Modeling Systems of Systems as Nested Actor Systems Based on Petri Nets

Matthias Wester-Ebbinghaus[1], Daniel Moldt[2] and Simon Adameit[3]

Abstract: Modern software systems are frequently characterized as systems of systems. Agent-orientation as a software engineering paradigm exhibits a high degree of qualification for addressing many of the accompanying challenges. However, systems of systems demand for means of hierarchical/recursive decomposition that are not inherently rooted in the agent-oriented paradigm. We present a model that still relies on the actor metaphor, but shifts the focus to collective agency. We propose a universal model of a system unit that both embeds system actors and is itself embedded as a collective system actor in surrounding system units. Consequently, we can apply our model of a system unit at arbitrary levels of a system of systems and compose the overall system by means of nested actor hierarchies. (High Level) Petri nets as our modeling technique supply precise operational semantics for the functioning of these kind of systems. In addition, we offer abstraction mechanisms that allow for rather high-level or low-level views and smooth transitions between them.

1 Introduction

Modern software systems are frequently characterized as inherently distributed and heterogeneous *systems of systems* [Mai99] whose parts potentially exhibit a great deal of operational and managerial independence [LMW05, Nor06, HHVE07]. An increasing emphasis is laid on the co-evolution of software and social systems that together form socio-technically integrated information systems. Among the large spectrum of work concerned with these kinds of systems we take agent-orientation as a vantage point for our work presented here. Agent-orientation as a software engineering paradigm is in many respects very well suited to deal with the above mentioned kind of systems. It advocates flexible, high-level interactions between system parts that exhibit a great deal of local freedom and initiative (agents). Furthermore, it fosters socio-technical integration by applying concepts to the software-technical side that are congruent to the real-world inspirations: positions, roles, services, delegation, speech-act based interactions, norms, etc.

However, it is the system of systems aspect that poses problems for agent-orientation. A system of systems perspective inherently demands for different levels of abstractions with

[1] University of Hamburg, Department of Informatics, Vogt-Kölln-Straße 30, D-22527 Hamburg, wester@informatik.uni-hamburg.de
[2] University of Hamburg, Department of Informatics, Vogt-Kölln-Straße 30, D-22527 Hamburg, moldt@informatik.uni-hamburg.de
[3] University of Hamburg, Department of Informatics, Vogt-Kölln-Straße 30, D-22527 Hamburg, 6adameit@informatik.uni-hamburg.de

regards to the granularity of system parts studied at each level. This is especially true under a socio-technical perspective as social systems inherently exhibit multiple levels of (actor) abstraction [Sco03]. Agent-orientation on the other hand features a rather flat decomposition, namely “decomposing problems in terms of autonomous agents” [Jen00]. As analyzed in [BvdT05] the *component view* of objects where objects may be composed of other objects in a recursive fashion was lost with the transition from object- to agent-orientation.[4]

Consequently, we have derived an approach that still relies on the actor metaphor but shifts the focus from the classical individual agent concept to the concept of a collective actor. The basic idea is depicted in Figure 1 in UML style. We use the general term *system unit* for

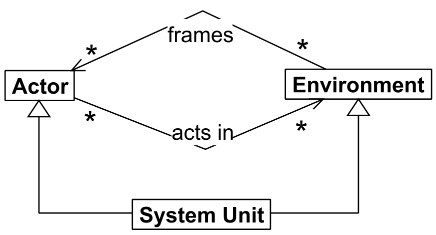

Fig. 1: Basic system unit concept

collective actors at the software level in order to express its capacity to provide a building block for wider systems while at the same time being a differentiated and useful entity of its own. A system unit consists of actors that embody it and and to whom the system unit provides an environment. While the embedded actors are for one part engaged in activities that are completely internal to the system unit, they are also engaged in activities where they act *on behalf of* the system unit in wider surroundings that are again system units. Thus, they make the system unit a collective actor that acts „as a whole“ in surrounding system units. We arrive at a recursive understanding of systems of systems composed of collective actors. Of course such a system also includes individual actors at some point. These can be understood as classical agents.

The paper at hand deals with one specific aspect of our overall approach and closes the gap between different themes of earlier work. Figure 2 helps us to position our work both in the wider context of our research efforts and in relation to other approaches.

Our wider approach ORGAN (**Org**anizational **A**rchitecture with **N**ets) evolves around an architectural proposal for software systems of systems that we term *multi-organization systems (MOS)*. The concept of a MOS is inspired by real-world organizational settings. It features software units derived from different kinds of collective social actors, e.g. *teams, departments, enterprises* and *organizational fields*. Each of these units is characterized by a distinct configuration of certain actors and associated activities. We have first introduced

[4] It should be noted that agent-orientation has begun to make efforts to overcome these drawbacks. Nowadays, there exist approaches that view collectivities of agents as first class abstractions that can collectively act on their own (e.g. holons [FSS03], JACKTeams [AG09], socially-constructed agents [BvdT05], platform agents [Röl04], organizations as specialized agents [HMMF08]).

a reference architecture for MOS in [WEMRM07] and in [WEM08] (top left of Figure 2). The proposal is quite abstract, being more of a coarse characterization. In [WEM09], we have supplied a precise technical model of how software systems based on the collective actor idea can be operationalized (bottom left of Figure 2). It is based on the high-level Petri net formalism of reference nets [Kum02] that fuses classical Petri nets semantics with the possibility of nesting Petri nets inside each other (following the initial nets-in-nets approach in [Val03]) and have them communicate via synchronous channels. In addition, reference nets have excellent tool support by means of the RENEW [KWD09] tool which made our operational models directly executable. However, between the abstract concept

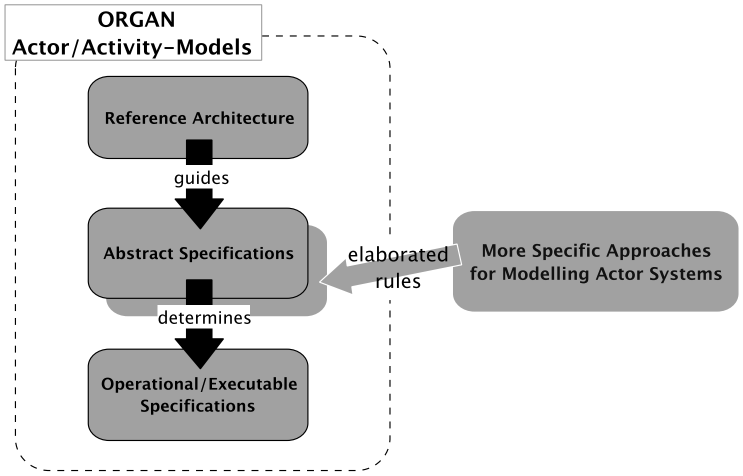

Fig. 2: ORGAN actor/activity-models

of a MOS and the fine-grained technical mechanisms to operationalize collective actor systems, we have to date not really elaborated on how to actually specify/model specific instances of these kinds of systems. In this paper we close this gap. We introduce a modeling approach that is based on the precise operational semantics of Petri net but that also offers various abstraction mechanisms to hide arbitrarily many details (middle left of Figure 2). Consequently, this approach can be used to smoothly transcend from very detailed to very abstract models of a system of systems and vice versa. Note that this approach is not *necessarily* tied to our MOS reference architecture (this accounts for the operational/executable models as well). It features generic principles to model systems of systems based on the universal concepts of actors and activities. The modeled collectivities need not fit into the categories of the MOS reference architecture. Consequently, the approach presented here can also be regarded independently from our wider research context.

Finally, the figure shows a relation to approaches external to ORGAN. The generic and universal character of our approach allows (and in many cases demands) to be supplemented by more specific approaches. Our approach alone basically allows to model which kinds of actors take part in which kinds of activities. But existing approaches to model collectivities of agents (e.g. teams, groups, organizations, institutions, c.f. [BHS07]) feature more elaborated means for describing structural and behavioral configurations. These are of course specific to their respective approach. Consequently, such approaches can be consulted selectively to enrich (parts of) our models with additional rules. We have shown in [WEKBM08] the value of considering different existing modeling approaches from the field of multi-agent systems with respect to addressing different system levels in a MOS.

2 Actors and Activities

The idea is to develop a universal model of a system unit. We follow Bertalanffy's [Ber56] classical definition of a system *"A system is a set of interacting units with relationships among them"*. We can identify the general notions of structure and behavior of a system. In our case we choose to conceptualize internal system parts and their interactions as actors and activities in which the actors participate. They will be considered in this section. The topic of additional relationships between actors will be covered in the subsequent sections.

We use the term activity in accordance with the UML [Boc03]. An activity models some part of system's behavior. It does so by combining elementary behavioral elements (i.e. actions) into more complex specifications of behavior by means of control and data flows. This of course leaves much room for specialization. In the following we will look at the specific understanding of activities for our system unit model.

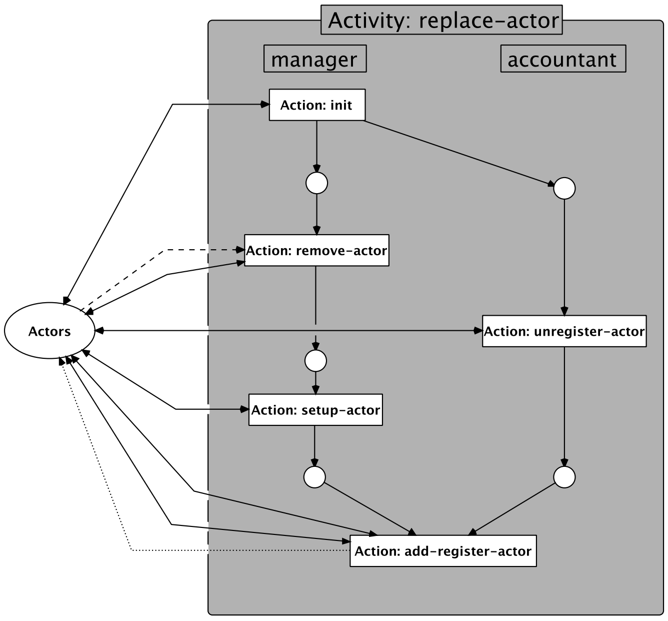

Fig. 3: A system unit net with one activity scheme

Figure 3 shows a fragment of a specific system unit as a Petri net. There is a place to the left of the figure where all the actors of the system unit reside. In addition, it contains a subnet to the right that embodies an activity specification that the actors can utilize. The actions that the activity is composed of are modeled by transitions. An action is always performed by an actor or synchronously by multiple actors. The participation of an actor in an action execution can have three possible effects on the actor:

1. **use/modify** *(modeled by an arc with double arrow tips):* The actor contributes to the execution of an action. So the actor is just „used" to accomplish the action. In addition, the actor might also undergo changes because of its participation.
2. **add** *(modeled by a dotted arc with the arrow tip at the actor place):* In the course of the action, the actor is added to the system unit. In a closed system unit, this concerns only the creation and addition of an actor by another actor who is already part of the system. In the case of an open system unit (c.f. Section 4), it may also be the case that new actors enter the system from the outside.
3. **remove** *(modeled by a dashed edge with the arrow tip at the action transition):* The actor is removed from the system unit. This could just be the actor itself leaving. Or it could be the forced removal of the actor by another one who has the required authority. In the case of an open system unit, removal might also mean migration to other system units outside of the current one.

The activity subnet showed here follows similar conventions like AUML interaction diagrams (c.f. [CMR03]). This means that vertically aligned transitions belong to an *activity role* and are to be executed by the same actor for one activity instance. The name of the role is displayed at the top of this so called *life line* of the role. In addition to life lines, places that connect transitions horizontally model a message exchange between the actors that occupy the respective roles. In particular, the activity shown here models the replacement of an actor of the system unit by another actor. the activity includes roles for a manager and an accountant. The actor playing the manager role tells the actor playing the accountant tole to un-register an existing actor. Afterwards, the manager removes this actor from the system and sets up a new one. Finally, manager and accountant synchronize for a joint action to add and register the new actor simultaneously.

It should be emphasized that the activity subnet shown here does not model *one* specific occurrence of an activity. It rather models a scheme for or a pattern of an activity. It can be used in multiple instances. However, if the model should not just be used for illustrative purposes but also to be directly executed (e.g. in the context of a Petri net tool like RENEW), of course additional technical details are necessary, especially if the activity subnet should be usable in multiple instances at the same time. We will not deepen on this topic in this paper (but see the conclusion).

As one can imagine, if one wants to model a complete system unit with a possibly high number of activities, the detailed approach from the last figure becomes unfeasible. Thus we present our first mechanism of abstraction in Figure 4. The resulting model shows the activities that can take place and the corresponding roles that are part of the activities and need to be played by actors. The edges give information about whether the actor that plays the role will be used, added or removed during the activity. The model still has a Petri net semantics , it is just that all activities are folded into one transition and appear as atomic. Behavior specifications in terms of control and data flow are now hidden.

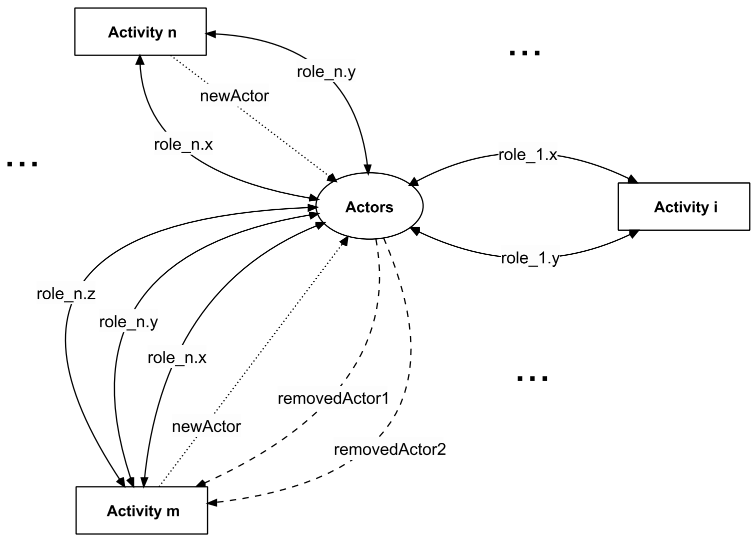

Fig. 4: System unit net with abstract activity specifications

3 Actor and Activity Sets

The abstraction level from model from Figure 4 still requires to consider each individual activity specification. It may be desirable to model actor-activity relationships on a still higher level of abstraction. For this purpose, this section introduces to model *sets* of actors and activities. This also allows to regard subsets and super-sets (set unions) depending on the desired level of abstraction. Just as different hierarchy levels of actor and activity set inclusion are regarded, we obtain hierarchical levels of system abstraction.

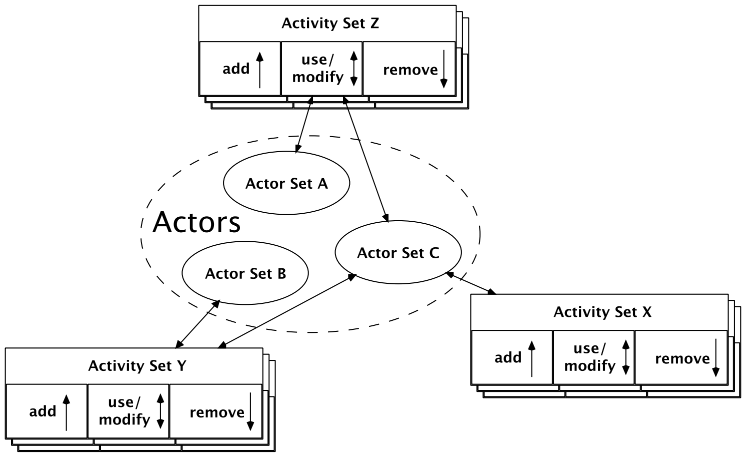

Fig. 5: Actor and activity sets

Figure 5 gives a first illustration. Now, actors are no longer associated with activities via specific effects (usage, addition, removal). Instead, it is just stated, which actor sets are associated with which activity sets. Consequently, the model does no longer exhibit Petri net semantics. However, Petri net semantics may be re-obtained by refining the model in terms of the two abstraction levels presented in the former section.

Despite losing specific information concerning activity execution, the model from Figure 5 does also introduce information that was not present beforehand. By „coloring“ the former uniform set of actors and relating the resulting distinguishable actor subsets with different activity sets, we can begin to make stronger statements concerning actor relationships. For example, we see that the actor sets A and B both are directly related to actor set C as they are engaged in the same activity sets respectively while A and B are not directly related to each other.

However, in order so transcend between abstraction levels of a system, it is not feasible to always be obliged to model *all* identifiable actor and activity sets exhaustively at a time. Consequently, we introduce shorthand notations for modeling relationships between actor and activity sets. These shorthands are based on the introduction of set unions on the other hand and the identification of subsets as the counterpart.

First of all, Figure 6 introduces the modeling of *contingent* activity participation of actors in activities. This shorthand notation models the circumstance that an actor set only partic-

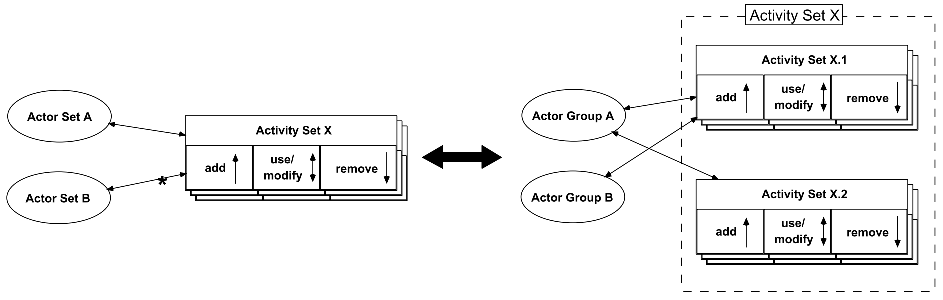

Fig. 6: Shorthand notation (on the left side): Contingent activity participation

ipates in *some* activities of an activity set. Consequently, resolving the shorthand notation, we obtain two subsets of the earlier activity set, one subset where the actor set is required and one subset where the actor set is not associated. The actor set A is only included in Figure 6 for illustrative purposes. As it is associated to the original activity set via an ordinary participation arc, it consequently is required for both identified activity subsets.

The next step is to directly abstract away from some particular actor sets. For this purpose, our model allows to build the union of these actor sets and just consider the newly obtained super-set as a holistic actor set. In the opposite direction, the modeler might see the necessity to identify subsets for an actor set that was considered as one single actor set before. Building actor set unions or identifying actor subsets always goes hand in hand with considerations concerning associated operations on the activity set side and considerations concerning activity participation on the different levels of abstraction.

Figure 7 exemplifies the shorthand notation of building actor set unions and how to resolve these shorthands when subsets are (re-)identified. In Figure 7 (a), resolving the actor set

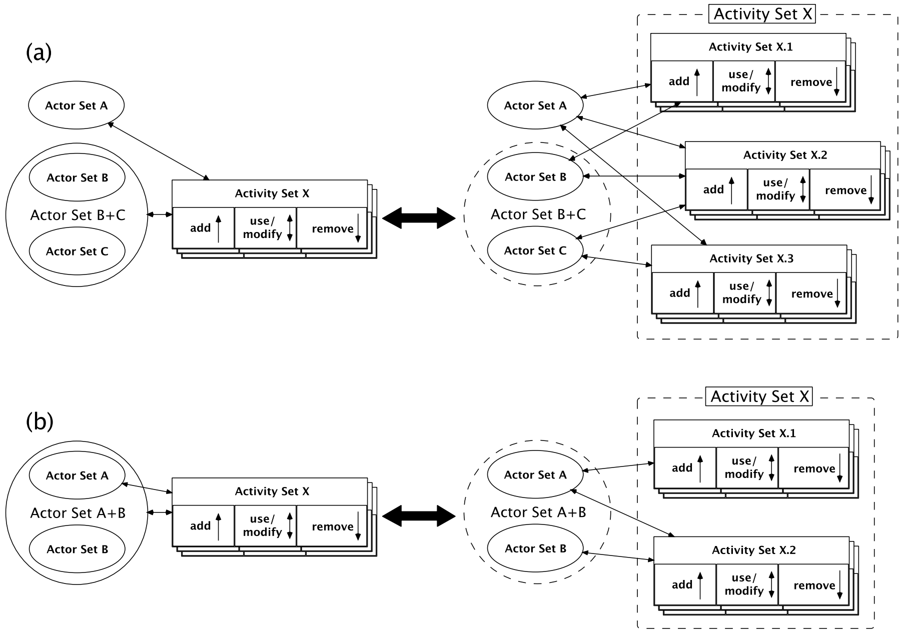

Fig. 7: Shorthand notation (on the left side): Hierarchical actor sets

B+C to the subsets B and C introduces three *potential* activity subsets for the former activity set X. However, not all of the subsets actually have to exist. Some might just be empty. Figure 7 (b) illustrates the case where activity participation of an actor set and one of its subset are mixed.

As actors and activities are dual concepts, the source of changing the level of abstraction might also be activity set union or the identification of activity subsets. Figure 8 illustrates this as a counterpart to the former figures. All in all, no matter whether set unions or

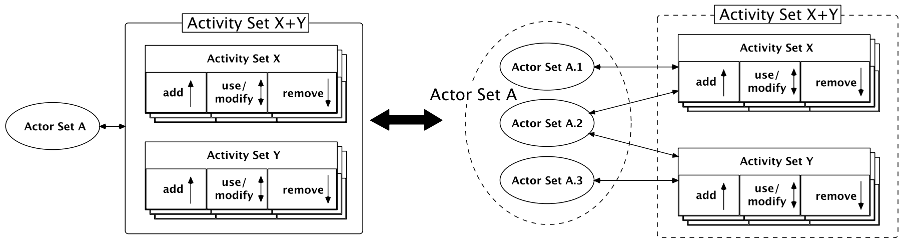

Fig. 8: Shorthand notation (on the left side): Hierarchical activity sets

identification of subsets are initiated from the actor or the activity side, they always lead to according considerations on the opposite side.

In the notations we have introduced so far the duality between actors and activities has been preserved. Both can be refined and coarsened in similar ways by introducing subsets and super-sets. These abstraction mechanisms enable us for example to design a system unit in a top-down manner by starting with the most abstract view of just differentiating between very coarse actors and activities and refining them until we arrive at Petri net models. As a counterpart, it is also possible to design a system in a bottom-up manner by identifying specific activities and actors in the beginning, modeling these and abstracting away from them to appropriate sets and super-sets as one discovers overall relations.

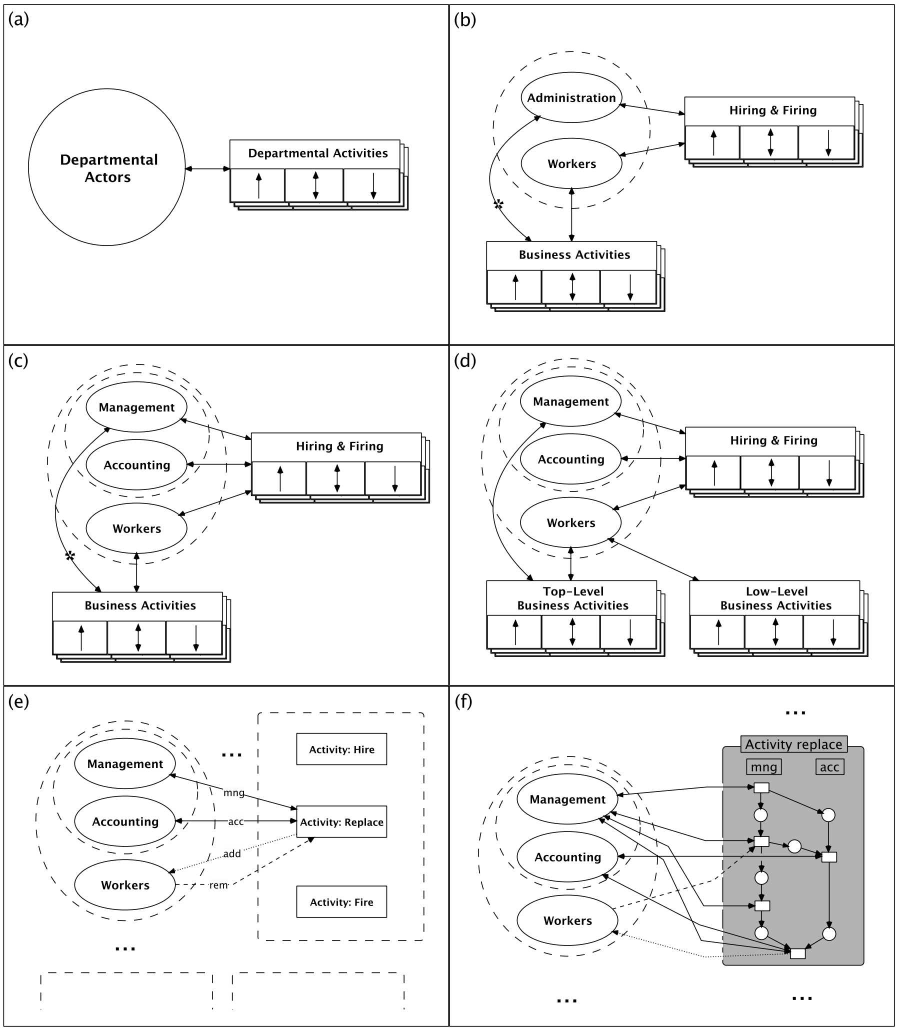

Fig. 9: Exemplary top-down design of a particular system unit.

Even more important, our model allows to transcend back and forth between levels of abstraction and especially to focus on a particular part of a system, elaborate on it and leave its context on an abstract level. As an illustration of combining all the abstraction mechanisms introduced thus far, Figure 9 shows an exemplary top-down design of a system unit.

The model we have considered so far fulfills the requirements set by Bertalanffy's definition of a system. It allows us to model structure as well as behavior of a system. We allow for a smooth modulation of abstraction for both of these. But so far, we have only regarded *one* system unit. In the next section, we advance to regard systems of systems.

4 Collective Actors

Our claim is that the ideal building block for systems of systems is based on the philosophy of having system units that act both as environments for actors and as actors themselves. Until now, we have talked about actors mainly in terms of actor sets. Even on the more

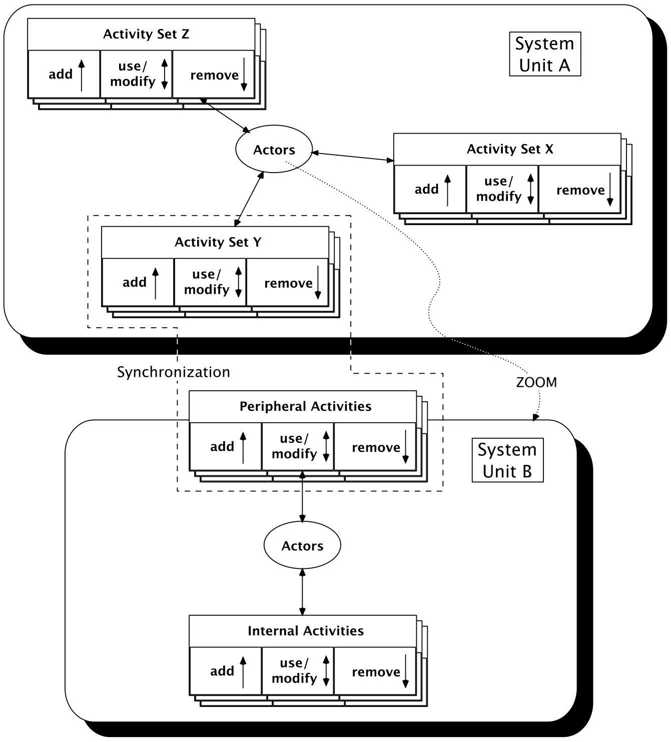

Fig. 10: Collective agency via peripheral activities

detailed level of Section 2 we have said nothing more than that actors have to be able to occupy activity roles in order to interact with other actors. Now, following our initial proposal, the actors of a system unit can again be system units as collective actors. A system unit's collective actions are actually carried out by its internal actors who act *on behalf of* or *in the name of* the system unit as a whole. Following the approach taken so far, actions are only carried out in the course of activities. Consequently, a system unit that acts as a whole has to offer activities that include actions that take effect not only in the context of the system unit itself but also in surrounding system units and thus at higher system levels. This idea is modeled in Figure 10. System unit B acts in the context of system unit A because it utilizes peripheral activities to allow its internal actors to carry out actions that appear as actions of B itself at the level of A.

We want to keep up with our emphasis on Petri net semantics underlying all of our models and elaborate on the Petri net semantics of peripheral activities in Figure 11. We see that

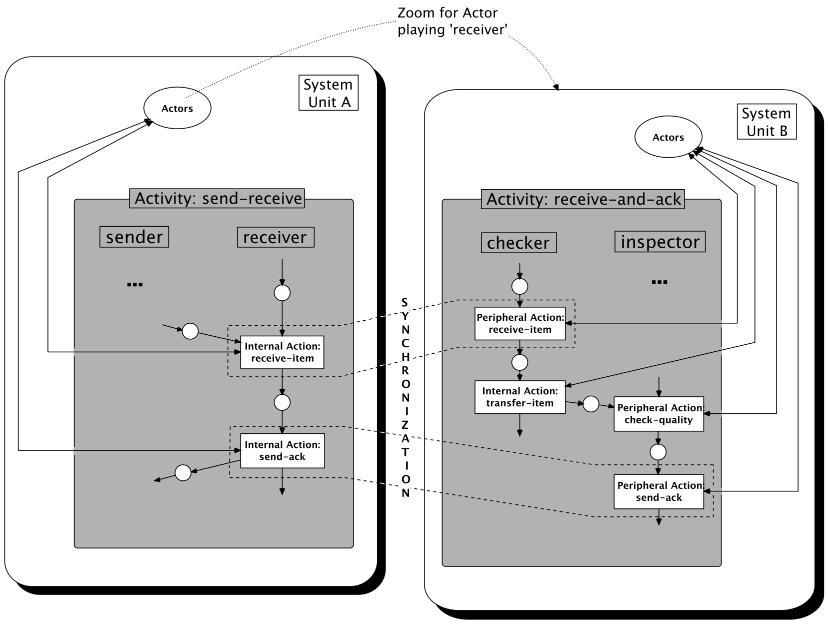
Zoom for Actor playing 'receiver'
System Unit A
Actors
Activity: send-receive
sender
receiver
...
Internal Action: receive-item
Internal Action: send-ack
SYNCHRONIZATION
System Unit B
Actors
Activity: receive-and-ack
checker
inspector
...
Peripheral Action: receive-item
Internal Action: transfer-item
Peripheral Action: check-quality
Peripheral Action: send-ack

Fig. 11: Petri net semantics of peripheral activities

a peripheral activity includes peripheral actions. These are exactly the actions that have effects on multiple system levels. It is important to note that for a peripheral activity not all actions have to be peripheral. In Figure 11 there are system unit A and system unit B. System unit B is an actor embedded by system unit A. System unit A offers the activity send-receive. In order to participate in this activity, B has to implement a role of the activity. In this case, it is the receiver role. At the level of A, it simply appears that B carries out the actions required by the role. But from the perspective of B itself, this is accomplished via the receive-and-ack activity. This activity features two activity roles itself and the peripheral actions required to play the receiver role are distributed over both roles. In addition, some internal actions are necessary. Concerning Petri net semantics, transitions for peripheral actions have to be synchronized with transitions for actions at the level of the embedding system unit as it is illustrated in the figure. As mentioned earlier in this article, the high level Petri net formalism of reference nets exactly offers both the possibilities of net nesting and transition synchronization between horizontally adjacent nets. It is no problem to obtain arbitrarily deep nesting of actors and arbitrarily long synchronization chains between actions of multiple system units. In the example, it might for example be possible that the actor playing the role checker in the context of B is again a system unit as a collective actor. Consequently, its actions to implement the checker role would again be peripheral actions that would have to be carried out by *its* embedded actors.

Basically, we arrive at an operational understanding of collective agency. In the example, we have two actors that act on behalf of a system unit and together implement an activity role that the system unit as whole plays at an even higher system level. This understanding of collective agency finally allows us to model systems of systems based on nested actor systems and Figure 12 exemplifies an extract of an overall system that evolves around one specific system unit A. In the figure, different synchronization lines are illustrated,

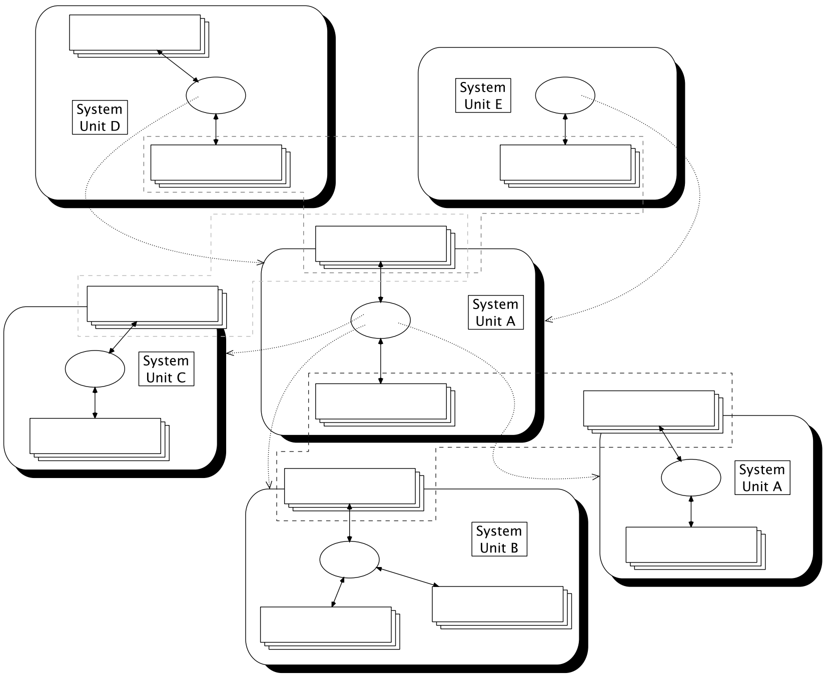

Fig. 12: Simple system of systems setting based on system units

showing different cases of activity layering for the sake of collective agency. The figure also illustrates that actor nesting needs not be unique and disjoint. It is possible that actor A is both embedded in D and E. Consequently, there exists a connection between D and E that these two might not even be aware of.

5 Conclusion and Outlook

In this paper, we have presented a modular approach to comprehend systems of systems by means of composing system units. Each system unit may be regarded under a platform perspective, where it offers environment frames for its inhabitants. Furthermore the same unit may be regarded under a collective agency perspective, where it collectively acts as a holistic entity in the context of a higher-level system unit. The model is based on Petri net semantics which gives a clear view of how to operationalize it. However, we have also introduced a variety of abstraction mechanisms that allow to smoothly transition from low-level to high-level system views and back.

In the introduction, we have described how the work presented in this paper fits into the wider context of our research. Figure 2 from the introduction illustrates how the modeling approach presented in this paper fits between our abstract proposal of a reference architecture for MOS and our fully executable operational models. Here, we want to be a bit more specific concerning this fit by being more specific concerning the architecture and the operational/executable model. This illustrates the gap that we closed with the work in this paper. For the reference architecture, we have qualitatively distinguished departments, organizations, organizational fields and the society as iteratively embedded specific types of system units. A coarse overview of the proposal is given in Figure 13.

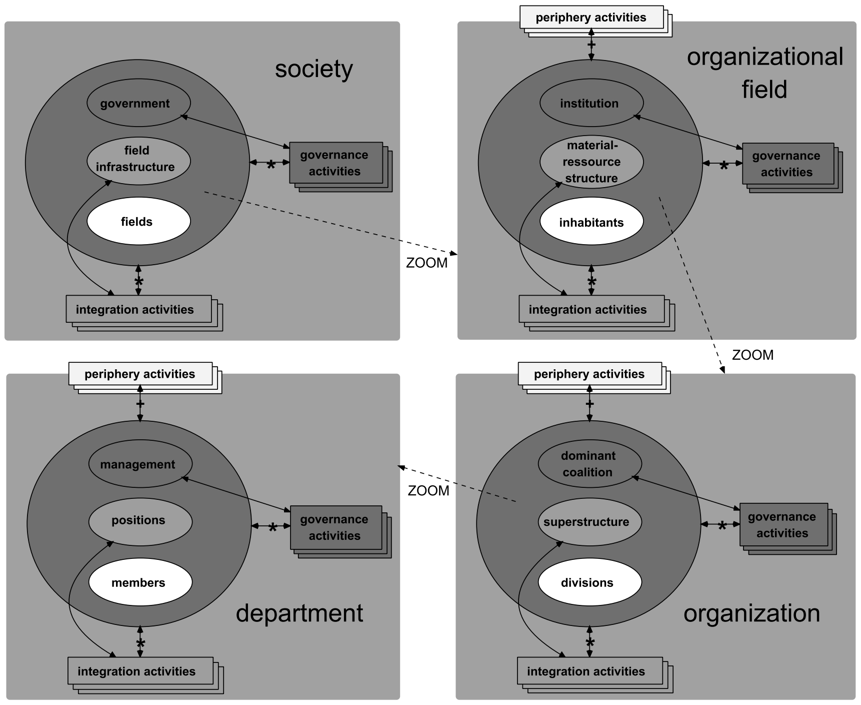

Fig. 13: Architectural proposal for multi-organization systems

In the opposite direction, we have based our proposal for operationalizing collective actor systems on reference net semantics. Its most basic idea is shown in Figure 14. Here, system activities rest on a specific place and for action execution, a synchronization between actor and activity nets has to be carried out. Internal actions are distinguished from peripheral actions. For internal actions, a system unit calls an internal actor via an :act()-channel in order to be synchronized with an activity (called via an :iStep()-channel). In the peripheral action case, things are similar but with the addition that now the system unit itself is also called via its own :act() channel by its surrounding system unit (and now the activity is called via a :pStep()-channel). The figure exemplifies one particular case where an action

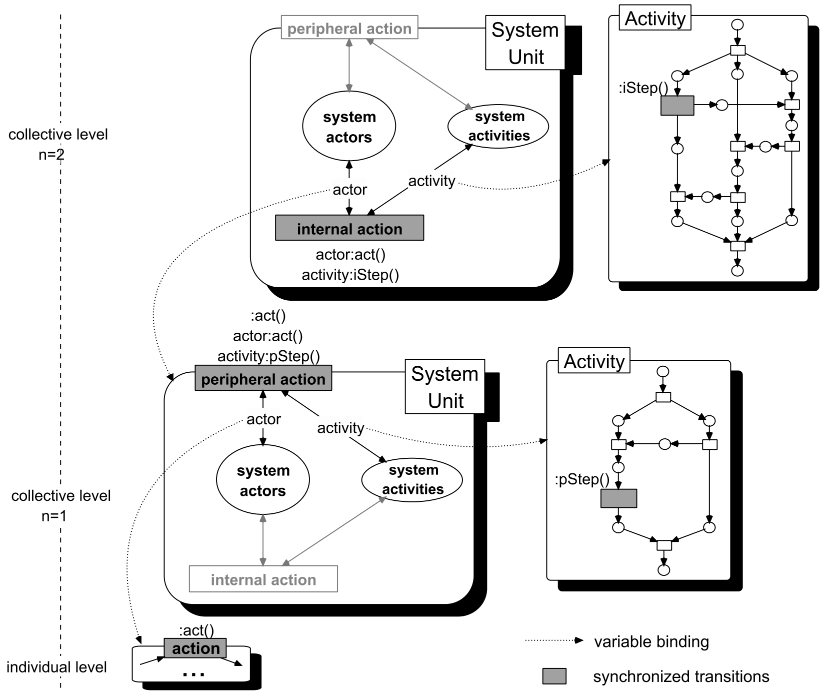

Fig. 14: Operationalization of system units: An action occurrence

occurs that is an internal action for one system unit and a peripheral action for its embedded system unit. However, the figure still lacks a considerable amount of necessary operational details.

References

[AG09] AOS-Group. Jack Intelligent Agents Team Manual. Available at: http://www.aosgrp.com/documentation/jack/JACK_Teams_Manual_WEB/index.html, 2009.

[Ber56] Ludwig von Bertalanffy. General System Theory. In Ludwig von Bertalanffy and Anatol Rapoport, editors, *General Systems: Yearbook of the Society for the Advancement of General Systems Theory*, volume 1, pages 1–10. Ann Arbor, MI: The Society, 1956.

[BHS07] Olivier Boissier, Jomi Hübner, and Jaime Simao Sichman. Organization Oriented Programming: From Closed to Open Organizations. In G. O'Hare, A. Ricci, M. O'Grady, and O. Dikenelli, editors, *Engineering Societies in the Agents World VII*, volume 4457 of *LNCS*, pages 86–105. Springer, Heidelberg, 2007.

[Boc03] Conrad Bock. UML 2 Activity and Action Models. *Journal of Object Technology*, 2(5):43–53, 2003.

[BvdT05] Guido Boella and Leendert van der Torre. Organizations as Socially-Constructed Agents in the Agent-Oriented Paradigm. In Marie Pierre Gleizes, Andrea Omicini, and Franco Zambonelli, editors, *Engineering Societies in the Agents World V*, volume 3451 of *Lecture Notes in Computer Science*, pages 1–13. Springer Verlag, 2005.

[CMR03] Lawrence Cabac, Daniel Moldt, and Heiko Rölke. A Proposal for Structuring Petri net-based Agent Interaction Protocols. In Wil M. P. van der Aalst and Eike Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 102–120. Springer Verlag, 2003.

[FSS03] Klaus Fischer, Michael Schillo, and Jörg Siekmann. Holonic Multiagent Systems: A Foundation for the Organization of Multiagent Systems. In *Holonic and Multi-Agent Systems for Manufacturing, First International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS)*, volume 2744 of *Lecture Notes in Computer Science*, pages 71–80. Springer Verlag, 2003.

[HHVE07] Andreas Hess, Bernhard Humm, Markus Voss, and Gregor Engels. Structuring Software Cities - A Multidimensional Approach. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 122–129, 2007.

[HMMF08] Christian Hahn, Cristian Madrigal-Mora, and Klaus Fischer. A Platform-Independent Metamodel for Multiagent Systems. *Autonomous Agents and Multi-Agent Systems*, 18(2), 2008.

[Jen00] Nicholas Jennings. On agent-based software engineering. *Artificial Intelligence*, 177(2):277–296, 2000.

[Kum02] Olaf Kummer. *Referenznetze*. Logos Verlag, Berlin, 2002.

[KWD09] Olaf Kummer, Frank Wienberg, and Michael Duvigneau. *Renew – User Guide*. Universit"at Hamburg, Fachbereich Informatik, Theoretische Grundlagen der Informatik, Hamburg, release 2.2 edition, 2009. Verf"ugbar "uber die Internetseite http://www.renew.de.

[LMW05] Josef Lankes, Florian Matthes, and Andre Wittenburg. Softwarekartographie: Systematische Darstellung von Anwendungslandschaften. *Wirtschaftsinformatik 2005*, 2005.

[Mai99] Mark Maier. Architecturing Principles for Systems-of-Systems. *Systems Engineering*, 1(4):267–284, 1999.

[Nor06] Linda Northrop. *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Software Engineering Institute, Carnegie Mellon, 2006.

[Röl04] Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen*. Logos Verlag Berlin, 2004.

[Sco03] W. Richard Scott. *Organizations: Rational, Natural and Open Systems*. Prentice Hall, 2003.

[Val03] Rüdiger Valk. Object Petri Nets: Using the Nets-within-Nets Paradigm. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advanced Course on Petri Nets 2003*, volume 3098 of *Lecture Notes in Computer Science*, pages 819–848. Springer Verlag, 2003.

[WEKBM08] Matthias Wester-Ebbinghaus, Michael Köhler-Bußmeier, and Daniel Moldt. From Multi-Agent to Multi-Organization Systems: Utilizing Middleware Approaches. In Alexander Artikis, Gauthier Picard, and Laurent Vercouter, editors, *International Workshop Engineering Societies in the Agents World (ESAW 08)*, 2008.

[WEM08] Matthias Wester-Ebbinghaus and Daniel Moldt. Structure in Threes: Modelling Organization-Oriented Software Architectures Built Upon Multi-Agent Systems. In *Proceedings of the 7th International Conference an Autonomous Agents and Multi-Agent Systems (AAMAS'2008)*, pages 1307–1311, 2008.

[WEM09] Matthias Wester-Ebbinghaus and Daniel Moldt. Modeling an Open and Controlled System Unit as a Modular Component of Systems of Systems. In Michael Köhler-Bußmeier, Daniel Moldt, and Olivier Boissier, editors, *Proceedings of the International Workshop on Organizational Modelling (OrgMod 2009), University of Paris*, pages 81–100, 2009.

[WEMRM07] Matthias Wester-Ebbinghaus, Daniel Moldt, Christine Reese, and Kolja Markwardt. Towards Organization–Oriented Software Engineering. In Heinz Züllighoven, editor, *Software Engineering Konferenz 2007 in Hamburg: SE'07 Proceedings*, volume 105 of *LNI*, pages 205–217. GI, 2007.