Agentworkflows for Flexible Workflow Execution

Thomas Wagner

University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences,
Department of Informatics, Theoretical Foundations of Computer Science Group
http://www.informatik.uni-hamburg.de/TGI/

Abstract. Dynamic aspects of workflow execution require flexible solutions. Especially in an interorganisational context many variable factors can only be determined during the actual execution of a workflow. These factors may require contextual, local changes within a process in order to adequately support and represent the real-world scenario. This paper describes the *agentworkflow* approach, which uses a combination of the agent and workflow concepts to address a number of challenges of workflow execution. Both agents and workflows are provided as high-level Petri nets.
The focus of this paper is the flexibility aspect of this approach. Agentworkflows allow the dynamic reconfiguration of the workflow specification. The key to this is the exchange of subworkflows depending on the changing circumstances of the workflow execution. This allows the workflow system to adapt to these circumstances and support users adequately.

Keywords: Workflows, Agents, Combination, Flexibility

1 Introduction

Business processes (BP) are becoming ever more complex, especially in large organisations. Their correct execution is crucial to the successful operation of a company. Any incident occurring due to errors in a BP needs to be avoided by all means. This is why BP are commonly facilitated with the help of workflow software systems. Workflow management systems (WFMS) map real-life BP to computerised representations and manage and control their execution, while automating the processes as far as possible and supporting human users during task execution.

Classically, WFMS are aimed at supporting static processes, that rarely change. Dynamic changes, which might become necessary on a case-by-case basis, are difficult to handle and often require different solutions for every situation. This is, however, a static approach to a dynamic problem and rather inefficient, especially if the required changes are only minor and affect very small parts of the workflow. Unforeseen changes that develop out of unique circumstances cannot easily be handled by a static WFMS. These changes cannot be handled on-the-fly and require a workflow modeller to implement a new version of the

overall workflow, especially dealing with the specific problem. The new workflow then has to be re-initiated and an equivalent to the previous workflows' state has to be established. This requires time and effort, which, in real-life situations, may be in short supply.

Because of this, another fundamental approach to workflow management can be helpful or even crucial. Instead of only supporting static workflows, a degree of flexibility should be added to the workflow management. It should be possible to exchange certain parts of a workflow during execution, depending on the current state of the system and workflow. If unforeseen changes occur, eligible workflow administrators and modellers need to be able to simply (re-)design the relevant parts and infuse them into the system in order to make further execution of the workflow possible. Standardised exception handling or skipping patterns can be used to automatically handle occurring issues and simplify the work of administrators.

Another important aspect relates to interorganisational workflows, i.e. workflows executed cooperatively between different organisational entities. In this context flexibility becomes especially important if partners in a workflow often change, which would require differing workflows each time in classical approaches. Interorganisational aspects will feature heavily in the discussion later on in this paper.

We propose an approach to flexibility in workflow execution that strongly relies on and profits from the agent-oriented paradigm. By using software agents we can exploit the naturally distributed nature of these software entities to profit in various ways. The approach, called *agentworkflows*, uses one agent for each workflow instance. This agent is responsible for the execution, handling and distribution of this workflow. The workflows themselves feature a hierarchical structure, utilising subworkflows nested in an overall workflow. These subworkflows are essential to the support of flexibility and will receive special attention in this paper.

This paper is structured as follows. In Section 2 we will discuss related work. Section 3 highlights the modelling background for our research. Section 4 then presents the agentworkflow concept and implementation, while Section 5 discusses the flexibility aspects of agentworkflows. Both agentworkflow concept and how it can be used for flexibility are illustrated in an simple example in Section 6, followed by the conclusion of the paper in Section 7.

2 Related Work

In this section we will discuss other approaches to flexibility in workflow management. The ADEPT (Application Development based on Encapsulated premodeled Process Templates) project described, for example, in [4] deals with flexible and robust workflow management. Using an advanced meta-model, which defines all possible, allowed process structures, the system developed in this project allows users to change single process instances or whole process templates. Most aspects of a process or template can be changed during run-time, as long as the

changes are allowed by the meta-model. Updating a process template updates all of its currently running instances, as long as the changes do not produce inconsistencies. The key to the flexibility in ADEPT is the so-called delta-layer. This layer is situated between process template data and process instance data and enables changes made to single instances. If the template is changed, the information in the delta-layer and process instance data are checked against the updated template and the decision whether to migrate or not is made. Compared to our approach, the agentworkflows, ADEPT offers a much higher degree of flexibility, but is very focussed on the processes. Agentworkflows add qualities to the workflow execution, which are usually only associated with agents. These go beyond the flexibility, which is the focus of *this* paper.

Another approach is presented in [6]. Similar to our approach the JBees WFMS is implemented using agents and Petri nets. Based on the agent-platform Opal and the Petri net tool JFern, the JBees system consists of several types of agents. Interactions between these agents enable the execution of workflows. Of special interest is the process agent, which similarly to our approach, encapsulates a process instance and is responsible for its execution. Workflow flexibility is also available in JBees using different algorithms to determine safe transfers between process instances. While JBees uses agents and Petri nets to implement workflow management it does not seem to offer the integration between agents and workflows we strife to achieve with our overall approach. The agentworkflows partially fulfil our goals and can be compared in scope to a WFMS like JBees.

[1] proposes a smart WFMS. During execution the WFMS possesses context awareness for a process, so that it can adapt to the current circumstances. The conceptual framework adds two key components to an otherwise regular WFMS: smart workflow descriptions and a context reasoner. The smart workflow descriptions contain generic activities that are only linked to particular tasks during runtime, similar to our approach. The context reasoner is responsible for evaluating current circumstances and linking the generic activities to tasks.

[3, 2] propose recursive ECATNets (Extended Concurrent Algebraic Term Nets) to model hierarchical and flexible workflows. Similar to our approach they differentiate between elementary tasks, which are directly executed by resources, and abstract tasks, which correspond to subworkflows.

Both the smart WFMS and the recursive ECATNets share similarities to how flexibility is handled (e.g. late coupling of tasks, representing subworkflows as tasks in overall workflows). However both approaches do not utilise agents. As mentioned before agentworkflows can benefit in more ways then flexibility from the employed agents.

All approaches, which have been discussed here, deal with flexibility in workflow execution in different and effective ways. Our approach exhibits similarities to certain aspects of these approaches, like the use of Petri nets and agent-orientation. In its current stage our approach may not offer the distinguished and elaborated possibilities offered by the other approaches. However, it is just one of the stepping stones toward a full integration of agents and workflows as

proposed in [11]. As such, the possibilities, w.r.t flexibility and other properties, in later stages will be significantly improved yet again, though this is outside the scope of this paper (see [17] for more information).

3 Modelling Background

The modelling background for our approach contains two major areas: agents and workflows. Agents are provided through the Mulan and Capa agent architectures ([12, 5]). Mulan is a conceptual agent framework/architecture based entirely on reference nets, a high level Petri net formalism introduced in [8]. Every aspect of a multi-agent system in Mulan, from agent protocols to the overall systems, is modelled in reference nets. As the reference net formalism follows the nets-within-nets principle [13], each layer is nested within its upper layer creating a four-level hierarchy. Capa is an extension to Mulan introducing full FIPA compliance to Mulan and replacing the upper levels of the Mulan hierarchy. This provides the functionality to allow distributed execution.

Workflows on the other hand are provided through workflow (Petri) nets. In our implementation workflow nets are specialised reference nets using a special transition to model the tasks of the process. These workflow nets were introduced in [7]. They follow the basic principles of (coloured) workflow Petri nets described in [14].

Since both agents and workflows have a common technological base, the reference net formalism, an integration of both concepts is not only possible in our approach, it is also quite natural. Both concepts can profit from one another, though the focus of this particular paper is on workflows benefiting from agent technology. The overall aspects of the work on integrating agents and workflows has been the subject of, for example, [11, 16, 10, 17].

Mulan/Capa and workflow nets have previously been used to implement WFMS functionality. In [15] an agent-based WFMS (AgWFMS) was presented, which provides full workflow functionality using the above mentioned technologies. The functionality required is naturally divided between several types of agents. Users can log into the system remotely, but workflows are executed centralised. The AgWFMS managed to capture and support many interesting aspects, like interoperability due to the FIPA compliance of Capa. Some aspects, however, such as distribution and flexibility of workflows, could not be adequately supported in the classic AgWFMS. This was one of the motivations to extend the system with our new approach.

The development and runtime environment for our systems is the Renew (**Re**ference **Ne**t **W**orkshop) editor. The editor was developed alongside the reference net formalism and is described, for example, in [9]. It supports the execution of all aspects described in this paper.

4 Agentworkflows

Our approach to flexible workflow management is called *agentworkflows*. This name reflects the combination of the agent and workflow concepts in this approach. The approach was originally described in [16] and general properties were discussed in [10, 17]. It is part of a larger, ongoing effort, also described in the previously mentioned works, but originating in [11]. This effort aims to combine agents and workflows into a new concept that exhibits the advantages and characteristics of both classical concepts. The overall goal in this is to address some of the shortcomings each classical concept (can) exhibit. The agentworkflow approach represents one of the later steps in the overall effort, that integrates both agents and workflows conceptually in the background, but still exhibits classical workflow behaviour to its environment.

The approach is based on the classic, regular AgWFMS, mentioned above. The agentworkflow approach replaces part of the workflow handling process in the AgWFMS and can thus be seen as a natural extension. The extended AgWFMS containing the enhanced functionality of and for agentworkflows is called AgWFMS*.

The basic principle behind our agentworkflow approach is that of hierarchical nested subworkflows. In short a workflow basically consists of a number of subworkflows, which are orchestrated in an overall workflow, called the *structure-workflow*. The relation between structure-workflow and subworkflows is handled through the tasks of the structure-workflow. Tasks in the structure-workflow correspond to subworkflows. Subworkflows can, conceptually, be other structure-workflows also containing further subworkflows. However, for readability and simplicity we will restrict ourselves to a two-level hierarchy in this paper. This means that subworkflows consist only of tasks being executed by workflow resources (human or automated). In other words, the structure-workflow defines the basic outline and connection (the structure) between the different subworkflows. One key aspect of the development of this approach was to allow distributed workflow execution. For the agentworkflow approach in particular this manifests itself in the subworkflows. Each subworkflow is independent from the others and can be executed on a different registered system in the network, depending on the requirements of the particular subworkflow. Since the data- and control-flow of the workflow is handled within the structure-workflow the different subworkflows can be executed independently and only need to be coordinated at their beginning and end.

One designated agent, the so-called *structure-agent*, is responsible for the execution of the structure-workflow and the aforementioned distribution aspect. For each new workflow instance a new structure-agent is instantiated. This structure-agent is only responsible for his own agentworkflow instance. The structure-workflow is part of the structure agent. In fact, the agent is not only responsible for the execution of the structure-workflow, it even handles most of it itself. For this (and further autonomy reasons) it contains and replicates some parts of the AgWFMS functionality.

The system functions as follows: When the structure-agent is started it receives the structure-workflow definition from the AgWFMS*. It instantiates and stores the workflow net internally and registers as a listener for this net (and this net only). This way, when tasks become activated the structure-agent can automatically react. When that happens it reads the description of the subworkflow directly from the task of the structure-workflow. From this description the structure-agent determines the circumstances for the subworkflow execution. It determines what kind of or what particular system this subworkflow needs to be executed on and then inquires which of these eligible systems are currently online and registered. It then proceeds to choose one of the (possibly) multiple systems for the actual execution of the subworkflow and contacts the interface agent of the chosen AgWFMS*. After authentication the interface agent can decide to accept or reject the subworkflow. If it rejects the subworkflow the structure-agent chooses another system and tries instantiation of the subworkflow again, unless no suitable systems are found, in which case error handling must occur[1]. If it accepts the subworkflow, (optional) input data is sent from the structure-agent and the subworkflow is instantiated locally at that AgWFMS*. This point is crucial to the flexibility aspect, since only the designation of the local subworkflow needs to be known to the structure-agent. The actual implementation of the subworkflow is, except for input and output data, independent from the structure-workflow. This will be discussed in the next section. The subworkflow is executed by the local[2] resources logged into that AgWFMS*. When the subworkflow has finished its execution the (optional) output data is transferred, along with the confirmation of the subworkflows success, from the AgWFMS* to the original structure-agent. The structure-agent then takes this information for the structure-workflow, which completes its own task-transition. This process is repeated for all subworkflows that become active during the execution of the structure-workflow. Once the structure-workflow reaches the final transition the workflow is finished and the structure-agent can persistently store the data and then terminate.

Figure 1 shows a snapshot of an execution of an exemplary agentworkflow. The structure-workflow is being executed by the structure-agent on AgWFMS*1. It contains two active tasks/subworkflows that are currently being executed on two AgWFMS*. AgWFMS*1 is home to the structure-agent and is executing subworkflow A, while AgWFMS*2 is executing subworkflow B. The figure clearly shows the relations between the agents and the workflows. The structure-workflow is only executed by the structure-agent. The structure-agent is in communication with the AgWFMS* agents to oversee subworkflow execution. The subworkflows are in no way executed by the structure-agent, but only correspond to tasks in the structure-workflow. The only agents involved in the actual execution of the subworkflows are the ones of the local AgWFMS* systems.

[1] This could, for example, include the search for alternate systems or require manual input from a user.

[2] *Local* in this context refers to the resources logged into this particular AgWFMS* and does not exclude resources connected to the AgWFMS* through a network.

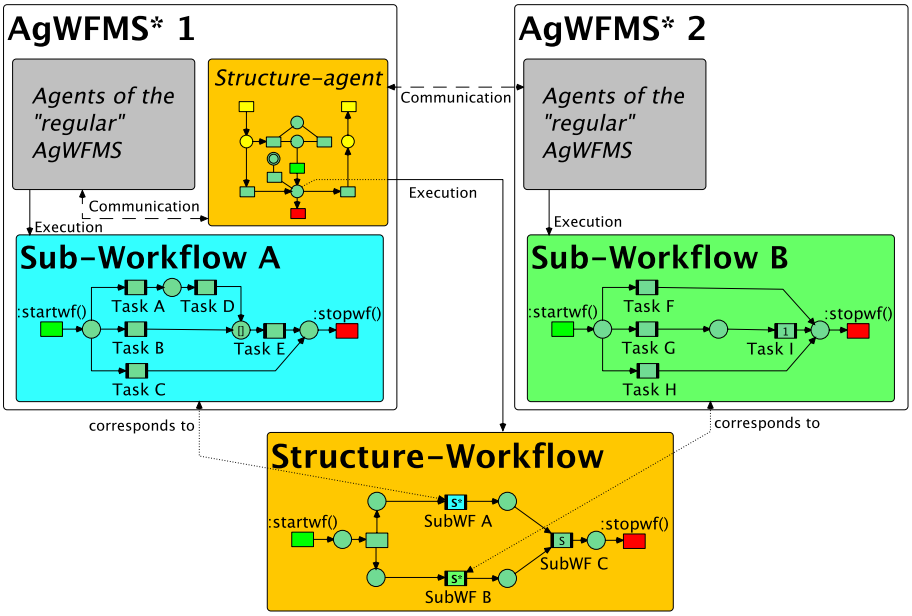

Fig. 1. Principle agentworkflow approach (from [17])

To conclude this description of the agentworkflow principle and AgWFMS* system we will now shortly discuss its general attributes. This will not include the flexibility aspect, which will be discussed in the next section especially dedicated to this aspect. One of the most prominent features of the agentworkflow approach is the clear encapsulation of a workflow instance through a software agent. It provides the workflow instances with a very clear identity during its execution. This can be advantageous for monitoring and maintenance of the system. A disadvantageous aspect of the encapsulation is, that it increases the number of agents active within the system which can lead to performance issues. Furthermore if the structure-agent or its agent platform is terminated erroneously, the entire agentworkflow is lost. This is a problem we aim to fix in the future.

Additionally the encapsulation opens up many of the possibilities of software agents for workflow instances. Since, logically, the structure-agent and the workflow can be seen as equivalent many attributes usually associated with agents can be related to the workflow. This is further supported by the fact, that both the workflows and agents rely on the same technical background: reference nets. Both the approach and the technological implementation add to the integration of agent and workflow principles as proposed by the overall effort in [11].

The most obvious possibilities opened up by the integration are the distribution of workflow execution and interoperability. Since now all parts, including workflow instances, are implemented as agents, they can be distributed almost arbitrarily on the network. Also, since Capa adheres to the FIPA standards,

interoperability with other agent systems is guaranteed, as long as the interfaces match up. In the future, further agent attributes can be added to the agentworkflow approach. Mobility for one matter can remedy some issues arising from the fact that the structure-agent serves as a central point of execution. In the original agentworkflow approach a AgWFMS* platform would have to stay online for as long as the structure-agent was active. If the structure-agent was able to migrate to other platforms, the AgWFMS* platform could shut down while the structure-agent continued its work on another AgWFMS* in the network. This could also be used in error handling situations.

Furthermore, intelligence and autonomy could be added to the structure-agent. These two attributes could be used in various ways, for example resource access or control. But in combination with the mobility and distribution aspects this becomes even more interesting, since it is possible to create an intelligent, adaptive, migrating workflow instance. This is, however, outside of the scope of this paper.

5 Flexibility in Agentworkflows

In this section we will discuss the agentworkflows with regards to their flexibility aspects. As other attributes, such as mobility and intelligence, have already shortly been discussed before, we will not address these here.

The most crucial aspect of the agentworkflow approach with regards to flexibility is obviously the exchange of subworkflows and the loose coupling between structure-workflows and their related subworkflows. In the local view of the structure-agent (and as such, the structure-workflow) each subworkflow possesses only a name, a place to be executed at and a (possibly empty) input and output. The internal workings of a subworkflow are completely transparent to the structure-agent. In fact, they are completely irrelevant to the structure-workflow, as long as the subworkflow is completely executed and the correct output, w.r.t. types, is produced. As a simple example imagine a subworkflow dealing with checking the validity of a document nested within a structure-workflow for a bank. As input, this subworkflow would possess the document, the output could be a report on the document. How and in which order the different attributes of the document are checked is of no concern to the structure-workflow[3], which just needs the report to continue its execution.

This key characteristic of agentworkflows can positively influence flexibility in workflow execution in a number of ways. First and foremost, it can enable the dynamic reconfiguration of the workflow specification. The simplest way to support this is to use a variable for the name of the subworkflow in the structure-workflow, instead of a constant identifier. This variable can depend on various factors, like results of previous subworkflows, and implicitly[4] represents the current circumstances of the workflow execution. When the task/subworkflow be-

[3] Of course it is of concern for the real-world application, but for this generalisation we can abstract from this.

[4] The variable has to be an identifier for the subworkflow.

comes activated now it is instantiated with an identifier depending on the current circumstances of the workflow execution. In this way, the structure-workflow can dynamically adapt to external factors of the execution. From a technical point of view, this is relatively easy to realise, since the unification algorithm of Renew allows for such substitutions. The idea can even be extended to the target AgWFMS* of the subworkflow. Depending on certain factors it might be necessary to not simply execute another subworkflow, but to also change the location. This can be achieved, pretty much in the same way it was done for the subworkflow identifier, although the computations might be a bit more complex. One might also consider extending this principle to input and output. This is however, more difficult, since input and output are usually quite closely tied to the subworkflow. Changing them in accordance to changing the subworkflow is possible (input and output are just variables anyway), but changing them independently would require the target platform to provide subworkflows with equal names and differing parameters. This would make the system more difficult to maintain, administrate and monitor.

It should be noted though, that the degree of flexibility added by this is not as high as could be desired. The subworkflows have to be known prior to workflow execution, so that they are available and compiled during runtime. This implies that unforeseen situations cannot be handled by our current approach. However, the particular flexibility added by the agentworkflows still enhances the system to deal with predictable, dynamic situations. This, in combinations with standardised error-handling mechanisms (e.g. subworkflows that involve administrators), can handle many practical scenarios. Furthermore, it is already quite easy to exchange or add subworkflows in the system. If unforeseen situations arise, adapted subworkflows could be modelled by a workflow modeller and introduced into the system. Making this functionality available more "on-the-fly" and integrating it more directly into the approach could remedy the current shortcomings in flexibility.

Though not implied by the "regular" agentworkflow approach it is also possible to modify the AgWFMS* in order to support the flexibility mechanic from its side. In principle, this is mostly equivalent to the variable identifier version described above, only that the variables depends on factors of the target AgWFMS* **and** the structure-agent. In this modification, the AgWFMS* can autonomously decide which subworkflow to instantiate. It can take the *proposed* subworkflow, the input and the output, as well as its own state into consideration in this decision. The effect would generally be the same, only that the ultimate control would lie with the target AgWFMS*. The combination of both approaches would yield an even higher degree of flexibility, since all factors local to the structure-agent *and* AgWFMS* would be taken into consideration during subworkflow instantiation.

There are some further aspects the subworkflow characteristic adds to workflow execution with regards to flexibility, though these do not impact as much as the ones described above and also focus on other variations of flexibility. Since the structure-agent is ultimately responsible for choosing which AgWFMS* should

execute a subworkflow, an adapted structure-agent with reasoning functionality and additional information about the different AgWFMS* available could enhance flexibility in regards to load balancing or other similar efficiency factors. The agent could, for example choose (among the set of suitable AgWFMS*) the system with the lowest workload or the one with the best network connection. The counterpart to this mechanic is to include the reasoning within the AgWFMS*, which yields similar results. If the AgWFMS* determines that its workload is too high, or that too few resources are available, it can reject the subworkflow, which would force the structure-agent to choose another AgWFMS*.

The subworkflow hierarchy also contributes to flexibility. The two-level hierarchy discussed in this paper is only the simplest and easiest to describe version of the agentworkflow approach. When considering workflow hierarchies of more then two levels, each additional level offers a more fine-grained degree of flexibility. This can be especially useful in an interorganisational context, since the additional levels of organisation can profit there.

Interorganisational workflow execution deals with workflows that are executed cooperatively by different organisations. In this context each organisation is responsible for its own part of the workflow. Even though the organisations are working together within the process, each organisation is, of course, interested in keeping the confidential details from the other partners. Furthermore, each organisation will prefer to use their own WFMS, instead of relying on a central one which is used by all partners. For these reasons the agentworkflow approach is well suited for the interorganisational context, since the encapsulation ensures security and the interoperability and distribution aspects support the second requirement.

Nevertheless the flexibility aspects discussed above are also quite effective in this context. First and foremost, the ability to exchange subworkflows without influencing the overall structure-workflow is even more useful. In an interorganisational setting, factors that influence workflow execution can be even more varied, dynamic and demanding, since the collaboration introduces a lot of complexity in the real-world scenario. Being able to handle these factors by providing different subworkflows for different examples reduces the complexity again.

As mentioned before, adding additional levels to the hierarchy can also be used as an improvement in this context. By allowing subworkflows to be structure-workflows again, the flexibility aspects available on the interorganisational level (original structure-workflow) become available to the individual organisation as well. If the individual organisation uses this hierarchy-level to distribute work between different departments, additional levels would again open up these possibilities to the departments and so forth.

The structure-agents ability to choose a WFMS for subworkflow execution could take on a fully different role in the interorganisational context. By using negotiation concepts, different WFMS of different organisations could offer to take over subworkflows, which in the real-world scenario would translate to organisations competing for work assignments/contracts. This is, however, outside of the scope of the current agentworkflow approach.

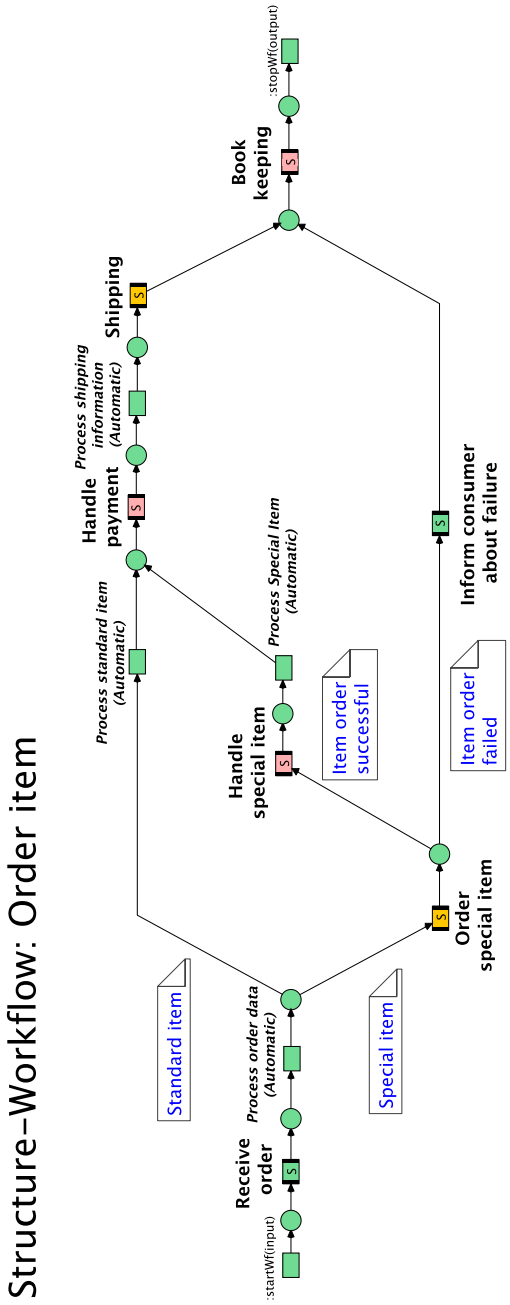

Fig. 2. Example Structure-Workflow

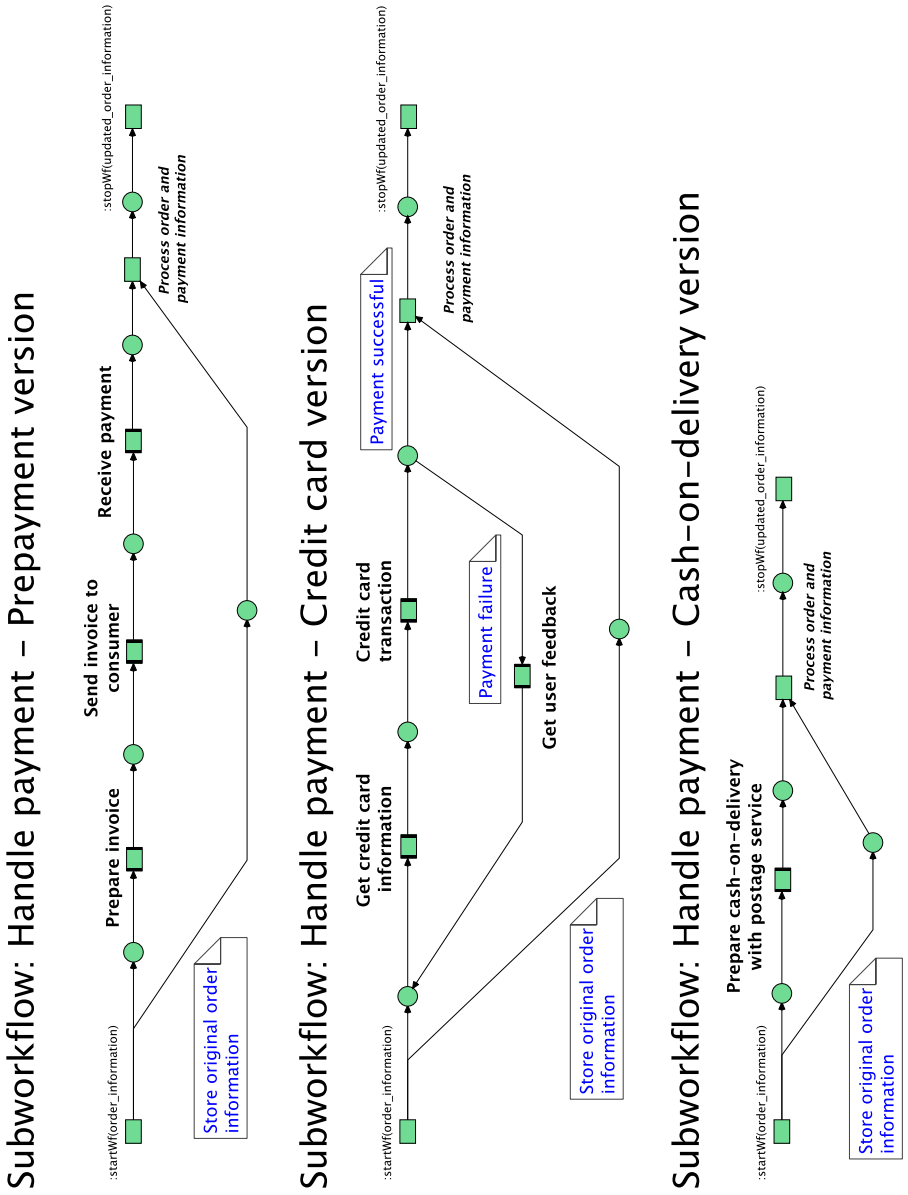

Fig. 3. Example Subworkflows

6 Example

In this section we will discuss a simple example of a flexible agentworkflow in an interorganisational setting. The structure-workflow can be seen in Figure 2. Workflow nets like these consist of regular Petri net places and transitions, as well as a few special elements. A workflow net starts with a transition connected to a synchronous channel labeled *:startWf(input)* over which it can receive parameters, and ends with a transition connected to synchronous channel labeled *:stopWf(output)* over which an optional result can be send. Tasks are represented as transitions with thick vertical bars, called task-transitions. Tasks in a structure-workflow representing subworkflows are also drawn as transitions with thick vertical bars and the letter S marked in the centre. The "regular" Petri net transitions within the figure represent (abstract) operations on the variables and data within the workflow. Usually they would/could feature more complex net structures as well as synchronous channels to receive outside information. However, to retain readability of the net for this paper we have chosen to abstract from a detailed and technical view in favour of a simplified version. For the same reason we omitted the exact, complex inscriptions on the example workflow nets. It should also be noted that, in order to keep the net size and complexity manageable error handling and aborting the workflow due to failures in subworkflows have largely been omitted.

Figure 2 represents a workflow for processing and handling an incoming order in a generic company offering many items. The company offers standard items, which are in stock and don't need special treatment, and special items, which have to be ordered from a third-party provider and handled differently (e.g. large items or large quantities of items). The workflow encompasses the different steps from processing the incoming order, handling the standard or special item, handling payment, shipping via another third-party and finally book keeping. Each of these complex tasks is modelled as a subworkflow. It should be noted that we consider the main company to be in charge of this workflow, so that only the two subworkflows for the third-party providers feature distributed execution. We can observe three different types of subworkflows in this example.

The first type are regular subworkflows, which do not exhibit or require flexibility. In this example these are the processing of the original order (*Receive order*) and informing the consumer about problems in ordering a special item (*Inform consumer about failure*). These subworkflows do not need to be flexible, since, given the scenario, one version for each subworkflow can handle the possible circumstances. However, changing these subworkflows to be flexible would only require adding some variable processing ahead of them, changing the inscriptions to support the variables and providing the different subworkflows on the systems they would need to be executed on.

The second type of subworkflows are the flexible, interorganisational ones. These are ordering a special item (*Order special item*) and handling the shipping (*Shipping*). These are flexible since, depending on the item, they might have to be taken care of by different companies and through different subworkflows. For example ordering a bulky, large household item like a refrigerator would

require a different company and subworkflow then ordering a smartphone. These subworkflows illustrate how agentworkflows in general and their flexibility in particular can support interorganisational settings.

The third and final type of subworkflows are flexible, but local ones. These are handling a special item which has been ordered and received and now needs to be temporarily stored (*Handle special item*), the handling of the payment for a consumer (*Handle payment*) and book keeping and accounting (*Book keeping*). These are flexible since special items may have special requirements in storage or handling and the company might offer different ways of paying for an item.

Three different versions of the *Handle payment* subworkflow are shown in Figure 3. The three versions all represent the process involved in handling and receiving payment from the user. All three have in common that the original order information is stored for later processing with the last task before the subworkflow is finished (the lower branch in all three versions).

The topmost version represents prepayment by the user. The company prepares the invoice, sends it out and, at some later point, receives payment before the item is shipped. The middle version supports payment by credit card. The company receives credit card information from the consumer and executes the transaction. At this point the transaction was either successful or failed. If it failed user interaction (e.g. re-entering the credit card information) is required. If the transaction succeeds the item can be processed and shipped. The lowest version models payment by cash-on-delivery. In this case the cash-on-delivery only needs to be prepared with the postage service before the item can be shipped. In this case processing payment would be included in later stages of the overall workflow (e.g. a corresponding version of the *Book keeping* subworkflow).

This example illustrates the kind of scenario for which agentworkflow flexibility is especially suited. The different possibilities are known beforehand (e.g. the different payment options offered by the organisation) and each can be modelled accordingly. During execution the correct subworkflow can be instantiated and the requirements given by the variable factors of the workflow (in this simple example the choice of payment) can be fulfilled. While the different versions of the *Handle payment* subworkflow only differ in small parts, other scenarios could require more substantial changes in subworkflows. This could also easily be handled by the agentworkflow approach.

Though the workflow of Figure 2 is a simple example, it serves to illustrate the agentworkflow approach quite well. Subworkflows can feature distribution and flexibility, and it is also conceivable to mix subworkflows and regular tasks to loosen the hierarchy. If further aspects of agentworkflows, like intelligence and mobility, are considered, it becomes clear that even these already versatile ways only scratch the surface of the overall approach and its possibilities.

7 Conclusion

In this paper we have presented an approach to workflow management, called agentworkflows. It incorporates elements of both agent orientation and classic

workflow execution to combine strengths of both fields. The agentworkflow approach exhibits many interesting attributes, like encapsulation, intelligence and distribution. The focus of this paper though, was on the flexibility introduced by it. The flexibility of the agentworkflows relies on using a hierarchy of workflows and subworkflows and on allowing the dynamic exchange of subworkflows dependent on variable factors. This enables the dynamic reconfiguration of workflow instances at runtime. We discussed this aspect of the approach in detail and finally gave an example of how it could be used in an interorganisational context. The example illustrated the versatile ways, in which agentworkflows and their subworkflows could be deployed.

The flexibility introduced by the agentworkflow approach makes it more suitable for real-world scenarios than a classically rigid approach. However, there are currently some shortcomings to our approach, since subworkflows need to be known and compiled before execution of the overall structure-workflow. This limits the possibilities of the approach, since on-the-fly changes become difficult. These limitations, however, do not relate to the general approach and need to be fixed on a technological level, rather then a conceptual one. Addressing them is one of our goals for future work. Furthermore we also aim to address the other flexibility aspects discussed in this paper, as well as generally extend the agentworkflow approach with more concepts from both agents and workflows. Enhanced agent mobility, intelligence and distribution can greatly improve workflow execution and the process view given by workflows can enhance the agent-side. We hope to combine the two paradigms to profit from one another and further our overall goal to provide the desired complete integration of both. Ultimately, the intent is to develop a novel, general unit concept, which can serve as agent, workflow or both, depending on the dynamic requirements at runtime. Agentworkflows are one of the later steps towards that goal.

In conclusion, the agentworkflow approach possesses many qualities beneficial to flexible workflow execution. Tt serves as an important basis for future work regarding the integration of the agent and workflow concepts. By itself, the approach offers a simple, yet elegant way of supporting flexibility in workflow management.

References

1. Abu Zafar Abbasi and Zubair A. Shaikh. A conceptual framework for smart workflow management. In *Information Management and Engineering, ICIME '09*, 2009.
2. Kamel Barkaoui, Hanifa Boucheneb, and Awatef Hicheur. Modelling and analysis of time-constrained flexible workflows with time recursive ecatnets. In Roberto Bruni and Karsten Wolf, editors, *WS-FM*, volume 5387 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2008.
3. Kamel Barkaoui and Awatef Hicheur. Towards analysis of flexible and collaborative workflow using recursive ecatnets. In Arthur H. M. ter Hofstede, Boualem Benatallah, and Hye-Young Paik, editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 232–244. Springer, 2007.

4. Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, and Martin Jurisch. Von ADEPT zur AristaFlow BPM Suite - Eine Vision wird Realität: "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen. *EMISA Forum*, 29(1):9–28, 2009.
5. Michael Duvigneau. Bereitstellung einer Agentenplattform für petrinetzbasierte Agenten. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg, December 2002.
6. Lars Ehrler, Martin Fleurke, Maryam Purvis, and Bastin Tony Roy Savarimuthu. Agent-based workflow management systems (WfMSs) - JBees: a distributed and adaptive WfMS with monitoring and controlling capabilities. *Information Systems and E-Business Management*, 4, Number 1 / January, 2006:5–23, 2005.
7. Thomas Jacob. Implementierung einer sicheren und rollenbasierten Workflowmanagement-Komponente für ein Petrinetzwerkzeug. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg, 2002.
8. Olaf Kummer. *Referenznetze*. Logos Verlag, Berlin, 2002.
9. Olaf Kummer, Frank Wienberg, Michael Duvigneau, Michael Köhler, Daniel Moldt, and Heiko Rölke. Renew – the Reference Net Workshop. In Eric Veerbeek, editor, *Tool Demonstrations. 24th International Conference on Application and Theory of Petri Nets (ATPN 2003). International Conference on Business Process Management (BPM 2003).*, pages 99–102. Department of Technology Management, Technische Universiteit Eindhoven, Beta Research School for Operations Management and Logistics, June 2003.
10. Daniel Moldt, José Quenum, Christine Reese, and Thomas Wagner. Improving a workflow management system with an agent flavour. In Michael Duvigneau and Daniel Moldt, editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering, PNSE'10, Braga, Portugal*, number FBI-HH-B-294/10 in Bericht, pages 55–70, Vogt-Kölln Str. 30, D-22527 Hamburg, June 2010. University of Hamburg, Department of Informatics.
11. Christine Reese. *Prozess-Infrastruktur für Agentenanwendungen*. Dissertation, Vogt-Kölln Str. 30, D-22527 Hamburg, 2009.
12. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2004.
13. Rüdiger Valk. Concurrency in Communicating Object Petri Nets. In *Advances in Petri Nets: Concurrent Object-Oriented Programming and Petri Nets*, volume 2001 of *Lecture Notes in Computer Science*, pages 164–195. Springer-Verlag, Berlin Heidelberg New York, 2001.
14. Wil M.P. van der Aalst. Verification of Workflow Nets. In *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, volume 1248, pages 407–426, Berlin Heidelberg New York, 1997. Springer-Verlag.
15. Thomas Wagner. A Centralized Petri Net- and Agent-based Workflow Management System. Number FBI-HH-B-290/09 in Bericht, pages 29–44. University of Hamburg, September 2009.
16. Thomas Wagner. Prototypische Realisierung einer Integration von Agenten und Workflows. Diploma thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, 2009.
17. Thomas Wagner, Jose Quenum, Daniel Moldt, and Christine Reese. Providing an agent flavored integration for workflow management. *LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)*, LNCS 6900, 2012.