Building a Process Layer for Business Applications using the Blackboard Pattern

Stefan Kleine Stegemann
Werum Software & Systems AG
Wulf-Werum Str. 3
21339 Lüneburg
sks@werum.de

Burkhardt Funk, Thomas Slotos Universität Lüneburg
Volgershall 1 21339 Lüneburg
{funk|slotos}@uni-lueneburg.de

Abstract: Contemporary business applications often employ a process layer in order to coordinate automated activities. One option to build such a layer is to use a workflow management system. But the all-or-nothing fashion of such systems makes them sometimes hard to integrate. In such cases, custom development is an alternative. Yet concepts for the micro-architecture of process layers in business applications are rare. We argue that the blackboard pattern, which is known to be suitable for knowledge intensive artificial intelligence applications, can provide an solid basis also for constructing process layers with workflow capabilities. This paper shows how the essential building parts of workflows are realized in a blackboard architecture. In this context, an in-depth discussion of important design and implementation tasks to be solved is given.

1 Introduction

Layered architectures [Bu96] are the standard for the development of modern software systems. The integration of a process layer on top of the business layer to separate business entities from processing rules and tasks follows the principle of separation of concerns. One possibility to build a process layer is the adoption of a workflow management system. But these systems often come in an all-or-nothing fashion [Ma00]. The focus on the non-programmer and a wide variety of features produced heavyweight and monolithic architectures [MJ00]. As a result, such systems are often hard to customize and to extend. Integration with existing applications can be a painful task, in particular when existing components overlap with the workflow management system [Mu99].

Consequently, software developers, who need a solution that can be tailored to their specific needs have difficulties in using off-the-shelf workflow management systems. They are often forced to build home-made solutions whenever they need workflow-capabilities [MJ00]. In this situation, two questions arise: how has a process layer to be structured in order to yield a robust and flexible solution and are there any supporting architectural patterns? To solve these problems, this paper suggests an approach that is based on a blackboard architecture. The underlying blackboard pattern is well known in the artificial intelligence domain. But there is only little research on how business applications can benefit from this pattern.

To provide the foundation for our approach, we give a brief introduction to the blackboard pattern and the complementary blackboard based control plan pattern. In the next step, the essential structure of a process layer with workflow capabilities is identified. We show how the elements of this structure can be addressed in a blackboard architecture. Our approach suggests a solution template for each of the prominent problems. Each template can be customized to the specific needs of an application. In this context, a detailed discussion considers important issues and design options.

2 Related work

Micro-workflows [MJ00, Mu99] is an existing architecture for process layers with workflow capabilities. Flexibility and extensibility is achieved by using object-oriented technology. Along the same line, we present an alternative approach that is not necessarily object-oriented. Instead, it leverages the well known blackboard pattern to create a robust and flexible architecture. However, the degree of flexibility and extensibility can be tailored to meet the specific requirements of a concrete application.

The blackboard pattern [Bu96] emerged in the domain of artificial intelligence [Er80, Ni89]. Despite the fact that the roots of the pattern date back to the early 80s, blackboard systems are still subject of more recent research activities [Co03]. Surprisingly, Hunt and Thomas [HT00] seem to be the first who suggested to use a blackboard architecture for the development of workflow capabilities in business applications. However, their work keeps at the surface and lacks a detailed discussion as well as a concrete approach.

“Vortex” [Hu99] is a programming paradigm that also draws on concepts which can be found in a blackboard architecture. It is suitable for a wide-range of decision-making problems, including workflows. Our approach differs in that it concentrates exclusively on process layers for business applications.

3 Introduction to the blackboard pattern

The first system that was based on the blackboard model was the HERSAY-II speech recognition system [Er80]. The term blackboard is a metaphor that represents the core idea of this model: a group of specialists gathers around a blackboard in order to solve a particular problem. Initially, the blackboard contains only the problem description. Specialists write additional information on the blackboard based on their specific knowledge. By combining the information, specialists derive new information and eventually solve the problem.

The blackboard model was conceptualized by Nii [Ni89] who introduced a blackboard framework. The blackboard pattern, as described by Buschmann et al. [Bu96], suggests an architecture for software systems that is based on this framework. A blackboard system which follows this pattern is divided into three major parts [Bu96, Co03].

- The domain knowledge is partitioned into knowledge sources. A knowledge source contributes its particular knowledge to the problem solving process. The contribution takes place by changing or adding data to the blackboard. A knowledge source has an input condition that checks whether the required input data is present on the blackboard. Only if the input condition evaluates to true the knowledge source can be executed.
- The blackboard provides a data structure that contains the problem-solving state. This includes all the data necessary to solve a problem. Knowledge sources can read data from and write data to the blackboard.
- The control is the component which rules the system. Depending on the situation on the blackboard, it selects appropriate knowledge sources and executes them. The control is sometimes called the supervisor [DWK01].

The term "knowledge source" is closely related to the domain of artificial intelligence. It reflects the concept of an expert who contributes knowledge in form of data. In business applications, however, the focus is more on task completion rather than on data. In this context, a knowledge source is an active entity that is responsible for executing a particular piece of work (e.g. sending an email). To better reflect this nature we replace the term "knowledge source" with "*action*".

From a technical point of view, the blackboard pattern is an extension of the shared repository pattern as discussed by Lalanda [La98]. The most important characteristic of this pattern is the emphasis on indirect communication between the components of a system [La98]. For the blackboard pattern this means that actions communicate solely via the blackboard, or more precisely by passing data through the blackboard. An action never calls another action directly.

Indirect communication tends to make systems difficult to understand, especially when a system grows and the number of interactions increases. The problem is that interactions between components cannot be easily inferred from the source code. Writing data to a blackboard is like an event that triggers other actions. But it is not right obvious which actions these are. The information is hidden in the control component as well as in the input conditions of participating actions.

Despite the drawback in understandability, architectures that are based on indirect communication have a number of advantages. The most important one (among others which are discussed in detail in Lalanda [La98] and Buschmann et al. [Bu96]) is low coupling between components. In a blackboard architecture, actions are loosely coupled because they don't interact directly. As a consequence, actions can be added, replaced or removed with minimal impact on other actions.

Unfortunately, there is a second kind of coupling in a blackboard system which plays against flexibility. Actions are coupled through their data structures because they read and process data that is produced by other actions. Data coupling can become problematic especially when multiple actions depend on the same structure. This structure then becomes a fragile part of the system. Changing it means that a potentially large number of actions may be affected. To reduce the effects of data coupling, it is important to use small and focused data structures. The number of actions which depend on a particular structure should be minimized.

3.1 Control plan

The blackboard pattern requires the implementation of a control component which is responsible for the coordination of actions [Bu96]. The control implements a strategy which defines the actions to be executed in a given situation in order to progress towards the goal. The original blackboard pattern suggests developing domain or problem specific controls [Bu96]. This means that a control contains all domain specific logic that is required to select an action. It also means that the strategy is coded into the control. The control becomes the part of a system where everything is "glued" together. If the strategy has to be changed, the control has to be modified. Removal or addition of actions also has an impact on the control which reduces the aforementioned flexibility. If the control becomes a problematic component, Buschmann et al. [Bu96] suggest that the situation can be improved by using the strategy pattern [GHJ95].

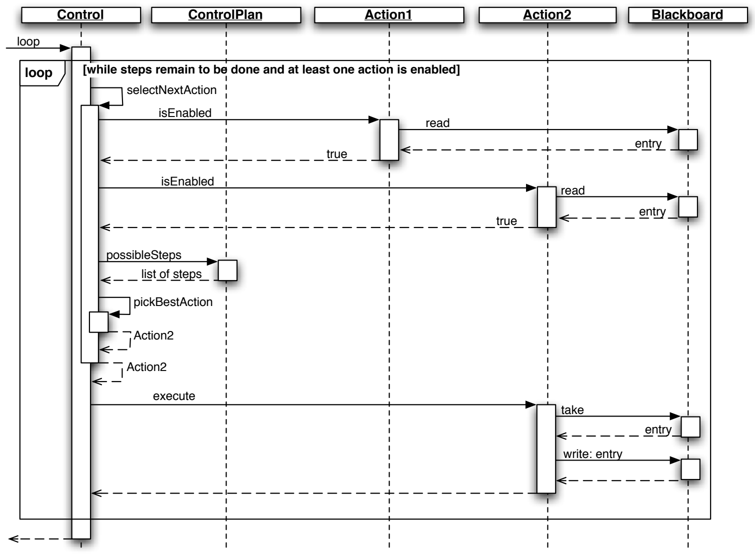

Figure 1: Interactions in a blackboard system (with control plan)

The blackboard based control plan pattern [La97] takes the idea of the strategy pattern further. It complements the blackboard pattern and sets the foundation for highly flexible blackboard systems. A major issue of the control component is the intermixing of two tasks: selection of actions and tracking the status of control flow. The blackboard based control plan addresses this problem by establishing a clear separation of concerns. This is achieved by defining a meta controller that follows the steps in a control plan. The plan defines the steps to be executed to achieve a goal. Each step corresponds to the activation of one or more actions.

4 Building the process layer

In business applications, the process layer is usually employed to implement workflow functionality. The concept of a workflow commonly refers to automated processes that fit a model which consists of two tiers [GT98, Ma00] (figure 2). The business functions which are relevant to a process reside in the work tier. Functionality is split into multiple activities where each activity represents a well-defined task in a workflow. The prominent question for the work tier is how activities are implemented (4.1). Execution and coordination of activities is a responsibility of the flow tier. Two aspects are important here. First, the flow has to be defined, i.e. it must be specified under which conditions an activity has to be executed (4.2). Second, the work tier must execute the flow definition, i.e. perform the activities at appropriate points (4.3).

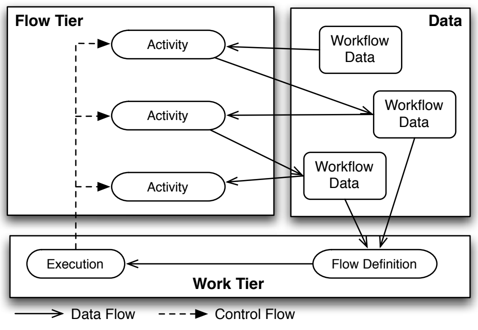

Figure 2: Workflow model with two tiers and data

The data aspect of a workflow is the third viewpoint to be considered (4.4). Both, work and flow tier operate on workflow data. An activity processes certain input data. The result can be output data which in turn is the input data for subsequent activities. In the flow tier, the flow definition may access workflow data in order to determine activities to be executed in a particular situation.

The following paragraphs illustrate and discuss how three main aspects - the work tier, the flow tier and workflow data - can be addressed in a blackboard architecture.

4.1 Activities

Problem: How can we construct an activity that represents a function in a process?

Solution: Implement an action for each individual activity. The input condition of an action checks if the data that is required for the action to work is present on the blackboard. When an action is executed, it reads the necessary input data from the blackboard and writes the result back to the blackboard.

Example: As an example, consider a simple workflow for order processing. A customer places an order for certain products. The availability of the ordered products has to be checked as well as the customer's solvency. Based on the result of these checks, the order is either accepted and processed further or rejected.

Such a workflow will show activities like for example "check availability of products" or "reject order". An action has to be implemented for each of these activities. Let's look at the action for the "check availability of products" activity. The input condition checks if an order exists on the blackboard. When the action is executed, it contacts the warehouse management system to check whether the ordered products are available. The result of the check is written to the blackboard.

Discussion: The most crucial part in the development of actions is probably the input condition. Considering the example of an action for the "reject order" activity, it is tempting to check in the input condition if either the event "products not available" or the event "customer not solvent" happened. This means that the action would not be executable before one of these events occurred. The problem here is that responsibilities from the flow tier are mixed into the work tier. Let's assume the workflow has to be changed. If the ordered products are not available and the customer is a premium customer, a message should be send to the customer service center instead of rejecting the order. As a consequence, the input condition of the "reject order" action has to be modified in order to consider premium customers.

The more complex a change is, the more actions are likely to be affected. The flexibility of a workflow is reduced dramatically. It follows that flow aspects should not be incorporated into actions. Instead think of what an action really needs to work. Is it possible for the "reject order" action to work without the "product not available" event? Yes, because the bare minimum this action needs to work is only the order itself. Consequently, the input condition of the action should check only for the presence of an order. It is up to workflow execution component to decide whether an enabled action is actually executed in a particular situation or not.

Reusability of an action is another aspect when designing its input condition. Actions with rather abstract tasks, such as "send message to ..." may be reused for different activities, even in different workflows. To enable reuse, the input condition must not be tied to a particular state in a workflow. This also affects the data structures which have to be designed for reuse in different scenarios.

Problem: How can we mediate between actions with incompatible data structures.

Solution: Implement a transformation action that converts between the incompatible data structures.

Example: If an order is accepted or rejected, the customer has to be notified via email. An action for sending emails already exists but it does not know anything about orders or customers. Instead, it expects to find an email data structure on the blackboard. This structure contains information like subject, receiver and email body. To reuse this action for the order processing workflow, a transformation action has to be implemented. This action would read the decision about wether the order was accepted or rejected as well as the customer data from the blackboard. It would create a new instance of the email data structure and fill it with the data from the blackboard. Finally, the email data structure is written to the blackboard where it will be found by the email sending action.

Discussion: As discussed in section 3, actions are coupled through their data structures. This is a problem if we want to reuse existing actions in different places of a workflow. In case of the email sending action, it is not feasible to modify the action in such a way that it knows about orders and customers. This would dramatically reduce it's reusability because actions are coupled at the data level. Implementing a transformation action solves the problem without introducing this kind of coupling between the two regular actions.

A transformation action does not correspond to an activity in a workflow. Its sole purpose is to enable actions with incompatible data structures to work together. Transformation actions are registered together with regular actions that implement activities. But unlike the latter, they are not tied to a particular place in a workflow. The workflow execution runs enabled transformation actions before selecting and executing a regular action.

4.2 Defining the flow

Problem: How can we define and track the flow of control in a workflow?

Solution: Implement a control plan that encapsulates the flow. Such a plan defines the steps which have to be carried out during workflow execution. A step basically corresponds to an activity but it does not specify the concrete action to be executed. The workflow execution successively asks the control plan for the next step. In order to determine the next step, the plan can inspect the blackboard or apply other criteria which are specific to the control plan implementation. It is possible that a plan decides that multiple steps are possible in a given situation. In such cases, the workflow execution must decide which step among the possible ones it executes.

Example: Let us assume that a new order has been placed on the blackboard and the control plan is asked what to do next. The plan looks at the blackboard and finds an order that has not been validated (no validation result exist). Therefore, it decides that both of the steps “check availability of products” and “check customer solvency” are possible. Let us further assume that the latter step has been executed and that the solvency check is positive. The plan now finds an order and the result of the solvency check on the blackboard. The result was positive and hence the "check availability of products" step remains to be done. If the solvency check had revealed that the customer is not solvent, the plan would find a negative result on the blackboard. Since orders from customers who are not solvent are not accepted, the plan can decide that the order has to be rejected immediately.

Discussion: The control plan is a strategy that encapsulates the flow aspect of a particular workflow. As a consequence, modifications regarding the flow of work are local to the plan instead of rippling through actions and control. It is even possible to change the whole plan by plugging a different control plan into a workflow instance. Nevertheless it has to be carefully evaluated whether such a degree of flexibility is really necessary for a given system. Changing the plan, in particular at runtime, results in a number of problems which have to be solved. The most challenging issue is how workflow executions which started with the old version of a plan should be handled [KCD99].

If the flow is not pre-determined, either for a complete or for a partial workflow, the plan can be adjusted at runtime. This approach is described in the original pattern [La97] and is based on the idea that a control plan follows a particular goal (e.g. "determine whether an order should be rejected or accepted") . Steps are selected in order to achieve the goal. If the plan is dynamic, the goal may change at runtime based on some workflow-specific criteria. By taking this approach to an extreme, it is possible to implement nondeterministic flows where the order of activities is individual to each workflow execution.

We do not propose a complete and ready-made workflow system and hence it is difficult to suggest how a control plan should be implemented. However, we identify two basic approaches, a rule based and a state based approach. The former determines the possible steps by applying a set of rules against data on the blackboard. From the rules that evaluate to true, the activities which may or have to be executed in a particular situation are inferred. Taking the state based approach, a control plan would maintain an internal state in order to keep track of the workflow. In this scenario, the control plan is basically a state machine. The rule-based approach requires that the state of a workflow can be derived from the workflow data. It shows a good flexibility because adding, changing or removing rules is easy compared to changing a state machine. On the other hand, a state based approach allows precise tracking of the flow. It does, however, require a feedback from the execution component of a workflow. In order to update its state, it has to be notified whenever a step has been completed.

Another important criteria is the complexity of control constructs in a workflow. Van der Aalst et al. [Aa03] identified and documented a number of different control patterns of varying complexity. A control plan must be able to model the patterns which occur in a workflow. Therefore it is recommended to identify the required control constructs before making decisions about the implementation of a control plan. We find the state based solution is better in representing aspects like concurrent execution of steps or complex splits and joins. Dynamic and adaptive flows with complex decisions call for a rule-based implementation. For simple workflows with limited expected changes, on the other hand, a simple approach where the flow is hard-coded in the control plan can be feasible.

4.3 Execution

Problem: Who is responsible for executing a workflow? How can we map actions to steps in a control plan and how can we execute them as well as monitor the execution?

Solution: Implement a control component that can be configured with a control plan and a set of actions. For each execution of a workflow, a new control is instantiated, configured and attached to a blackboard. Whenever the blackboard is modified, it triggers the control which in turn asks the plan for the next steps to be executed. The control plan returns the steps but it does not specify the actions which would execute those steps. Instead, the control tries to find suitable actions by applying a particular strategy. Only actions which are enabled (the input condition evaluates to true) are considered. If no enabled action is found for a given step, it remains uncompleted.

Example: The implementation is straightforward. In fact, this could be a generic control. It simply gets passed the control plan, the actions for our workflow and a strategy that determines one or more actions for a given step. Let's assume that a new order has just arrived and the next steps are "check product availability" and "check customer solvency". The control takes the first step and asks the strategy for a suitable action. A simple strategy could maintain a lookup table which maps the name of a step to the corresponding action. For the "check product availability" step it would probably return an action called something like "CheckProductsAvailableAction". If this action is enabled, it is executed. If it is disabled, the control continues with the next step ("check customer solvency").

Discussion: Instead of establishing a direct connection between control plan and actions, the control plan pattern suggests to decouple these components by establishing a mapping strategy [La97]. Thus the implementation of an activity can be replaced without touching the plan. If the mapping strategy is flexible enough, it may determine the action for a step at runtime by taking certain criteria like "quality of service" into account. This approach not only improves flexibility but also enables interesting techniques like fail-overs between actions.

For an example, consider the "check customer solvency" step. There may be a action that does a live check using an online connection to agencies like the german SCHUFA. However, this check only works when a connection is available, which may not always be the case. So what to do if the connection is not available? A solution is to implement a second action that does, for example, a validation based on some scoring criteria. Now the strategy has to select one of the two actions. It would prefer the online check action and if this action cannot work due to connection problems, it would return the fallback action.

As with the control plan, there is no general implementation technique for a mapping strategy. Lalanda [La97] recommends to establish an equivalence relation between actions and the steps in a plan. Such a relation would express the ability of an action to complete a particular step. A mapping strategy that is based on this idea prefers the action with the highest ability first. If this action is not executable (it's input condition is false), the one with the next-highest ability is selected and so on. However, this is only one possible approach. In some situations, a non-generic, hard-coded mapping strategy that takes other criteria into account (such as in the example before) may work equally well or better.

A second important aspect which has to be mentioned in the context of the control component is related to control flow patterns [Aa03]. As with the control plan, the control structures in a workflow dictate the requirements to the control component. In particular, workflows sometimes require concurrent execution of steps. In many cases it does not really matter whether the steps are performed truly concurrent or one after another. But this is not always feasible, i.e. a concurrent execution may be required, for instance due to performance constraints. In such cases, the control component has to be designed and implemented to cope with this requirements. A detailed investigation of possible techniques to address such a scenario would be helpful and remains to be done.

4.4 Data

Problem: How can we make sure that multiple workflows which are executed at the same time are working on their own set of data? How can we structure the blackboard that multiple concurrent running workflows share data?

Solution: Within the blackboard pattern, the natural place to store data is the blackboard. Actions read data from and write new or modified data to the blackboard. If the blackboard is shared by multiple workflow executions, each one gets its own dedicated zone on the blackboard. Exclusive read and write access to such a dedicated zone is granted to the actions and, if necessary, to the control plan of a workflow execution. The control which executes a workflow is attached to the associated dedicated zone, not to the whole blackboard. Changes which happen in other zones on the same blackboard are not populated to the control. Data which has to be shared between multiple workflow executions can be placed in a shared zone.

Example: When a new order arrives, a new dedicated zone for processing this order is created. The order is written to the zone and a new order processing workflow is started. When the actions for "check customer solvency" and "check product availability" are executed, each one reads the order from the zone and writes the result back to the same zone. The next action (either for the activity "reject order" or "accept order") reads the order as well as the results and does it's work, for example initiate the delivery of products. When the workflow execution has finished, the whole zone, and with it the data, is removed from the blackboard.

Let's now assume that the fallback action for solvency checks needs access to a scoring table. The table is rather static and seldom updated. Of course all executions of the order processing workflow need to access the table, so placing it into the individual zones is not a good idea. A solution is to create a shared zone on the blackboard where the table is stored. Read access to the shared zone is granted to all workflow executions in order to allow participating actions to access the table.

Discussion: Structuring the blackboard according to the problem at hand is a common approach in blackboard applications [Bu96, Co03]. Possible data structures range from trees to flat partitions [Co03]. The idea of zones is mentioned by Hunt and Thomas [HT00] and is well suited to the problem domain of workflows in business applications. In our approach, a zone is an isolated area on a blackboard where data objects can be stored. A given data object can exist only in one zone. Access to the blackboard can be controlled at the zone level, i.e. a workflow execution gets access only to those zones which contain data relevant to this workflow.

Zones are used to model different levels of data visibility, or scopes. We mentioned two scopes here: data that is visible to an individual workflow execution and data that is shared among a number of executions. Russel et al. [Ru04] identified seven different levels of visibility which are common in workflows. A complete evaluation how these levels can be implemented within our approach remains to be done. Nevertheless do zones provide a working solution to the problem of data visibility.

5 Conclusion

After a basic introduction of the blackboard pattern and the complementary blackboard based control plan, we identified the prominent building blocks of a process layer with workflow capabilities. Separation in work- and flow tier provided a basic structure. Workflow data is a crosscutting aspect which affects both tiers. We illustrated how the tasks emerging from this structure can be addressed in a blackboard architecture. Important design decisions and consequences have been discussed. The proposed approach is a lightweight solution for constructing a process layers with workflow capabilities. It provides a micro-architectural template for custom development of such a layer when using a fully-fledged workflow management system is not feasible. The approach leverages the blackboard pattern to create a flexible solution that can be tailored to the specific needs of a system.

In order to validate the applicability of our approach for process layers in a wide range of business applications, it is necessary to show that commonly used control flow constructs as well as data scopes can be implemented. In this context, an evaluation of the approach against workflow control and data patterns [Aa03, Ru04] would be helpful and remains to be done.

The open-source OpenBBS framework [Op06] is an implementation of our approach. The framework was derived from a study that investigated possible uses of the blackboard pattern in business applications. Despite being in an early stage, it already provides the essential parts required to develop blackboard based process layers. Future development will add the missing bits and complement the framework with additional generic control plan implementations. OpenBBS is currently used to implement a customer registration workflow in a B2B online trading application. Further work is done to investigate if and how the core trading part of the application can be switched to a blackboard architecture.

References

[Aa03] van der Aalst, W. M. P.; ter Hofstede, A. H. M.; Kiepuszewski, B.; Barros, A. P.: Workflow Patterns. In: Distributed and Parallel Databases, 14(2003) 3, p.5-51.

[Bu96] Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.: Pattern-oriented software architecture - a system of patterns. Wiley, Chichester, 1996.

[Co03] Corkill, D. D.: Collaborating Software: Blackboard and Multi-Agent Systems & the Future. In: International Conference on Information Fusion (Fusion 2005), Philadelphia, Pennsylvania, 2005,

[DWK01] Deugo, D.; Weiss, M.; Kendall, E.: Reusable Patterns for Agent Coordination. In: A. Omicini; F. Zambonelli; M. Klusch; R. Tolksdorf (Eds.): Coordination of Internet Agents - Models, Technology, and Applications. Springer, Berlin, 2001.

[Er80] Erman, L. D.; Hayes-Roth, F.; Lesser, V. R.; Reddy, R. D.: The Hersay-II speech understanding system: Integrating knowledge to resolve uncertainty. In: ACM Computing Survey, 12(1980) p.213-253.

[GHJ95] Gamma, E.; Helm, R.; Johnson, R. E.: Design patterns: Elements of reusable object-oriented software. Addison-Wesley, Reading, Mass., 1995.

[GT98] Georgakopoulos, D.; Tsalgatidou, A.: Technology and Tools for Comprehensive Business Process Lifecycle Management. In: NATO Advanced Study Institute on Workflow Management Systems, 1998, p.324-363.

[Hu99] Hull, R.; Llirbat, F.; Simon, E.; Jianwen, S.; Dong, G.; Kumar, B.; et al.: Declarative workflows that support easy modification and dynamic browsing. In: International Conference on Work activities Coordination and Collaboration, San Francisco, 1999, p.69-78.

[HT00] Hunt, A.; Thomas, D.: The Pragmatic Programmer: From Journeyman to Master. Addision-Wesley, Boston, 2000.

[KCD99] Koksal, P.; Cingil, I.; Dogac, A.: A Component-based Workflow System with Dynamic Modifications. In: Next Generation Information Technologies and Systems, 1999, p.238-255.

[La97] Lalanda, P.: Two complementary patterns to build multi-expert systems. In: Patterns Languages of Programs, Monticello, Illinois, 1997.

[La98] Lalanda, P.: Shared repository pattern. In: Patterns Languages of Programs, Monticello, Illinois, 1998.

[Ma00] Manolescu, D. A.: Micro-Workflow: A workflow architecture supporting compositional object-oriented software development. Ph.D. Thesis. University of Illinois, Urbana, 2000.

[MJ00] Manolescu, D. A.; Johnson, R. E.: A micro-workflow component for federated workflow. In: OOPSLA2000 Workshop on Implementation and Application of Object-Oriented Workflow Management Systems III, Minnesota, 2000.

[Mu99] Muth, P.; Weissenfels, J.; Gillman, M.; Weikum, G.: Integrating Light-Weight Workflow Management Systems within Existing Business Environments. In: 15th International Conference on Data Engineering, Sydney, 1999, p.286-293.

[Ni89] Nii, H. P.: Blackboard Systems. In: A. Barr; P. Cohen; E. A. Feigebaum (Eds.): Handbook of Artificial Intelligence. Addison-Wesley, Boston, 1989.

[Op06] The OpenBBS Framework. Online. Availabel at http://openbbs.sourceforge.net/, accessed on 2006/11/02.

[Ru04] Workflow Data Patterns. Online. Available at http://is.tm.tue.nl/research/patterns/documentation.htm, accessed on 2006/08/22.