FROM PROCESS TO SOFTWARE SYSTEMS' SERVICES
Using a Layered Model to Connect Technical and Process-related Views

Christian Prpitsch
Institute for Computer Science and Business Information Systems
University of Duisburg–Essen, Germany

Keywords: Abstraction, layer, process support, workflow, reusable subprocess, collaboration, e-learning.

Abstract: A layered model is a solution to the problem of two disjunct points of view on the same problem. One group has a technical background but does not know much processes. The other group's members have a process perspective. This contribution introduces a meta–model consisting of three layers. The outer ones represent the disjunct points of view. The inner one contains elements of both other layers and combines them. It also contains a system–independent scripting language to automatically configure software systems.

1 INTRODUCTION

The administration of collaborative courses in a learning environment is a collection of complex and repeated tasks. A collaborative course is in this contribution a course where learners build groups and do some exercises. Administration is done by the lecturer and technican, called acotrs, in terms of watching and moderating the learning process and taking care of software systems. We answer the question what language or meta–model should be used for supporting both lecturer and technicans.

The example scenario is derived from e–learning but the presented model can be transfered to many other use cases. Therefore one has to interchange actors and the workflow itself. We evaluated the model by using it in a scenario of cooperative writing. It is well known to most scientists

Users not being familiar with complex information systems often have difficulties to make decisions about the usage of software. They usually do not have enough competencies and experience in the evaluation of systems requirements. System administrators in general belong to a group of staff not being familiar with complex workflows in different business cases. So they need a model to exchange requirements and opportunities.

This contribution starts with a short overview about existing meta–models on both sides. Then we outline a scenario of a collaborative course in e–learning. With this scenario we try to model both the perspective of a workflow as seen by the lecturer and the perspective of software systems and services as seen by the technicans with the shown approaches. This is either not possible or does not meet the needs. After that we present our approach of connecting both perspectives by defining an additional layer of abstraction. In the last section we shortly describe the improvements introduced by our approach.

2 RELATED WORK

This section summarises some selected approaches of modeling the perspectives business process and software systems as described in other research projects on this matter. We selected them by relevance on their specific area. A business process is related to a business case with defined start and termination. A workflow is defined as the technical support of a business process. The workflow contains subprocesses doing small subtasks (Mueller, 2005, 8). In this contribution we use the term *workflow* to express the model of a complete business case and *subprocess* for subtasks. A workflow is independent of concrete software systems but uses services for describing them. It can be used in different environments by assigning different software systems. Subprocesses are treated as patterns and can be reused in many workflows.

When we compared many different workflows in the course of the study there were several common patterns concerning parts of the process. Some of these patterns belong to cooperative writing as de-

Prpitsch C. (2008).
FROM PROCESS TO SOFTWARE SYSTEMS' SERVICES - Using a Layered Model to Connect Technical and Process-related Views.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - ISAS*, pages 298-304
DOI: 10.5220/0001703202980304
Copyright © SciTePress

scribed by (Lowry et al., 2004). They show several patterns how collaborative work can take place in the task of writing a text. Controlling of writing tasks is explained in (Posner and Baecker, 1992).

The first approach is the *Unified Activity Management (UAM)* by (Harrison et al., 2005) and (Cozzi et al., 2006). It is based on an artefact named *activity* which brings semantics and structure to a given problem and consists of metadata and assignments. The activity is a layer of abstraction in the workflow. It is visible and editable by every user. So the user is aware of the activity and the usage of an activity management system. They include references on resources and users / people. Every user inerts a role in the context of the activity. The approach UAM is designed to support a number of users while performing cooperative tasks. UAM is used as a tool to coordinate and control a business process. There is an evaluation of UAM in (Cozzi et al., 2006, 708) with a case study of accomodation for new internees at a scientific institution.

The standard for orchestration of web services *Web Service Business Process Execution Language (BPEL)* by *OASIS* (OASIS, 2007) defines a language to orchestrate existing web services with connection elements derived from process definition. XML is used as meta–language. BPEL does not specify any graphical representation. The *Business Process Modelling Notation (BPMN)* (Object Management Group, 2007) by OMG is one language to solve this problem. The metamodel consists of tasks, (sub–)processes and control sequences arranged in so called swimlanes. In BPEL there are abstract and concrete workflows specified. A concrete workflow is ready to be executed by a workflow processing engine. Any abstract workflow uses the same syntax and semantics as a concrete one but there is some information missing so that it can not be executed. A good comparison is the object–oriented pattern of abstract and concrete classes.

There is an extension to BPEL called *BPEL for People (BPEL4People)* (Agrawal et al., 2007) by an industrial consortium. It adds human beings into the BPEL model. Users can be grouped to roles. They carry out activities. This extension is ignored because BPEL needs a central workflow engine and therefore does not meet the requirements.

Abstract and concrete workflows are also mentioned by (Deelman et al., 2003). The goal of that concept is to create workflows which are defined in terms of services instead of systems. The term “workflow” means a complex process of performing data intensive operations on grid environments. An abstract workflow is not runnable within a workflow engine. It has to be transfered into a concrete one by replacing the services with concrete systems providing those services.

Modelling of business processes is done by the usage of *Event–Controlled Process Chains (EPCs)* as first mentioned by (Keller et al., 1992). An EPC consists of alternating events and business–functions connected by control sequences. They are used in an extended Version for describing business cases for enterprise ressource planning systems. A business–function is carried out by an anctor which can be a human being or any technical system. There is an approach to use EPCs as graphical representation of BPEL by (Mendling and Ziemann, 2005).

3 SCENARIO

This section shows a shortened scenario from the context of e–learning. It is used to get requirements on a meta–model for modelling this kind of scenarios. The author has some years of experience with the scenario in virtual study course. The scenario is a course about basic principles in programming. The students get a simplified real–world problem and have to create a runnable program. They are requested to build small groups of 3 to 5 students having mixed qualifications.

The actors in the scenario are a lecturer, a technican per software system, and many students. On the technical site there are one Learning Management System (LMS) and one programming environment (PE). All the learning content is contained in the LMS. While creating the course for the first time, the lecturer made a workflow of how collaboration will take place, which people are involved, and where to get the technical resources. This workflow is then written on a piece of paper. Every term it is executed by hand. The workflow is illustrated in figure 1.

Students have to form groups and tell the lecturer by e–mail about that to manually create corresponding usergroups in the LMS. The disadvantage of this subtask is the media break while using e–mail (Whittaker and Sidner, 1996; Fisher et al., 2006). Every group gets their own workspace in the LMS and PE where only group members and lecturers are allowed to step in. In the PE the lecturer has to order the creation of the workspace including all necessary software by the administrator. Thereafter the PE is ready to be accessed by the students. For equality reasons all workspaces have to be identically prepared. After the students have finished their exercise they submit the solution by mail. The lecturer grades the submissions and the scenario is terminated.

The lecturer had to decide what systems to use for the course. There are requirements hidden in the non–

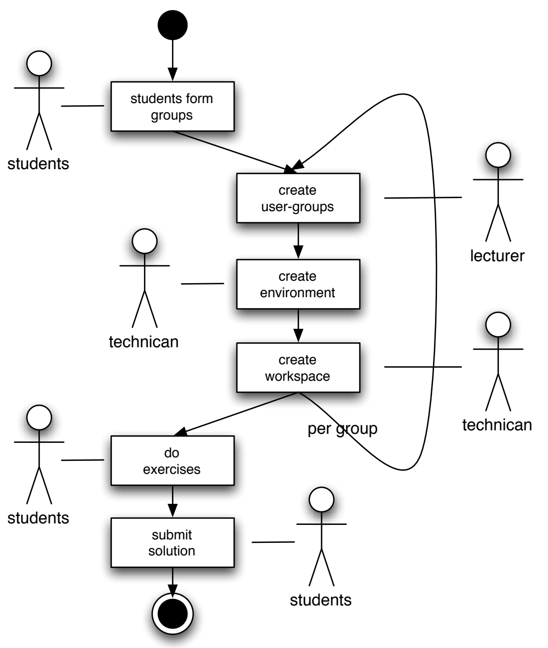

Figure 1: Process with actors while doing the course.

formal workflow. The workflow itself is not able to be executed automatically. The abstracted workflow of the course "Introduction into Programming" shares process patterns (Lowry et al., 2004) with any course where students have to write a text in groups. In a more complex scenario there are additional systems to be integrated in the workflow (e.g. version control).

This is the simplest form of the workflow. To reduce manual interaction and make it more realistic, we add some more process control. It is derived from *IMS Learning Design (LD)* (IMS Global Learning Consortium, 2003). There are special dates, when exercises have to be completed. After these dates there are sample solutions put into a common workspace to be viewed by everyone. The same pattern is to be used if students are requested to bring their work to a defined state until a set date. This is a task in the PE, where the LD–player is not available. In the scenario of collaborative writing there are dates to synchronize the work, as shown by (Lowry et al., 2004, 74ff).

We discovered in this scenario some weeknesses concerning process management. First, the workflow is not persisted in a formal way. Therefore, it is impossible to automatically execute it. It is not efficient to create it manually for supporting every single course. The repeated tasks to support every group is simple to execute but creates a lot of effort in sum of all groups. A central controlling instance is to be avoided because autonomy is left at the systems. The following requirements are needed to make a solution:

- workflow is in a formal notation
- minimal effort to implement into existing software
- no central controlling instance
- repeatable tasks must be supported
- reusable subprocesses to be used while modeling
- flexible adding of reusable artefacts
- support of time–based actions

4 EVALUATION OF EXISTING APPROACHES

In this section we will discuss the existing approaches. The evaluation is done by comparing the meta–models from section 2.

All presented standards and approaches do not provide a solution for modelling the scenario from both technical and process–related perspective. The connection between a technical model consisting of software systems and their provided services and a process–related one with actors and tasks is not available yet. In EPC there is an ability to map a process on a BPEL–model as mentioned by (Mendling and Ziemann, 2005). As mentioned in the scenario we must not have one central application for controling the workflow. If we would use BPEL we had to use a so called *workflow engine* (OASIS, 2007). Therefore these technologies to connect process and technical perspective are not usable in our context because of BPEL.

Both (Deelman et al., 2003) and BPEL (OASIS, 2007) use a term "abstract workflow" but in a different manner so both are not compatible. We define an abstract workflow the same as (Deelman et al., 2003). The definition from BPEL is used for a *partly concrete workflow*.

The UAM approach addresses mainly endusers. They have to work on the artefacts and therefore systems require changes which means additional effort while implementing it. In the scenario from section 3 is no need to present artefacts to a user. In our experience there have often been difficulties while presenting too much new technology to a user. Therefore the solution's artefacts have to be invisible to some roles like students.

Our additional requirement of time–based actions cannot be modeled from both perspectives. The technical models use timers with very short duration (seconds to hours) but usually do not give access to long–

run dates like three months or so. The process–oriented ones provide timers “until a date” or “during a period”. Both are incompatible.

5 SOLUTION BY ADDING ABSTRACTION

Our solution sections the problem from section 3 into layers as presented in figure 2. The idea is to use two layers, each of them to be used by one group. Groups are divided by tasks and competencies. Then we introduce a third layer between them to be used as a connection. The upper layer is used by a group with competencies in formal description of business processes. In the above scenario this is the lecturer. The lowest layer is to be used by technicans with high competencies in administrating software systems. They are also able to describe services provided by their systems. We use the middle layer to create a connection between both layers. Figure 3 shows the process by using our model. It starts with the initial creation and terminates when the course is ready to be used by a class. The next paragraphs will explain the three layers and their connections.

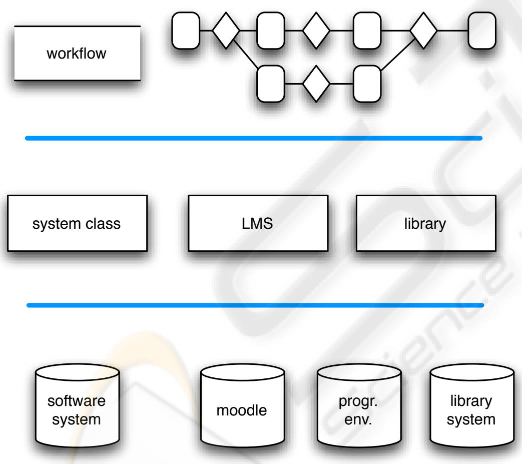

Figure 2: Layers of abstraction with some items.

The layer of software systems represents any systems or applications provided for use in the later mentioned workflow. Usually a system is implemented as a web accessible application but we also assume standalone applications and systems without user–interface belonging to this. Any application has to provide services. The definition of web services (SOAP by W3C) explains a *service* as a machine–to–machine service without any direct user–interaction. To have a complete description of all provided services we have to add the definition of user–interfaces as well. While web services are to be defined by WSDL the users’ services are to be defined by a more task–related notation in normal language. To get the available services of one special system it has to implement a service called *explain* which provides all needed metadata about the system.

The goal of defining web services by the *Web Services Definition Language (WSDL)* is to enable inter–system–operability. In order to achieve the interoperability of system classes one has to define a set of services to be provided by every system of a class. There are also projects dealing with this: The MISTEL project (Bopp et al., 2006), the JAVA Content Repository API Standard (Nuescheler and Piegaze, 2006) and the IMS Resource List Interoperability (IMS Global Learning Consortium, 2004). These projects describe some systems, abstract from their special implementation by the definition of services and interchangable formats.

We define a *system class* by its provided services. Every class has to implement a *service set* containing mandatory and optional services. Different system classes do not need to have disjunct service sets. There are some groups of services as predefined classes, e. g. LMS, PE and Document repository. Their users’ services are well known, so we do not mention them again. The metadata provided by the explain–service refers to system classes and tells about optional services. One software system can belong to several system classes if it fulfills their service sets. System classes have to be defined from the perspective of a user and not of a technican. Therefore the grouping of services is done along with the definition of tasks one could do with the system class (e. g. authoring) or that could be done by the system class itself (e. g. versioning).

The most abstract layer is the *workflow* consisting of subtasks and control sequences (e.g., conditions, sequences, break points). Substasks consist of services provided by system classes. So it is matched on one or more system classes. The creator of a workflow is in general a user not being familiar with technical details. It is sufficient to describe a real–world workflow in terms of services and control sequences. They need competencies in describing a subtask in terms of services and control sequences. They can use predefined subtasks as well. Their task is to make use of their competencies and experiences in creating and rating workflows in their field of practice. Then a technican can decide what member of a system class to use while performing this workflow. The shortened process from creation of any process until its usage is shown in figure 3.

There is no “workflow engine” (see BPEL) or anything alike necessary to execute the workflows. They are created, transformed and distributed by a stand–alone or shared application. Then the workflow is executed peripheral by every system. It is intended by our approach to step in systems as less as possible to keep the implementation simple. Any permissions are left under control of the system itself. Therefore we decided not to use a central workflow engine. Now we will explain how the workflow is put into the systems.

The disadvantage of implementing a special environment for every single usergroup is addressed by the automation of workflow–implementation. Therefore the formal description of the workflow and decisions what systems to use are needed. A formal and reusable definition of how to set up the requested environment is needed. Then an engine is able to create the complete environment or even parts of it. In the scenario from section 2 we use the ability for creating parts of the environment. The usergroups are known to the workflow–engine so it can automatically create the environment.

The technology to be used for this scripting is at the time not finally defined. We actually use a format similar to XSLT to define the workflow independent from systems. This file is called *workflow file*. It contains also the configuration parts needed to create environments (see scenario). This configuration is very simple because it is a goal to be independent of software systems and just depend on system classes. One of the commands is *create workspace* along with some arguments. The second part is an XML–file containing actual data like usergroups and systems' metadata we named *data file*.

The technical process of instantiating a workflow is as follows: The workflow file “is applied” on the data file. The result is a script composed of the workflow with actual configuration details inside, as e. g. systems and usergroups. There are some parts named *config* in this file. They are pushed to the systems named inside. A system receives this command–script and either executes it directly or transforms it before. The system can apply an XSLT–file on it and gets a system–specific script because the script is still in XML. The format of this specific script has no impact on the other systems involved in the workflow.

Changes can occur while operating software systems. Our model only needs few maintenance as expleined in the following. Adding a system requires to classify it into a system class. Then the system is ready to be used. When deleting a system it has to be deleted in all system classes, too. Replacement of a system requires the update of its name and metadata in all system classes. Update of a system can require the same procedure. When a system disappears all its occurences in workflows can be found automatically.

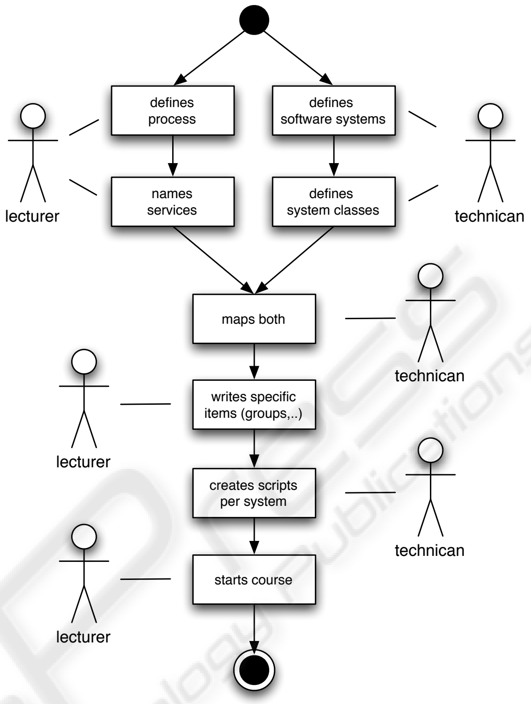

Figure 3: Defining a process in our model.

The communication between systems is done via web services SOAP. Such an architecture is called a *service–oriented architecture* as defined by (Conrad et al., 2006). The decision is based on platform independency of this kind of middleware. The second reason is the usage of HTTP as a transport protocol. Both the user–interface (in most cases a web GUI) and the machine interaction can be carried by the same protocol. As mentioned in the former paragraph we use XML to exchange information so it is consequent to transport them by a technology based on XML itself. Some services are globally defined. The most important service is the explain–service, as mentioned above. Each two systems have to agree on the exact specification of the content they interchange. In case of documents this would be the metadata set. There is a suitable solution of systems communication defined by the project MISTEL in (Bopp et al., 2006).

6 DISCUSSION OF THE APPROACH

The transformation is under supervision of the user who has to make decisions. None the less users are provided with proposals of suitable options. The different concept of abstract workflows from BPEL is used by our approach to provide the ability for partly creation of an executable workflow. We use the same definition of *abstract* as (Deelman et al., 2003). A newly created workflow is abstract by definition. It is transformed into a runnable workflow by users with assistance of the workflow management application. In this task we use the definition from BPEL to store partly defined workflows.

Our approach is focused on supporting the role lecturer. Technicans also use the approach to support lecturers by caring about their systems. In our experience it is favourably to present details only to staff really needing them, especially technical facts. In the scenario from section 2 the artefacts are transparent to users of the role student. The automatic configuration of many instances of the same software system is daily work to many system administrators. They often use self–created scripts or management–tools to improve their work. With usage of our approach those scripts are defined in a system–independent language.

The initial effort to create an environment implementing our approach depends on the systems which have to be integrated. Some licenses of commercial systems prohibit the necessary extensions so they can not be used. One working solution is the creation of proxy systems transforming SOAP calls into the systems internal format. A web service stack based on HTTP is available to nearly all programming languages, especially the widely used JAVA, PHP and Microsoft .NET.

We improved the scenario from section 3 by creating a model in terms of the abstraction model from section 5. The first improvement takes place before the course starts. The lecturer creates a model of the workflow and passes it to technicans in order to check system classes and available software systems for existance. This model can be reused partly or in whole by other lecturers who use a similar workflow. They just have to change few artefacts or control sequences. Configurations for systems are derived from the model. They are passed to the systems and a course is ready to start.

After the first step of the course is completed there are usergroups. The lecturer receives them from the LMS in an XML format. This file is merged with the model to get specific configuration–commands. These commands are passed to the intended systems via their configuration web service. This is the main improvement if the workflow would be used with only one course. Before a decision is made about using our approach or not it has to be calculated if the effort to create the scenario is less than the saved time while creating environments for every single usergroup. The step of submission can be done automatically by setting a timer. When it expires there is no more work on the exercise possible, all results are packaged and sent to the lecturer.

The effort on doing repeated tasks decreased. Students did not notice the improvement but had a profit by faster creation of their groups with equal environments. As mentioned at the beginning of section 2 it is possible to transfer this scenario on collaborative writing. Patterns identified by (Lowry et al., 2004) are to be predefined and can be reused in any future workflow.

7 CONCLUSIONS

This contribution shows a solution as model a business process on a high level and attach technical details expressed as system classes. We defined an interface between the process–oriented meta–model used by people creating or describing business processes and the technical description of software systems. While decomposing complex business processes into smaller subprocesses we create reusable artefacts. As supporting several processes we get many predefined artefacts so the chance on having something nearly ready to use rises.

Future research will be looking for even more abstraction from technical details. The goal is to allow ad hoc networks (or grids) not being installed in just the one organisation but using services and content from many systems. In the future the question should be *what to combine* to reach a goal and not *what to create*. Within the context of service–oriented architecture and service–component–oriented architecture we think we can add some benefit.

ACKNOWLEDGEMENTS

This work is funded by the *Deutsche Forschungsgemeinschaft (DFG)*, section Scientific Libraries, see www.dfg.de and the MISTEL project's website www.systemkonvergenz.de.

REFERENCES

Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., Koenig, D., Leymann, F., Mueller, R., Pfau, G., Ploesser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., and Zeller, M. (2007). Ws–bpel extension for people. Specification Vers. 1.0, online: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf.

Bopp, T., Hampel, T., Hinn, R., Pawlowski, J., and Prpitsch, C. (2006). Mistel – an approach to access multiple resources. In Manolopoulos, Y., Filipe, J., Constantopoulos, P., and Cordeiro, J., editors, *Proceedings of the 8th International Conference on Enterprise Information Systems – ICEIS 2006*, pages 319 – 322, Paphos / CY. INSTICC, INSTICC.

Conrad, S., Hasselbring, W., Koschel, A., and Tritsch, R. (2006). *Enterprise Application Integration*. Spektrum, Heidelberg / DE.

Cozzi, A., Farrell, S., Lau, T., Smith, B. A., Drews, C., Lin, J., Stachel, B., and Moran, T. P. (2006). Activity management as a web service. *IBM Systems Journal*, 45(4):695 – 712.

Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., and Koranda, S. (2003). Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25 – 39.

Fisher, D., Brush, A. J., Gleave, E., and Smith, M. A. (2006). Revisiting whittaker & sidner's "email overload" ten years later. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 309 – 312, New York / USA. ACM Press.

Harrison, B. L., Cozzi, A., and Moran, T. P. (2005). Roles and relationships for unified activity management. In *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, pages 236 – 245.

IMS Global Learning Consortium (2003). Ims learning design information model. Spezifikation, IMS Global Learning Consortium Inc., online: http://www.imsglobal.org/learningdesign. Ver. 1.0.

IMS Global Learning Consortium (2004). Ims resource list interoperability information model. Spezifikation, IMS Global Learning Consortium Inc., online: www.imsglobal.org/rli. Ver. 1.0.

Keller, G., Nuettgens, M., and Scheer, A.-W. (1992). Semantische prozessmodellierung auf der grundlage ereignisgesteuerter prozessketten. *Veroeffentlichungen des Instituts fuer Wirtschaftsinformatik, University Saarbruecken / DE*, 89:1 – 31.

Lowry, P. B., Curtis, A., and Lowry, M. R. (2004). Building a taxonomy and nomenclature of collaborative writing to improve interdisciplinary research and practice. *Journal of Business Communication*, 41:66 – 99.

Mendling, J. and Ziemann, J. (2005). Transformation of bpel processes to epcs. In Nuettgens, M. and Rump, F., editors, *Proceedings of EPK 2005*, pages 41 – 53, Hamburg / DE. CEUR Workshop Proceedings Vol. 167.

Mueller, J. (2005). *Workflow–based Integration*. Springer, Berlin / DE.

Nuescheler, D. and Piegaze, P. (2006). Content repository api for java technology specification - java specification request 170. Technical report, JSR 170 Group.

OASIS (2007). Web services business process execution language. Standard 2.0, OASIS, online: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.

Object Management Group (2007). Business process modeling notation specification. Specification v 1.2 draft, Object Management Group (OMG), online: http://www.omg.org/cgi-bin/apps/doc?dtc/07-06-03.pdf.

Posner, I. R. and Baecker, R. M. (1992). How people write together. In *Proceedings of the 25th Hawaii International Conference on System Sciences*, volume 4, pages 127 – 138. online: http://www.kmdi.toronto.edu/rmb/papers/D26.pdf.

Whittaker, S. and Sidner, C. (1996). Email overload — exploring personal information management of email. In *Proceedings of the ACM Conference on Human Factors in Computer Systems (CHI)*, pages 276 – 283. ACM, ACM Press.