Model-Driven Middleware Support for Team-Oriented Process Management

Matthias Wester-Ebbinghaus and Michael Köhler-Bußmeier

University of Hamburg, Department of Informatics
Theoretical Foundations of Informatics
{wester,koehler}@informatik.uni-hamburg.de
http://www.informatik.uni-hamburg.de/TGI

Abstract. Management of collaborative processes involving multiple parties is one of the dominant topics in contemporary information system research. While the process perspective is quite well understood and supported by a wide range of modeling approaches, it is necessary to go beyond the process perspective alone. We specifically address the following question: If we consider the involved parties of a collaborative process as a *team*, then (1) which are the general *formation rules* for such a team together with the collaborative process it carries out and (2) to which concrete underlying *organizational structure* do these rules apply? To address this question, we present the organizational modeling approach SONAR. The accompanying models are rather high-level and illustrative but at the same time they are rich enough in order to generate executable models and other kinds of code that together form the core of a middleware implementation for team-oriented process management.

1 Introduction

Management of collaborative processes involving multiple parties is one of the most dominant topics in contemporary information system research, especially in the field of business process management (BPM) but also on a smaller scale in the field of computer-supported cooperative work (CSCW) or community support. The process perspective itself is quite well understood and there exists a wide range of more or less similar process modeling approaches (differing in specific aspects), including workflow nets and their descendants [1,2], the Business Process Modeling Notation (BPMN) [16], the Web Service Business Process Execution Language (BPEL) [4], Event-driven Process Chains (EPCs) [13] and the Yet Another Workflow Language (YAWL) [3]. However, there remains the question of organizational structures behind a given set of processes, which is not addressed in a thorough and systematic way by these approaches. We want to formulate this question a bit more vividly in the following way: If we consider the involved parties of a collaborative process as a *team*, then (1) which are the general *formation rules* for such a team together with the collaborative process it carries out and (2) to which concrete underlying *organizational structure* do these rules apply?

To answer this question, a more comprehensive modeling approach is necessary, encompassing both a system's processes and structure in an integrated manner. While this is to some extent addressed in the field of enterprise architecture management (EAM), EAM is a rather high-level discipline and is at least not necessarily concerned with models whose primary purpose is to be directly transferred into software artifacts (although this may be true for some parts, especially for process models). Contrary to that, the field of organization-oriented multi-agent systems is primarily concerned with rather comprehensive organizational models that exhibit a close gap to software-technical deployment [5,8]. Here we find models that encompass multiple integrated organizational perspectives (e.g. structure, function, interactions, norms). But despite this multi-perspective approach, we typically still find approaches where either a structural or a process perspective dominates. In approaches like $\mathcal{S}$-$\mathcal{M}$OISE$^+$ [12]) or TEAMCORE/KARMA [17], a structural perspective dominates and a process perspective has to be inferred from certain functional specifications or from normative requirements concerning action execution. In approaches like ISLANDER [9], a process perspective dominates and a structural perspective has to be inferred from decompositions of the process models.

In this context of process and overall organizational modeling, we present the Petri net-based organizational modeling approach Sonar (**S**elf-**O**rganizing **N**et **AR**chitecture).[1] It explicitly addresses the question from above concerning structure and process perspectives in teamwork modeling. We provide a way to capture the whole context of team-oriented process management: from the underlying organizational structure over team formation up to process execution by the team.

We have laid specific emphasis on achieving the following combination: (1) Sonar models are simple enough to be easily understood and analyzed (by means of standard Petri net tools). (2) Sonar models are rich enough to capture the interplay of various organizational concepts in such detail that we can automatically generate executable models and other kinds of code from them. In this context, Figure 1 gives an overview of the results we present in this paper. We concentrate on the modeling approach and in which ways model parts are

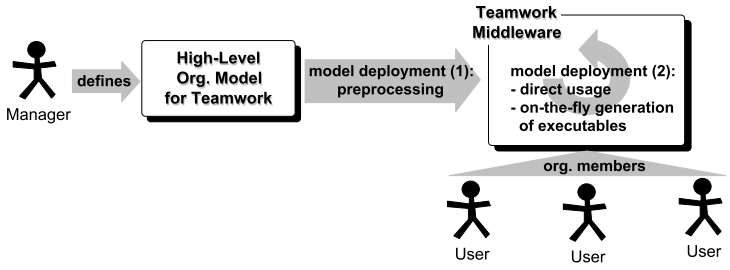

Fig. 1: Model-driven support for teamwork, general overview

[1] We are not dealing with the "self-organizing" part in this paper, but see the conclusion.

made executable or other code is generated from them. The figure can be regarded as capturing the most fundamental way of applying our SONAR approach. In the outlook of the paper we address several extensions that build upon it, but here we just regard the basic case. A manager creates a high-level organizational model that is void of execution details. The different model parts undergo a preliminary deployment phase where they are pre-processed and then passed on to a middleware layer for team-oriented processes management. Here, models are actually deployed and support the different phases/aspects of teamwork. Some of the pre-processed model parts are persistent and directly used while others serve an on-demand generation of temporary executables. Users access the middleware layer and act as organizational members that participate in teamwork (this users might be social but also artificial/software-technical actors).

In the remainder of the paper, we flesh out this rather abstract description. In Section 2, we introduce the SONAR modeling approach. In Section 3 we elaborate on transformations of original SONAR models in order to obtain code from them and in Section 4 we describe how this code is embedded in an agent-based middleware for teamwork. We conclude our work in Section 5 and give an outlook to advanced and future topics of our research.

Note that both the SONAR modeling approach and the middleware implementation rest on our previous work (cf. especially [14,15]). Several extensions, simplifications and improvements have been introduced over the years and in this paper, we present the consolidated current state with original contributions concerning both the modeling approach and the middleware support.

2 Organizational Models Based on SONAR

For organized activities two fundamental (and opposing) requirements have to be taken into account, the *division of labour* into various tasks and the *coordination* of carrying out these tasks. For SONAR, this can be rephrased more concretely and with reference to the terminology used in the introduction of the paper. *Coordinated carrying out of tasks* corresponds to a team executing a distributed (multi-party) workflow (DWF). *Division of labor* corresponds to the formation of such a team together with a DWF definition. Formation takes place according to general formation rules and a specific organizational structure to which these rules are applied. Consequently, SONAR models center around the duality of DWF (process) and organizational structure models. Both sides have to be coherently related with one another.

SONAR is based on Petri nets which offer both a graphical representation and formal semantics. In [14] we present SONAR in a formal way with theorems and proofs. However, in this paper we present a new version of SONAR, where the differences concern mainly a more readable and better structured organizational structure model. We will avoid formal specifications and instead give a rather illustrative introduction of the SONAR modeling approach. We just assume a general understanding of Petri nets (cf. [10]).

We will consider a running example throughout the paper. As SONAR models are based around the duality of distributed workflow (DWF) and organizational structure models, we could start with either of them. Here we begin with the workflow perspective. Figure 2 shows a DWF for collaboratively submitting a paper. The DWF is distributed in the sense that it encompasses multiple roles, here

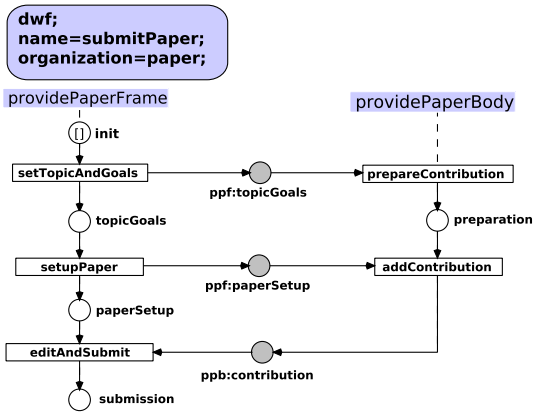

Fig. 2: Workflow with multiple roles for submitting paper

providePaperFrame and providePaperBody. Each action of the DWF is mapped onto exactly one role. Actions are modeled as transitions. They are connected by places. Places connecting actions belonging to the same role form the *DWF life line* of that role and we arrange such a life line vertically in our models.[2] Places connecting actions of different roles can best be considered as *message transfers between roles* and we draw them as horizontal connections. One can consider the places between the transitions of different roles as the *interface* between these roles. Places and transitions of a DWF model are named. Names of message places are prefixed with a key for the role sending that message (for example ppf: for the role providePaperFrame). Such message place names have to be unique across the whole set of DWF models of an overall SONAR model (see below for the reason). If a DWF model is decomposed into role parts and there exists an interface between two roles, each role part gets its own copy of the corresponding interface places.

Figure 3 shows another DWF. More exactly, it shows a DWF fragment. This fragment consists of two roles superviseSubmission and writeIntroAndConclusion. These two roles can be used to refine the role provideFrame from

[2] Of course, we do not rely on graphical arrangements in order to determine the different role parts of a DWF. We are currently working on an action inscription language for DWF transitions. So far, such an inscription does at least contain the name of the role that the transition belongs to. This is even more important when DWF life lines are not just sequences as in the rather simple examples in this paper. They may include forks, joins and concurrency. However, we have omitted the transition inscriptions in the DWF figures of this paper as the different role parts should be easily identified.

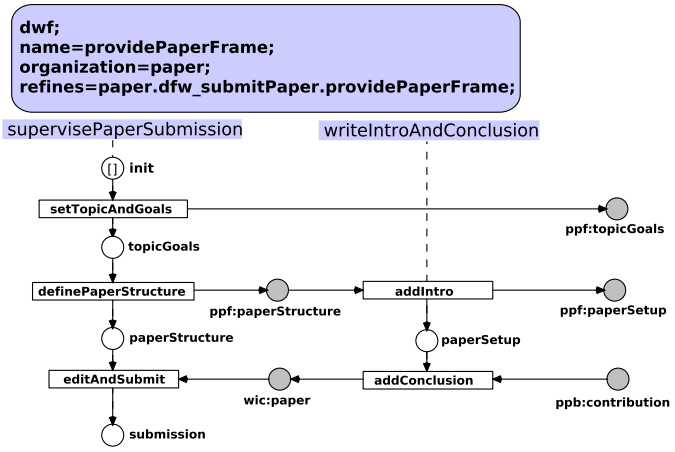

Fig. 3: Refined workflow part for the providePaperFrame role from Figure 2

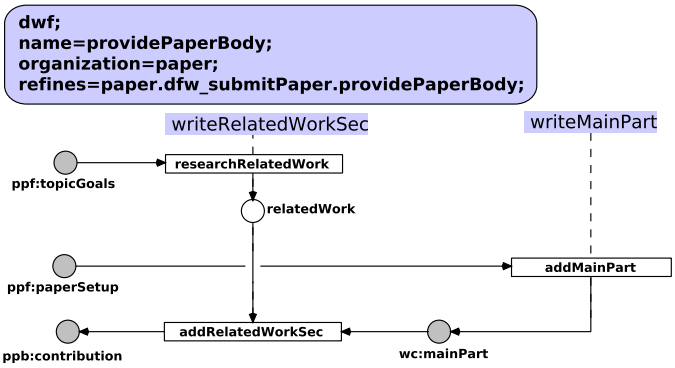

Fig. 4: Refined workflow part for the providePaperBody role from Figure 2

Figure 2. Note that on the right side of Figure 3, the two roles superviseSubmission and writeIntroAndConclusion *in combination* share the same interface as the role providePaperFrame in Figure 2 in terms of message places (whose names have been carried over and uniquely identify them). In fact, it is possible to substitute the two combined roles superviseSubmission and writeIntroAndConclusion for the role providePaperFrame and obtain the same input/output behavior to the outside, i.e. from the viewpoint of the partner role providePaperBody.

Likewise, Figure 4 shows a DWF fragment, where the two roles writeRelatedWorkSec and writeMainPart can be used to refine/substitute the role providePaperBody from Figure 2 while obtaining the same input/output behavior from the viewpoint of the partner role providePaperFrame.

Following this line of thought, it is of course also possible to substitute both roles providePaperFrame and providePaperBody with the combined roles from Figures 3 and 4 respectively as both refinements respect the original input/output

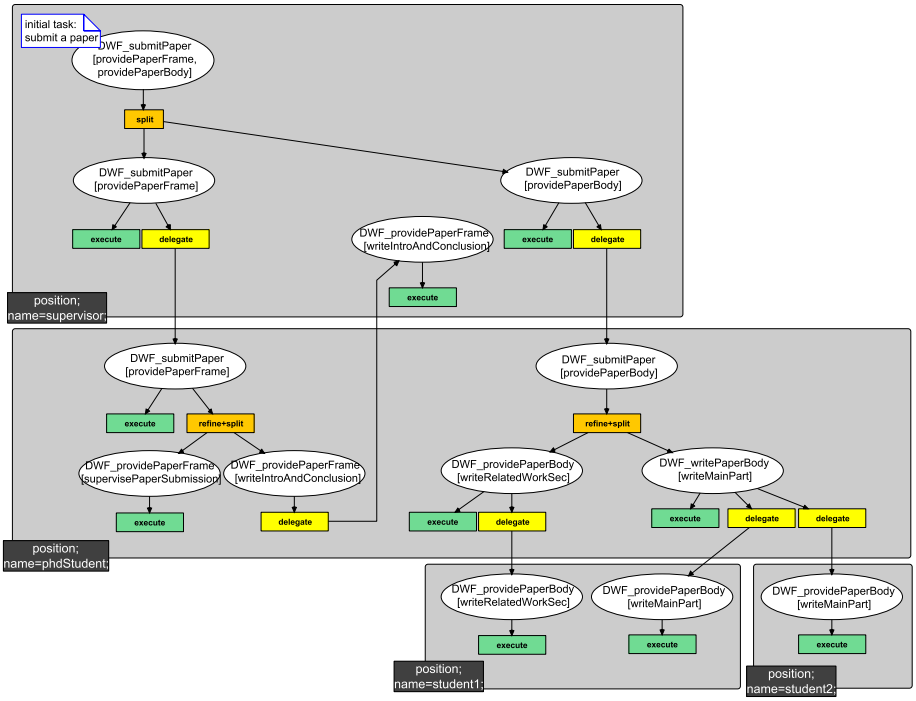

Fig. 5: Delegation model for jointly submitting a paper

behavior of the substituted roles and the overall composition thus fits together. To conclude, we arrive at basically four possible DWFs for jointly submitting a paper. Further models of role refinements would lead to more possibilities of DWF composition. It remains to supplement such a set of DWFs and DWF fragments with a model that determines not only when to compose which DWF parts but also who takes on which roles in a finally composed DWF. This is where SONAR organizational structure models come into play.

Organizational structures in SONAR are basically modeled as delegation structures. Figure 5 shows such a delegation net for the running example of joint paper submission.[3] A SONAR delegation net comprises multiple *positions* that are abstractions of actors (that occupy these positions when a SONAR organization is deployed). Positions are modeled as grey boxes that partition an underlying *task structure*. In Figure 5, we have as positions a *supervisor*, a *phd student* and two *students* that distribute tasks among themselves in order to jointly submit a paper. The underlying task structure is modeled as a Petri net. A place models a *task* and a transition models the *implementation* of a task. For this purpose, each transition has exactly one place in its preset. Task implementation can take on multiple forms and transitions are named accordingly:

[3] Delegation nets have been over-hauled compared to previous publications, cf. [14,15]. The explicit inscription of transitions with the implementation type that they represent leads to slightly larger but much more readable models.

1. *Execute:* The task is directly executed.
2. *Delegate:* The task is delegated.
3. *Refine:* Sub-tasks for a task are determined.
4. *Split:* A task is split into (already determined) sub-tasks.

The latter two cases are typically combined. All the implementation cases appear in Figure 5. Delegations are the kind of task implementation that relates two positions while refines, splits and executions are internal to positions.

The intertwining of a SONAR delegation net with DWF (fragment) models lays in the nature of the tasks. Each task in a delegation net corresponds to one or more roles in a DWF. Consequently, the places in Figure 5 are named according to the pattern $DWF_a[role_1, ..., role_n]$, meaning that the task corresponds to implementing the roles $role_1, ..., role_n$ from DWF DWF_a. Combining a delegation model with a set of DWF models leads to a straightforward notion of well-formedness of an overall SONAR model: (1) Delegation has to start with an initial task that corresponds to all roles of a complete DWF model (not a DWF fragment) and (2) task refinements must map onto associated role refinements in the set of DWF (fragment) models. Consequently, Figure 5 shows *one* possible delegation model for the DWF models from Figures 2 – 4 (likewise, other sets of DWF models may fit to the delegation model).

This way, the process perspective represented by DWF modeling is supplemented with an organizational structure perspective that guides both the formation of teams (where positions represent the team members) and the associated team DWFs. For example, the delegation model from Figure 5 allows multiple teams to be formed. An interesting fact is that team formation actually corresponds to the possible firings of the delegation net, its Petri net processes, cf. [11]. Each Petri net process of a delegation net model corresponds to a possible team.

Using Petri nets as the basis for SONAR modeling allows us to to take advantage of well-known analysis techniques. As we rely on simple place/transition (P/T) nets, there exist standard techniques and tools for checking the soundness of workflow net models or the free-choice nature of delegation models (cf. [1,7] and the ProM framework[4]). The interleaving of delegation and DWF models and especially the notion of role refinement of course goes beyond P/T net analysis. But especially the tool set from http://www.service-technology.org promotes a service-oriented perspective on Petri net models where Petri nets with interface places (open nets) are characterized in terms of their possible partners, cf. [20]. For our purpose, this allows to analyse whether a role and its refinement in terms of multiple roles really have the same input/output behavior and can be substituted with one another in DWF models.

3 Model Deployment

The models presented so far have been on a relatively high level. They are basically P/T nets, where some naming conventions have to be followed. There

[4] http://www.promtools.org/

are no execution details, except for the fact that Petri nets inherently have an operational semantics. It is not even necessary to model all possible DWFs that can occur during the execution of a SONAR organization. Instead, it is sufficient to model some initial DWF models and then just add models for selective role refinements.

In order to utilize the models in the context of a SONAR-based middleware layer for teamwork, some deployment steps are necessary. Based on the preceding section, it is now possible to be more specific on the model-driven support for teamwork illustrated in Figure 1 from the introduction. Figure 6 shows a SONAR-based re-interpretation of the figure.

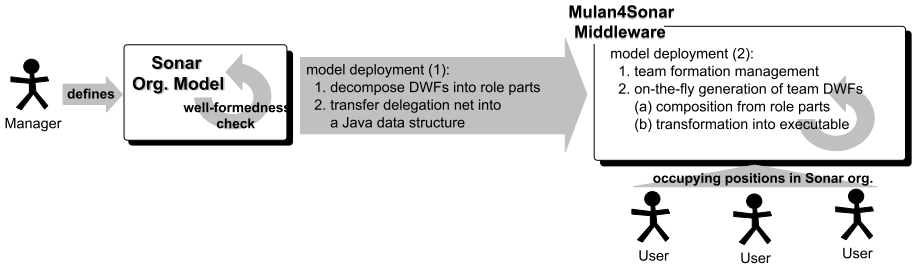

Fig. 6: Model-driven support for teamwork based on SONAR

After checking well-formedness of all aspects of an overall SONAR model (see the previous section), the first phase of model deployment is pre-processing. One immediate question is whether to use the Petri net models themselves (enriched/extended for deployment)[5] or whether to transfer them into other artifacts. Although we do not deal with run-time re-organization in this paper, this aspect has a strong impact on answering this question. Changing the Petri net models and re-deploying them at run-time can get quite cumbersome and costly. Currently, we have decided to use the DWF models directly in their Petri net form and to transfer the delegation model into a Java data structure. We treat DWF models and thus *how things are basically done* as rather fixed and the role parts as the basic behavioral building blocks. Fundamentally changing the DWF models is often better done by starting from scratch (however, instead of changing DWF models, they can be extended by further role refinements). The delegation model on the other hand and thus *the context leading to the actual behavior* is prone to quite frequent and light-weight re-organization efforts: adding/removing positions, adding/removing delegation relationships, adding/removing execution etc. Consequently, we prefer a data structure that handles changes easier. In addition, such a data structure is helpful to share (communicate about) and process knowledge in the context of team formation (determining eligibility of task implementations, possible delegation partners or whole sub-teams etc.). To conclude, the pre-processing phase of model deployment comprises two parts.

[5] This is what we did for our previous versions of a SONAR middleware layer, cf. [15].

1. All DWF models are decomposed into their singular role parts, which makes it easy to dynamically compose team DWFs later on.[6]
2. From the delegation model, a Java data structure is generated. Figure 7 shows the according class diagram in UML style. More specifically, it is a *concept diagram* [6] and is supported by the tool suite that we use in the context of our multi-agent framework MULAN that we briefly address in the following section.[7] The class hierarchy resulting from a concept diagram comes with the handy feature that all objects of these classes have FIPA[8]-conform String representations, which allows to directly include them as message contents in agent communication. According to the class hierarchy from Figure 7, each SONAR delegation model is transferred into an *organization object* that contains all other information.

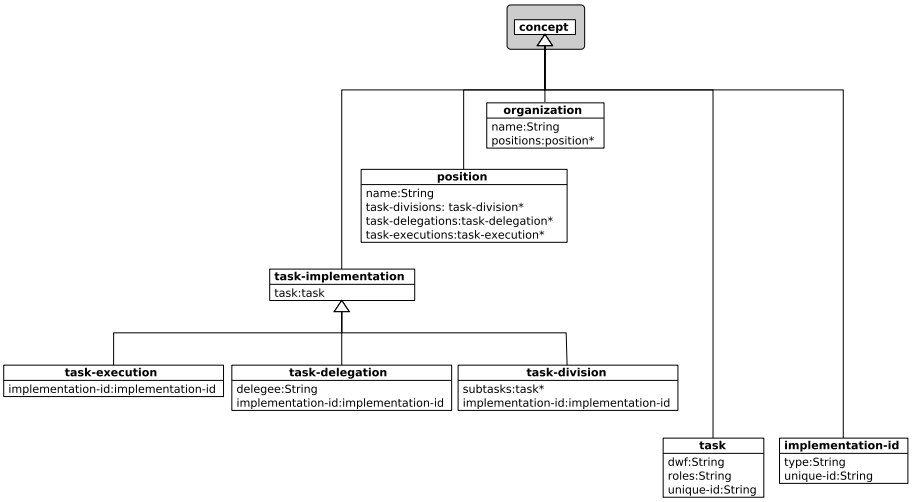

Fig. 7: Concept (or class) diagram for SONAR delegation models

In the second phase of model deployment, the pre-processed models are used by the MULAN4SONAR middleware layer to enable and frame teamwork among members of a SONAR organization. Members are (social or artificial) actors that access the middleware layer and occupy positions of a SONAR delegation model. We will elaborate on the MULAN4SONAR middleware layer and on how to access it in the next section.

[6] It might of course be possible to keep role compositions that always have to appear together in a team DWF, like superviseSubmission and writeIntroAndConclusion from the running example. But as we intend to have dynamic re-organizations of SONAR models at run-time (see the conclusion), further role refinements might be introduced. Thus it is simpler to keep track of each singular role part in the first place.

[7] see also http://www.paose.net.

[8] http://www.fipa.org

Basically, the Java data structure of the delegation model is used to manage task delegation and thus the team formation process. As soon as a team is formed, there is a unique team DWF associated with it: It is the composed DWF that consists of the role parts that are implemented by *execute* (instead of *refine, split* or *delegate*) transitions during the delegation process. The well-formedness of an overall SONAR model ensures that these role parts fit together. For example, the composed DWF from Figure 8 is the team DWF for a team where the delegation process has lead to the maximum level of task (and thus role) refinement for the running example of joint paper submission.

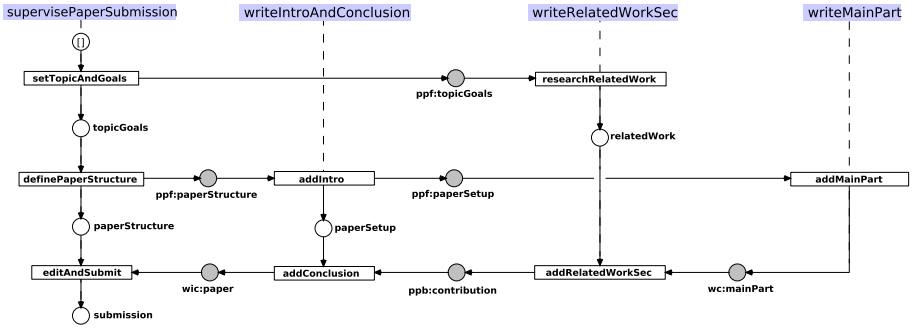

Fig. 8: A multi-role workflow for submitting a paper composed from role fragments

Consequently, team formation leads to an on-thy-fly generation of the corresponding team DWF. However, for such a team DWF to be executable in the context of the middleware layer, a further refinement and enrichment has to be carried out. This is also done by automatic generation. Basically, each action transition of a team DWF has to be enriched with execution inscriptions and has to be divided into a *call* and *return* part. Figure 9 exemplifies the substitution rule applied to a DWF transition for the addConclusion action of the team DWF from Figure 8. The transition is split into two transitions for call and return. The names of places lead to the generation of variable names that are bound to work-item and result objects of the action. The action call is parametrized with the role name, action name and a set of incoming work-items. The surrounding engine for the execution of team DWFs has to take care of forwarding the call to the position holder that implements this role for this team. In addition, the engine generates a unique action ID that can be used to associate action call and return. The action return is parametrized with a result object. We omit details on handling erroneous or aborted execution of DWF actions here.

Figure 10 shows the class diagram for content objects used in the context of executable team DWFs. More specifically, it shows the generic part. Concrete SONAR models are intended to extend the concept dwf-action-content with customized concepts. In fact, we are working on a high-level action inscription language for SONAR DWF models. Such a language can for example be used to attach pre-conditions, post-conditions and effects for/of DWF actions based on

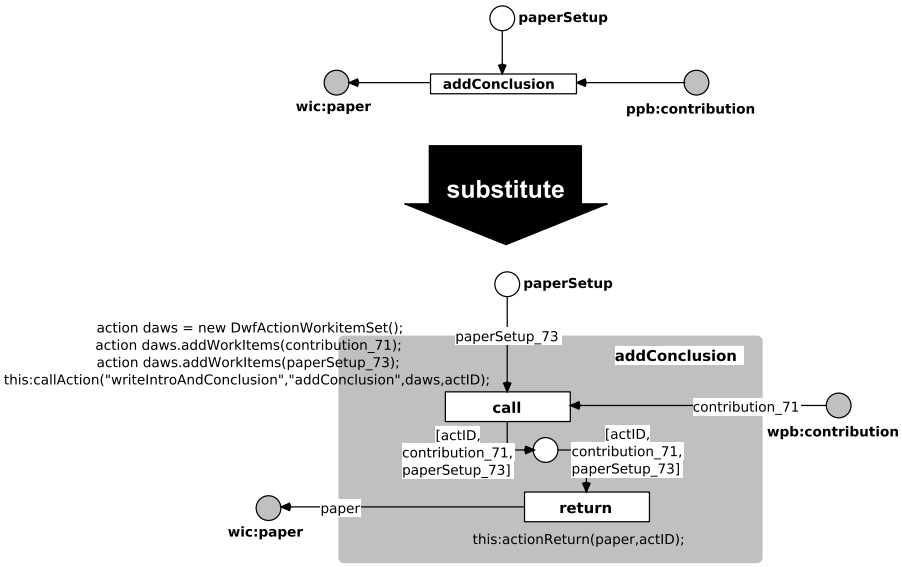

Fig. 9: Substitution rule (by example) for generating executable workflow models

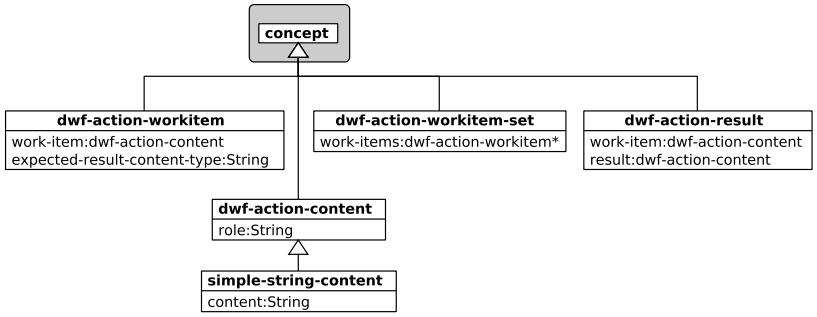

Fig. 10: Concept (or class) diagram for executable team DWF contents

the content objects and their attributes that are involved in the action. For this purpose it is necessary to explicitly define the according custom concepts.

To conclude this section, the illustrative and rather high-level Petri net models that a modeler has to create for a SONAR organization are rich enough to allow the generation of different kinds of executable artifacts for computer-supported teamwork. In the next section, we give an overview of a middleware implementation that utilizes these artifacts.

4 MULAN4SONAR: Agent-Based Teamwork Engine

We present a middleware implementation for teamwork support that is based on SONAR models and their deployment. There exist of course multiple possibilities and here we present our current approach, called MULAN4SONAR. This name

stems from the fact that it is based on the multi-agent system (MAS) framework MULAN (cf. [6] and www.paose.net). This framework provides the possibility to combine Java programming with Petri net modeling and simulation for realizing MAS and is consequently perfectly suited for our purpose. More concretely, MULAN relies on the high-level Petri net formalism of *Java reference nets* that is supported by the RENEW tool (www.renew.de).[9]

Figure 11 shows our general proposal for multi-agent system deployment of SONAR models. We have an *organization agent* that represents the SONAR or-

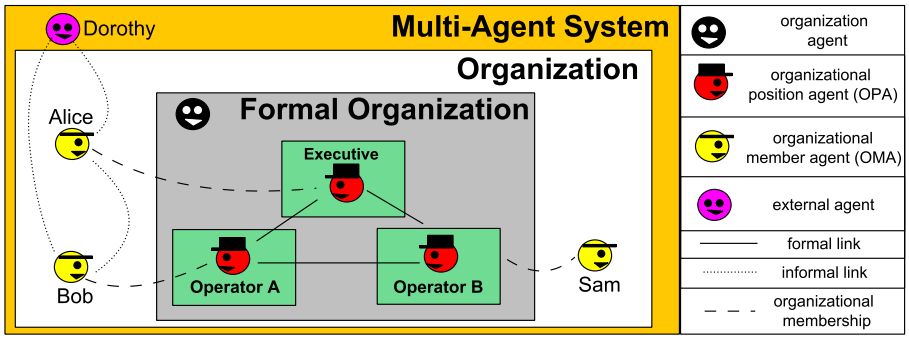

Fig. 11: Basic concept for SONAR-based multi-agent systems

ganization as a whole. It is responsible for initialization and for keeping a global perspective. With each position of a SONAR model we associate one dedicated agent, called an *organizational position agent* (OPA). From a conceptual point of view, the resulting OPA network (together with the organization agent) embodies a *formal organization* as each OPA represents an organizational *artifact* and not a *member/employee* of the organization. From a technical point of view, the OPA network is an agent-based middleware layer for supporting teamwork according to SONAR models. Consequently, each OPA represents a connection point for an *organizational member agent* (OMA). Each OMA interacts with its associated OPA to carry out organizational tasks and to make decisions where required. An OPA both enables and constrains organizational behaviour of its OMA. The OMA can effect the organization only in a way that is in conformance with the OPA's specification. In return, the OPA relieves its OMAs of a considerable amount of organizational overhead by automating coordination and administration. Conceptually speaking, OMAs implement/occupy the formal positions and thus represent the *informal part* of the organization. Note that an OMA can be an artificial as well as a human agent. OMAs might of course only be partially involved in an organization and have relationships to multiple other agents than their OPA or even to agents completely external to

[9] An example for using Java inscriptions in the context of a Petri net model was already shown in Figure 9. The other way round is equally possible, namely having Java objects monitoring, triggering or even controlling parts of the execution (simulation) of a Petri net model.

the organization. From the perspective of the organization, all other ties than the OPA-OPA and OPA-OMA links are considered as informal connections.

Our current implementation of MULAN4SONAR follows this general proposal. However, it does not (yet) feature OPAs as distinct agents. Instead, our current implementation features a central *organization agent* that manages the teamwork processes of a SONAR organization but also utilizes separate *OPA shells* for each position. Consequently, we have already prepared the implementation in way to be able to single out the shells as OPAs and thus to obtain a more distributed implementation. Figure 12 shows the three-level architecture of the organization agent in its current form. This architecture is actually realized based on the

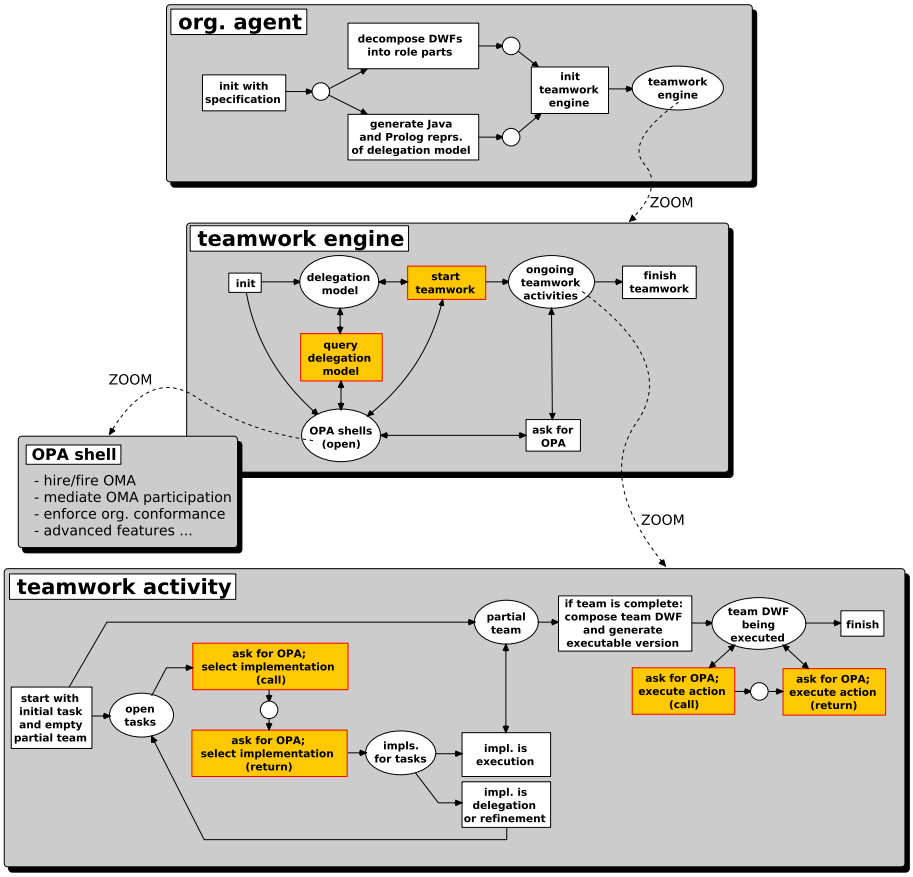

Fig. 12: Three-level architecture of the SONAR teamwork engine agent

high-level Petri net concept of *nets-within-nets* [18] where Petri nets can have other Petri nets as tokens and cross-net communication is possible. Nets-within-nets modeling and simulation is supported by the RENEW tool and is one of the fundamental features of MAS development with MULAN. The figure only shows

a high-level overview of the organization agent's architecture but it represents the actual implementation quite precisely.

On the top level of the *organization agent*, the organization is initialized and the pre-processing of the original SONAR models is carried out, just as described in the previous section. Afterwards, the *teamwork engine* is initialized with the pre-processed models as input and represents the second level.

The teamwork engine sets up *OPA shells* for all positions and makes the delegation model available to them. We will not go into detail concerning the manifold responsibilities of the OPA shells. We just assume that basically, they take care of binding users as members (OMAs) into the organization and manage the inclusion of the OMAs' actions and decisions in conformance to the organizational specifications. New *teamwork activities* can be started and represent the third level. Teamwork activities and OPA shells stay connected via the teamwork engine level.

A teamwork activity basically comprises the delegation process for team (DWF) formation and afterwards the execution of the composed team DWF by the team members that take on roles in the DWF. All of this is managed by the teamwork activity level, together with the involved OPA shells that can be consulted via the joint teamwork engine super-level.

5 Conclusion and Outlook

Starting from the question for an integrated treatment of structure and process perspectives in modeling collaborative systems, we have presented our SONAR approach. It provides a way to capture the whole context of team-oriented process management: from the underlying organizational structure over team formation up to process execution by the team. The accompanying models are rather high-level and illustrative but at the same time they are rich enough in order to generate executable models and other kinds of code that together form the core of the MULAN4SONAR middleware implementation for team-oriented process management.

Regarding our future research efforts, there is a range of topics that we and other people from our research group are working on.

- *Collaborative Agent Platform (deployment):* Our group has the long-term goal of developing an agent-based platform for computer-supported collaboration. We envision SONAR-based organizational models to be used for specific teamwork applications *on top of the platform* as well as for supporting the infrastructure *of the platform itself*, managing the various platform tasks and processes.
- *Self-organization:* Instead of the manager from Figure 1 we promote an alternative approach where a SONAR model has multiple management levels and the team processes on one level lead to the transformation of the specifications of the lower levels, cf. [15].
- *Hierarchy/holism:* While the idea of self-organization introduces multiple management levels in the context of *one* SONAR organization, we also address

the concept of having multiple levels of nested SONAR organizations. The basic idea is to have positions being occupied by *organizational units* that are SONAR organizations themselves. This allows to model inter-organizational scenarios and so-called *multi-organization systems*. The refinement concept for roles and tasks inherent to SONAR directly supports such an extension. For a thorough report of our research on modeling organizational units and multi-organization systems (not limited to SONAR), we refer to [19] (in German).

– *Simulation:* We are also interested in organizational simulation. We intend to enrich the models with quantitative information and apply routines to evaluate simulation runs with respect to certain criteria. Especially in combination with hierarchic models we are interested in studying the fit of different (types of) organizational units to one another (in terms of nesting relationships as well as in terms of cooperation effectiveness on the same level).

References

1. van der Aalst, W.: Verification of workflow nets. In: Application and Theory of Petri Nets 1997. Lecture Notes in Computer Science, vol. 1248, pp. 407–426. Springer (1997)
2. van der Aalst, W.: Interorganizational workflows. Systems Analysis - Modelling - Simulation 34(3), 335–367 (1999)
3. van der Aalst, W., ter Hofstede, A.: YAWL: Yet another workflow language. Information Systems 30(4), 245–275 (2005)
4. Alves et al., A.: OASIS web services business process execution language (WSBPEL) v2.0. OASIS Standard, 11. April 2007 (2007)
5. Boissier, O., Hübner, J., Sichman, J.S.a.: Organization oriented programming: From closed to open organizations. In: O'Hare, G., Ricci, A., O'Grady, M., Dikenelli, O. (eds.) Engineering Societies in the Agents World VII. Lecture Notes in Computer Science, vol. 4457, pp. 86–105. Springer (2007)
6. Cabac, L.: Modeling Petri Net-Based Multi-Agent Applications, Agent Technology: Theory and Application, vol. 5. Logos (2010)
7. Desel, J., Esparza, J.: Free Choice Petri Nets, Cambridge Tracks in Theoretical Computer Science, vol. 40. Cambridge University Press (1995)
8. Dignum, V.: The role of organization in agent systems. In: Dignum, V. (ed.) Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models, pp. 1–16. Information Science Reference (2009)
9. Esteva, M., de la Cruz, D., Sierra, C.: ISLANDER: An electronic institutions editor. In: The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, Proceedings. pp. 1045–1052. ACM (2002)
10. Girault, C., Valk, R. (eds.): Petri Nets for Systems Engineering: A Guide to Modelling, Verification and Applications. Springer (2003)
11. Goltz, U., Reisig, W.: The non-sequential behaviour of Petri nets. Information and Control 57(2–3), 125–147 (1983)
12. Hübner, J.F., Sichman, J.S.a., Boissier, O.: Using the $\mathcal{M}$oise$^+$ for a cooperative framework of MAS reorganisation. In: Bazzan, A., Labidi, S. (eds.) Advances in Artificial Intelligence – SBIA 2004. Lecture Notes in Computer Science, vol. 3171, pp. 481–517. Springer (2004)

13. Keller, G., Nüttgens, M., Scheer, A.W.: Semantische prozessmodellierung auf der grundlage „ereignisgesteuerter prozessketten (epk)". In: Scheer, A.W. (ed.) Veröffentlichungen des Instituts fü Wirtschaftsinformatik (IWi), Universitä des Saarlandes. Heft 89 (1992)
14. Köhler-Bußmeier, M., Moldt, D., Wester-Ebbinghaus, M.: A formal model for organisational structures behind process-aware information systems. In: van der Aalst, W., Jensen, K. (eds.) Transactions on Petri Nets and Other Models of Concurrency II: Special Issue on Concurrency in Process-Aware Information Systems, Lecture Notes in Computer Science, vol. 5460, pp. 98–115. Springer (2009)
15. Köhler-Bußmeier, M., Wester-Ebbinghaus, M., Moldt, D.: Generating executable multi-agent system prototypes from SONAR specifications. In: de Vos, M., Fornara, N., Pitt, J., Vouros, G. (eds.) Coordination, Organizations, Institutions and Norms in Agent Systems VI. Lecture Notes in Artificial Intelligence, vol. 6541, pp. 21–38. Springer (2010)
16. OMG: Business process modeling notation (BPMN) version 1.0. OMG Final Adopted Specification, Object Management Group (2006)
17. Pynadath, D., Tambe, M.: An automated teamwork infrastructure for heterogeneous software agents and humans. Autonomous Agents and Multi-Agent Systems 7(1–2), 71–100 (2003)
18. Valk, R.: Object petri nets: Using the nets-within-nets paradigm. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets: Advances in Petri Nets, Lecture Notes in Computer Science, vol. 3098, pp. 819–848. Springer (2004)
19. Wester-Ebbinghaus, M.: Von Multiagentensystemen zu Multiorganisationssystemen – Modellierung auf Basis von Petrinetzen. Dissertation, Universität Hamburg, Fachbereich Informatik. Elektronische Veröffentlichung im Bibliothekssystem der Universität Hamburg: `http://www.sub.uni-hamburg.de/opus/volltexte/2011/4974/` (2010)
20. Wolf, K.: Does my service have partners? Transactions on Petri Nets and Other Models of Concurrency II, Special Issue on Concurrency in Process-Aware Information Systems 5460, 152–171 (2009)