi-stage); Spade-1 [7] (multiple active copies). The framework presented here is more generic. Multiple instantiation is just one aspect of a broader view of interactions as first-class citizens.

The idea to promote interaction to first-class citizens was proposed in different settings. In the context of language/action perspective [15, 30–32], Action Technologies developed a workflow tool [28]. Speech-acts also form the basis for performatives in agent interaction languages, e.g. KQML [14]. Agents are used to implement workflows, e.g., in the Bond multi-agent system [29] and others (e.g., [20, 11]). In the more systems-oriented domains there have also been some proposals for inter-process communication (e.g. in Opera [17]).

Some of the ideas presented in this section have been adopted by our framework: batch-oriented operation, multi-tasks, and inter-process communication can be handled easily by the framework. In addition, the framework employs concepts such as performatives, channels, ports, knowledge bases, naming services, and the rigor of a Petri-net basis which allows for various forms of analysis and a straightforward and efficient implementation.

5 Conclusion

In this paper, we presented a framework which advocates the use of interacting proclets, i.e., light-weight workflow processes communicating by exchanging performatives through channels. As was demonstrated in [5], the framework can solve many of the traditional modeling problems resulting from the case-oriented paradigm.

In the future, we plan to explore the relation between channels and performatives. We are also compiling a list of interaction patterns. In our view, the interaction between proclets typically follows a number of well-defined patterns, e.g., a request performative is followed by an accept or reject performative. Finally, we plan to build a prototype to support the framework. This prototype will be used to support the reviewing process of the ACM biannual Siggroup conference following the model described in this paper.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
2. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
3. W.M.P. van der Aalst. Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets. *Systems Analysis - Modelling - Simulation*, 34(3):335–367, 1999.
4. W.M.P. van der Aalst. Process-oriented Architectures for Electronic Commerce and Interorganizational Workflow. *Information Systems*, 24(8):??–??, 2000.
5. W.M.P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Workflow modeling using proclets. Technical report cu-cs-900-00, University Of Colorado at Boulder, February 2000. http://www.cs.colorado.edu/ skip/proclets.pdf.
6. N.R. Adam, V. Atluri, and W. Huang. Modeling and Analysis of Workflows using Petri Nets. *Journal of Intelligent Information Systems*, 10(2):131–158, 1998.
7. S. Bandinelli, M. Braga, A. Fuggetta, and L. Lavazza. Cooperation support in the spade environment: a case study. In *Proceedings of the Workshop on Computer Supported Cooperative Work, Petri nets, and Related Formalisms (14th International Conference on Application and Theory of Petri Nets)*, Chicago, June 1993. ftp://ftp-se.elet.polimi.it/dist/Papers/ProcessModeling/CSCWPN93.ps.
8. P. Barthelmess and J. Wainer. Workflow systems: a few definitions and a few suggestions. In N. Comstock and C.A. Ellis, editors, *Proceedings of the Conference on Organizational Computing Systems - COOCS'95*, pages 138–147, Milpitas, California, September 1995. ACM Press.
9. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, Reading, MA, USA, 1998.
10. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual modeling of workflows. In *Proceedings of the OOER International Conference*, Gold Cost, Australia, 1995.
11. J.W. Chang and C.T. Scott. Agent-based wrokflow: Trp support environment (tse). *Computer Networks and ISDN Systems*, 28(1501), 1996.
12. C.A. Ellis. Information Control Nets: A Mathematical Model of Office Information Flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, pages 225–240, Boulder, Colorado, 1979. ACM Press.
13. C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, 1993.
14. T. Finin, J. Weber, G. Wiederhold, and et. al. Specification of the KQML Agent-Communication Language , 1993.
15. F. Flores and J.J. Ludlow. Doing and Speaking in the Office. In *Decision Support Systems: Issues and Challenges*, pages 95–118. Pergamon Press, New York, 1980.
16. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.
17. C. Hagen and G. Alonso. Beyond the black box: Event-based inter-process communication in process support systems (extended version). Technical report, ETH Zürich, July 1997. Technical Report No. 303. http://www.inf.ethz.ch/department/IS/iks/publications/files/ha98c.pdf.
18. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.

19. N. Jennings and M. Wooldridge, editors. *Agent Technology : Foundations, Applications, and Markets*. Springer-Verlag, Berlin, 1998.
20. M. Merz, B. Liberman, K. Muller-Jones, and W. Lamersdorf. Interorganisational Workflow Management with Mobile Agents in COSM. In *Proceedings of PAAM96 Conference on the Practical Application of Agents and Multiagent Systems*, 1996.
21. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
22. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
23. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets II: Applications*, volume 1492 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
24. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley, Reading, MA, USA, 1998.
25. J.R. Searle. *Speech Acts*. Cambridge University Press, Cambridge, 1969.
26. A.P. Sheth, W.M.P. van der Aalst, and I.B. Arpinar. Processes Driving the Networked Economy: ProcessPortals, ProcessVortex, and Dynamically Trading Processes. *IEEE Concurrency*, 7(3):18–31, 1999.
27. K. Swenson. Collaborative planning: Empowering the user in a process environment. *Collaborative Computing*, 1(1), 1994. ftp://ftp.ossi.com/pub/regatta/JournalCC.ps.
28. Action Technologies. *ActionWorkflow Enterprise Series 3.0 User Guide*. Action Technologies, Inc., Alameda, 1996.
29. Purdue University. Bond. the distributed object multi-agent system. http://bond.cs.purdue.edu, 2000.
30. E.M. Verharen, F. Dignum, and S. Bos. Implementation of a cooperative agent architecture based on the language-action perspective. In *Intelligent Agents*, volume 1365 of *Lecture Notes in Artificial Intelligence*, pages 31–44. Springer-Verlag, Berlin, 1998.
31. T. Winograd. Special Issue on the Language Action Perspective - Introduction. *ACM Transations on Office Information Systems*, 6(2):83–86, 1988.
32. T. Winograd and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex, Norwood, 1986.
33. M. D. Zisman. Use of production systems for modeling asynchronous concurrent processes. *Pattern-Directed Inference Systems*, pages 53–68, 1978.
34. M. zur Mühlen. Evaluation of workflow management systems using meta models. In *Proceedings of the 32nd Hawaii International Conference on System Sciences - HICSS'99*, pages 1–11, 1999.