Towards Process Models for Disaster Response

Dirk Fahland[1]* and Heiko Woith[2]

[1] Humboldt-Universität zu Berlin, Institut für Informatik, Unter den Linden 6, 10099 Berlin, Germany, fahland@informatik.hu-berlin.de
[2] GeoForschungsZentrum Potsdam, Telegrafenberg, 14473 Potsdam, Germany

Abstract. In the immediate aftermath of a disaster routine processes, even if specifically designed for such a situation, are not enacted blindly. Actions and processes rather adapt their behavior based on observations and available information. Attempts to support these processes by technology rely on process models that faithfully capture process execution and adaptation. Based on experiences from actual disaster response settings, we propose to specify an *adaptive process* as a set of scenarios using a Petri net syntax. Our operational model provides an adaptation operator that synthesizes and adapts the system behavior at run-time based on the given scenarios. An example illustrates our approach.

Key words: process model, adaptation, scenarios, Petri nets, disaster response

1 Introduction

The fairly general notion of a process as a computational entity in a distributed system has found its application, among others, in workflows and process management systems. Here a *process model* relates atomic tasks, data and resources; the runs of the model mimic the behavior observed in reality. A correct model aids in planning and running complex processes while optimizing the use of resources. The validity of the model depends on assumptions, e.g. about the availability of data and resources and the possibility to enact tasks in a specific way.

Disaster response comprises all behavior immediately after a disastrous event with the aim of preventing loss of life and restoring order [5, pp.57]. Even at an intuitive level of understanding it becomes apparent that assumptions of a process or its model are likely to be violated in such situations; deviations are the rule. An actual process in this context is *adaptive* and reacts on critical changes in its environment by changing the process itself as it is being executed.

A feasible model of such a process must not only mimic these adaptation dynamics, called *instance level changes* [24]. The process model itself must be adaptable at run-time if the overall dynamics require sufficiently different process behavior, i.e. *type level changes* [24]. For this reason the model has to be *humanly conceivable* at any point in time to allow these changes.

* The author is funded by the DFG-Graduiertenkolleg 1324 “METRIK”.

In this paper, we present a novel approach for modeling adaptive processes. Rather than changing a static process model we propose to adaptively synthesize process behavior from scenarios. A *scenario* specifies a possible course of (future) actions and the therein involved resources in the context of a larger system. Whether a scenario suits a given situation is specified in a behavioral precondition. A process model then is a set of scenarios describing sequentially connected, concurrent, mutually exclusive, and overlapping behavior.

A run of the model is synthesized during execution by concatenating, merging and removing scenarios depending on observed behavior and their respective preconditions. Moreover, the set of scenarios that participates in the synthesis can easily be changed during the run. Our approach has a formal, operational semantics based on Petri nets [23]. Petri net-based process models are used in existing process management systems, for instance YAWL [3]. Our approach reflects some requirements from the disaster response domain, but our results should extend to other domains as well.

The remainder of this paper is organized as follows. In Sect. 2 we provide an introduction to disaster response and requirements for process models in this setting. In Sect. 3, we present our approach for scenario-based process models and sketch its possible use for process adaptation in Sect. 4. We discuss our model regarding literature and conclude in Sect. 5.

2 Requirements for acting during disaster response

Based on common sense and media, one has an intuitive understanding of the characteristics of a disaster and of its implications of acting in the immediate aftermath of a disastrous event. In the following we want to substantiate this understanding based on results from published literature and our experiences in an ongoing case study.

2.1 A requirement analysis for disaster response

In a requirements analysis based on nine empirical studies of disasters and disaster responses Jul [16] reflects American disaster management practices and classifies working in disaster response. She investigates how the characteristics of a disaster event influence acting in a response. We summarize those of her results which we consider as related to processes and process models.

Jul has found that because the knowledge about an event may arrive only as response is underway (or even later) the number and kind of involved organization changes. A notable effect is that '*established organizations may partner to form emergent organizations to address specialized demands.*' In these cases suitable operational structures and procedures are developed as the event evolves. [16, pp.140]

Individual responders may have to enact *agent-generated* tasks (based on their skills and experiences) as wells as *response-generated* tasks (due to a specific situation) concurrently. Depending on the event's *anticipability*, '*responders*

as well as responding organizations may need to develop new procedures and structures, and my be working in unexpected settings.' [16, pp.144]

Jul concludes from her analysis, among others, that response technology (1) '*should seek to aid response-driven tasks, such as planning, coordination and resource management*'; and (2) '*must, insofar as possible, allow flexibility and deviation in their application*' while '*imposing standard structures and procedures*' [16, pp.145]. We could confirm her findings in a concrete case-study.

2.2 Specific requirements from a concrete case-study

The authors conduct a joint case study with the "German Task Force for Earthquakes" co-ordinated by the GeoForschungsZentrum Potsdam, the German research center for geosciences. The task force '*was founded on March 22, 1993, jointly by geoscientists, civil engineers, sociologists, search and rescue specialists, and experts from the insurance industries. The major purpose of the [task force] is to co-ordinate the allocation of an interdisciplinary scientific-technical expert team after catastrophic earthquakes worldwide.*' [21]

The case study aims on (1) understanding actual work procedures of the task force during disaster response (2) finding which aspects can be supported by process models, and (3) propose suitable concepts and technology to do so. The case study is still ongoing, but we want to explain our findings so far confirming and extending the analysis of Sect. 2.1.

The task force is activated immediately after a natural disaster has occurred, collects necessary information about the event and its location, decides about, and prepares for an in-field mission. In case of a mission, at most a week after the initial event, task force members begin collecting and analyzing data at the disaster site including seismic data of aftershocks, post-seismic deformation, hydrogeological data, and the damage distribution as well as structural conditions of buildings. The results support local decisions and provide a scientific basis for an improved intermediate and long term mitigation of earthquake effects, and improve existing theories and their application.

At the site the team members naturally follow their agent-generated tasks for which they are qualified by their expertise as a geoscientist or engineer. At the same time they enact response-generated tasks; this involves for instance organizing accommodation, transport, and communication or maintaining mission critical equipment. By cooperation with other organizations like disaster response groups or military personnel the team members gain professional and organized support. In turn they can use their experienced background to support rapid decisions in coordinating rescue and damage mitigation efforts.

We are currently developing a process model. We succeeded in developing a static model for the initial preparation phase; the current model has about 120 tasks. But we found that this kind of model is infeasible for specifying the work at the disaster site for the following reasons: (a) Tasks and sub-processes are event driven. (b) A task may vary on its prerequisites, effects, duration, and costs. (c) Sub-processes (sometimes executed repeatedly) overlap and synchronize on common data and tasks. (d) As team members work spatially distributed

and communication is unreliable, decision making is also based on the possibility to make them jointly, or not. (e) Agent-generated tasks and response-generated tasks interfere with each other; each team member has to come up with a suitable, individual procedure as the situation evolves.

This result is not surprising in general and confirms a working hypothesis of most works on process adaptation and models for adaptive processes, e.g. [4, 6, 8, 22]. However, we were able to model well-structured fragments of the process: The task force members could provide information about their working through story-telling. Such a story isolates a specific aspect of the process dynamics and is part of the "acting recipes" of the task force members. These recipes are chosen, instantiated, and followed in a concrete context. We turn this fact into a process modeling paradigm in the following and discuss its features wrt. existing approaches thereafter.

3 Oclets – a formal model for adaptive processes

The requirements analysis in the previous section shows that the kind of processes we wish to model is subject and object to very complex dynamics. A static process model is likely to be incomplete and will exhibit a high complexity. Scenario-based specifications are one possibility to diminish behavioral complexity. We propose to adopt this paradigm for process models subsequently.

3.1 Scenario-based specifications

The paradigm of scenarios is widely accepted in protocol specifications using *message-sequence charts*. *Life-sequence charts* (LSCs) take this paradigm one step further by decomposing *behavior* of highly complex, distributed systems into reasonably-sized artifacts. [14]

An LSC specification consists of a set of charts, each specifying a scenario. A "normal" LSC denotes a partial, partially ordered run of a system over local events and exchanged messages. An *anti*-LSC specifies behavior that must not occur while allowing all other. A LSC has a *behavioral precondition* denoting which events have to be observed in which (partial) order to *activate* the chart. An active chart is violated if the system exhibits behavior that is not specified in the chart. A set of LSCs specifies the set of all runs where these charts are not violated.

The LSC technique is declarative in the sense that a specification classifies an observed run as valid or invalid. It has no operational semantics that allows to generate the set of all valid runs of an arbitrary LSC specification; though this is possible for subclasses [13].

3.2 Oclets – adopting scenarios to Petri nets

We want to use the paradigm of scenarios to model highly dynamic processes. We adopt the concepts of LSCs to Petri nets and supplement them with an

operational semantics that allows constructing all valid, partially-ordered runs of the system. In [11] we provide a formal model of our approach together with a proof of correctness. Subsequently, we given an intuitive explanation of the concepts and operations. A basic understanding of place/transition Petri nets will be helpful; an introduction is given in [23].

Oclets. We formalize a scenario as an *oclet* which is a finite, acyclic, labeled Petri net with some further annotations; Fig. 1 depicts some examples. As usual, a box depicts a transition which, in an oclet, denotes the *execution of a task*. A circle depicts a place, here denoting the *presence of a physical or virtual resource*. Arcs denote which resources are consumed and which resources are produced by executing a task. In this interpretation, an oclet denotes a scenario.

An oclet's *precondition*, a causally closed prefix, is depicted above the dashed line. It denotes the behavior that has to be observed to *activate* the oclet. The remaining part of the oclet describes its *contribution*: a *normal* oclet denotes a possible further execution while an *anti-oclet* disallows the execution; see Fig. 6.

In the following we construct complex behavior from oclets by repeatedly concatenating, merging, and removing oclets based on their preconditions. An *adaptive process* $A = \langle O, P_0 \rangle$ is a set O of oclets with a set P_0 of labeled places (describing the initial state). As an example consider the oclets of Fig. 1 and place p_{idle} with label idle constituting $A_1 = \langle \{o_1, o_2, o_3\}, \{p_{\mathsf{idle}}\} \rangle$; we denote a place or transition by its label subsequently.

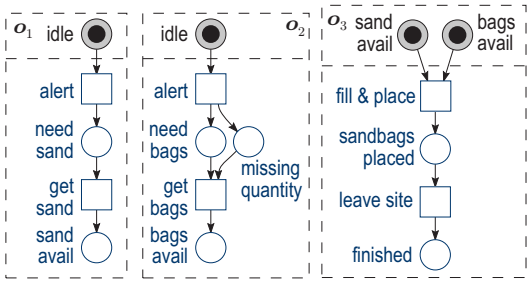

Fig. 1. Oclets o_1, o_2, o_3 of the adaptive process A_1

A_1 specifies a process of a response team in case of a flooding alert to build a sand bag barrier. A first requirement is to get sand after an alert (o_1). The team shall also get bags according to the missing quantity (o_2). Oclet o_3 denotes that as soon bags and sand are available, the bags shall be filled with sand and placed. After the sandbags are placed, the team may leave the site. Note that this example is not based on concrete data and might be too simplistic for an actual process. We extend it as we proceed in explaining our concepts.

3.3 Constructing partially-ordered runs from scenarios

We describe the behavior of an adaptive process by branching processes. Informally, a *branching process* (BP) relates to a partially ordered run like an execution tree relates to a sequential run: A sequential run denotes process behavior as a sequence of tasks by interleaving concurrent tasks; merging sequential runs at common prefixes yields an execution tree. A *partially ordered run* explicitly

denotes concurrency of tasks. Hence a BP is a directed acyclic graph that distinguishes causal ordering, concurrency and conflict of tasks. It is an important result in Petri net theory that a BP of a Petri net N can itself be represented as a Petri net β_N s.t. any run of β_N is a run of N. [9]

Fig. 2. Initial state s_0.

A *state* $\langle \beta, m \rangle$ of an adaptive process A is a Petri net-BP β together with a reachable marking m of places of β. We explain this notion of state and its relation to oclets in the following. The *initial state* of an adaptive process $A = \langle O, P_0 \rangle$ is the BP consisting of the places P_0 that marks each place $p \in P_0$ with one token, $m(p) = 1$. In our example, the initial state s_o of A_1 consists of the marked place p_{idle} only, see Fig. 2.

The BP of a state $\langle \beta, m \rangle$ can be extended or reduced by *applying* an oclet o which yields a step $\langle \beta, m \rangle \xrightarrow{o} \langle \beta', m \rangle$. Since the BP is a Petri net, its marking m may enable a transition t of BP allowing the step $\langle \beta, m \rangle \xrightarrow{t} \langle \beta, m' \rangle$.

Adding behavior with state-based preconditions. An oclet o that has a precondition consisting only of a set of places is *enabled* in a state $\langle \beta, m \rangle$ if the precondition of o is contained in the current marking m. To be precise, we need an injective mapping h from equally labeled places of the precondition of o into a set Y of marked places of β. We call Y *enabling set* and h an *embedding* of o's precondition into β. Our example oclets o_1 and o_2 are enabled in s_0 of Fig. 2

If o is enabled in state $\langle \beta, m \rangle$ then we may perform the *adaptation step* $\langle \beta, m \rangle \xrightarrow{o} \langle \beta', m \rangle$ by adding the contribution of o, i.e. o minus its precondition, to β. Formally, append a copy of o's contribution to the enabling set and iteratively merge a newly added node with an existing node if both have equal labels and the same predecessors. This may result in merging all new nodes with existing ones. Our formal definitions in [11] are more involved and attribute an oclet as enabled only if the corresponding adaptation step changes β; here we continue at a more intuitive level.

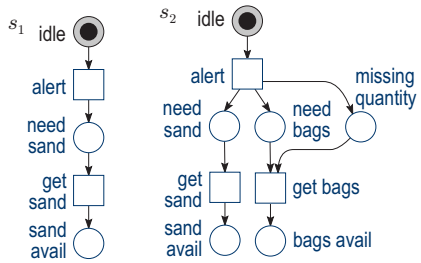

Fig. 3. The step $s_1 \xrightarrow{o_2} s_2$ of A_1.

In our example, oclets o_1 and o_2 are enabled in s_0. The result of the step $s_0 \xrightarrow{o_1} s_1$ is depicted on the left in Fig. 3. In s_1, o_2 is still enabled and the step $s_1 \xrightarrow{o_2} s_2$ adds o_2 and merges the transitions alert that originate in o_1 and o_2, respectively; see Fig. 3. In s_2 transition alert is enabled and may fire: The *transition step* $s_2 \xrightarrow{\mathsf{alert}} s_3$ consumes the token on idle and produces a token on need sand, need bags, and missing quantity (which is standard Petri net semantics).

For a consistent semantics, we prioritize adaptation steps over transition steps and require to *apply all enabled oclets* before (nondeterministically) firing one enabled transition. With this definition, we could easily construct and complete the (only) run of our example model A_1.

Now assume that in our example it may actually happen that the response team can only get some bags instead of all, but that it continues with the process afterwards. Oclet o_4 of Fig. 4 specifies this behavior. We include o_4 in our adaptive process; o_4 is enabled in s_3 and the step $s_3 \xrightarrow{o_4} s_4$ adds get some bags in conflict to get bags, see Fig. 4. Both transitions are enabled in s_4, but only one of them may fire. The steps $s_4 \xrightarrow{\text{get some bags}} s_5 \xrightarrow{\text{get sand}} s_6$ mark sand avail and bags avail which enables o_3 to finish the process.

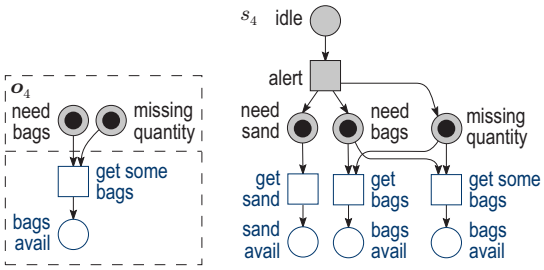

Fig. 4. Oclet o_4 and state s_4 after applying o_4

However, o_3 does not suit the slightly changed process as its enabling depends on the marking of s_6 alone. By strictly looking at the process definition, the response team would finish its work even if the number of bags was insufficient. Thankfully, this is not the case in reality. An actual response team would rather assess the *history* of its actions and *adapt forthcoming tasks and behavior* if necessary.

Oclet o_5 of Fig. 5 specifies the following scenario: If only some bags could be retrieved, the response team determines the missing quantity as the bags are filled and placed. Then the team is notified that sand is still available but some more bags are needed; the team gets the latter.

Adding behavior with history-based preconditions. The precondition of oclet o_5 in Fig. 5 is more complex and also denotes a partial execution that must have led to the current state in order to activate the oclet. Generalizing our earlier definition, an oclet o is *enabled* in a state $\langle \beta, m \rangle$ if the entire precondition of o is contained in β (including arcs) s.t. the marked places of o are contained in the current marking m. Formally, the *enabling embedding* h of o into β has to be an arc-preserving injection between equally labeled nodes of o and β. The definition of an adaptation step remains as before.

If we include o_5 in our adaptive process, o_3 and o_5 are enabled in state s_6. If we had fired get bags instead of get some bags oclet o_5 would not be enabled. The mandatory adaptation steps $s_6 \xrightarrow{o_3} s_7 \xrightarrow{o_5} s_8$ yield the state s_8 that enables transition fill&place. By the adaptation step of o_5, transition fill&place gets a new post-place missing quantity according to our adaptation semantics; the result of $s_8 \xrightarrow{\text{fill\&place}} s_9$ is depicted in Fig. 5. *Thus, depending on the history of the process execution the transition* fill&place *has different effects.*

Oclet o_5 introduced task get bags again which now also depends on task notify team. Observe that notify team and leave site are in conflict. The latter belongs to the "all went well"-scenario of oclet o_3 while the former was introduced by the "missing resource"-scenario of o_5. Both scenarios make contradicting statements about how to continue after the sandbags have been placed. There are

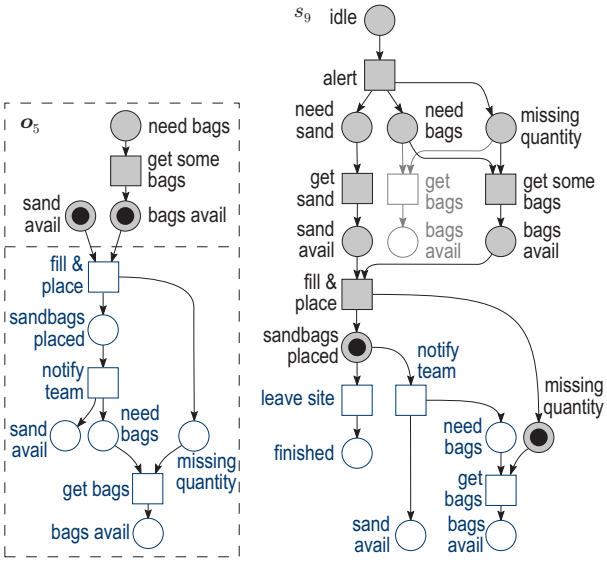

Fig. 5. Oclet o_5 and state s_9 after applying o_5

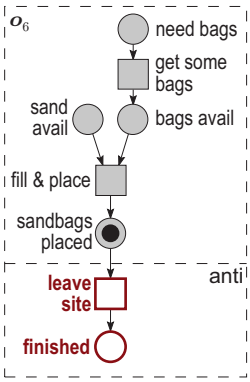

Fig. 6. Oclet o_6

several ways how to resolve this contradiction while keeping the process sound; we propose one in the spirit of scenarios.

In an actual process, the response team would know that the "all went well"-scenario was violated; it would not leave if only some bags could be retrieved. We use *anti-oclets* as mentioned in Sect. 3.2 to *explicitly forbid tasks and behavior depending on the execution history*. The anti-oclet o_6 in Fig. 6 specifies that after only some bags were retrieved and filled, the team *must not* leave the site. We include o_6 in our example process.

Removing behavior with history-based preconditions. The enabling condition for anti-oclets is essentially the same as before; oclet o_6 is enabled in state s_9 of Fig. 5.

If an anti-oclet o is enabled in state $\langle \beta, m \rangle$ then the *adaptation step* $\langle \beta, m \rangle \xrightarrow{o} \langle \beta', m \rangle$ removes the contribution of o from β. Formally, extend the enabling embedding h of o's precondition in β to map a maximal prefix of o into β. These are the nodes of o's contribution which are actually present in β; remove them from β and iteratively remove all nodes of which a predecessor was removed.

In our example, the adaptation step $s_9 \xrightarrow{o_6} s_{10}$ removes transition leave site and place finished from s_9. Now notify team is the only enabled transition (because leave site is no longer present). After the step $s_{10} \xrightarrow{\text{notify team}} s_{11}$ places sand avail, need bags, and missing quantity are marked.

For semantical consistency, we first perform *an adaptation step for each enabled normal oclet* (adding all possible behavior), then *an adaptation step for each enabled anti-oclet* (removing all infeasible behavior) and then *fire one enabled transition* (nondeterministically choosing one possible, feasible behavior).

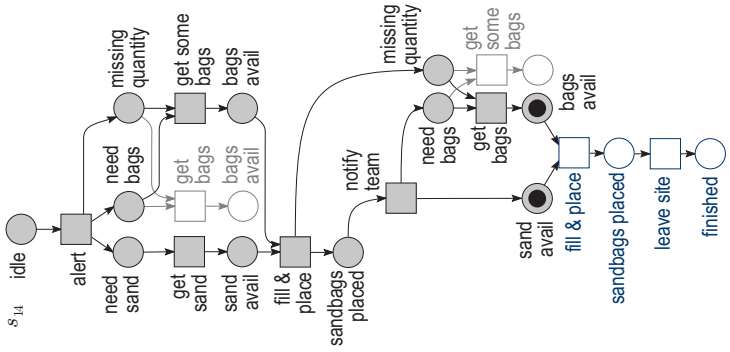

Fig. 7. State s_{14} depicting the result of a behavior synthesis from $o_1, \ldots, o_6$

This definition concludes the basic concepts of our approach for adaptive processes. Let's finish our example: Because o_4 is enabled in s_{11} we have to adapt $s_{11} \xrightarrow{o_4} s_{12}$. Again, the team could either get some bags or all bags wrt. the missing quantity. The former transition yields a state which is equivalent to s_6 (wrt. execution history): the same oclets and transitions get enabled. *This way oclets express loops in process executions.* We assume that this time the team gets all bags, $s_{12} \xrightarrow{\text{get bags}} s_{13}$ enabling o_3 only, and does not enter the loop again. The result of $s_{13} \xrightarrow{o_3} s_{14}$ is shown in Fig. 7. From there the steps $s_{14} \xrightarrow{\text{fill\&place}} s_{15} \xrightarrow{\text{leave site}} s_{16}$ get the process into its final state, finished.

4 Using oclets to realize adaptive processes

We have just presented the basic concepts of our approach for modeling processes with scenarios. In this section we want to sketch how these concepts can be used in modeling adaptive processes.

We already mentioned in the introduction that we understand an actual process to be *adaptive* if it reacts on critical changes in its environment by changing the process itself. This definition is cyclic. Splitting the cycle helps: (1) the behavior of an adaptive process can change at run-time and (2) the change of the behavior is triggered by the process. We explain how our model helps in realizing (1) and spend some ideas on (2).

Considering (1), in our case the *process type* of an adaptive process is given by a set of oclets, see Sect. 3.2. A *process instance* is a state of an adaptive process, being a branching process and a reachable marking as explained in Sect. 3.3. It includes its current run-time state, its execution history, and a part of its future behavior. Our semantic model synthesizes, and extends, an instance from the set of oclets at a time and executes it. Changing the set of oclets changes the future process behavior, as exercised in the previous section.

In our approach, a *type level change* is realized by changing a set – of oclets. Moreover, it smoothly yields *instance level changes* as an instance is incremen-

tally synthesized from a set of oclets at a time. In practice it might be necessary to use instance migration strategies when changing the process type s.t. different instances have different sets of oclets which eventually converge [24].

Regarding (2), our model allows to trigger behavior change in closed systems as follows. Assume two different oclets o_K and o_L having the same precondition where o_K suits some context K and o_L suits some other context L. The adaptive process $A_K = \langle O \uplus \{o_K\}, m\rangle$ shall be feasible in K; a change to $A_L = \langle O \uplus \{o_L\}, m\rangle$ makes it feasible for L. This kind of change must be triggered externally when someone observes a context change from K to L.

The oclets O can be extended to distinguish contexts K and L; we demonstrated this in Sect. 3.1 when introducing o_4. Let O^+ denote these oclets. In the same way, the preconditions of o_K and o_L can be extended differently to distinguish contexts, let this be o_K^+ and o_L^+; oclets o_3 and o_5 of Sect. 3.3 are an example. Now the process $A^+ = \langle O^+ \uplus \{o_K^+, o_L^+\}, m\rangle$ distinguishes context K from L and chooses the appropriate behavior. What was an explicit adaptation of behavior from A_K to A_L is now gracefully included in the process dynamics.

5 Conclusion

We have presented an approach to model adaptive processes that can change their behavior at run-time. Such processes arise for instance during disaster response. We presented conceptual and concrete requirements for process models in this setting which exceed the capabilities of static process models.

We proposed to model an adaptive process in scenarios with an intuitively understandable Petri net syntax which we formalized as oclets. We contributed an operational adaptation step that synthesizes the behavior of the process from its oclets at run-time. The synthesis depends on the process execution and may add, remove, or modify future behavior. This effectively leads to a run-time adaptation of process dynamics. Further, the process model can be changed at run-time to adapt to process-external changes. We demonstrated the concepts of our approach in an example from the emergency response settings.

The semantic model for our approach are Petri net branching processes. It is operational and may be verified using efficient model-checking techniques [10].

Discussion and Related Work. The problem of adapting process behavior has been researched from various angles. Many works consider the adaptation of workflows by run-time application of transformation rules on a static process model [4, 6, 8, 22], or by a notion of relating an old system specification to a new system specification to guarantee a correct replacement [1]. Some of these approaches, e.g. [22], are more expressive than our approach. But our oclets make the condition when to perform an adaptation step an explicit part of the operational system specification. This is not the case in the mentioned works, where process adaptation must be triggered externally.

Hee et al. demonstrated how to make process dynamics dependent on execution history in a Petri net model using transition guards [15]. We contribute an

intuitive Petri net-notation for history-dependent dynamics; the actual relation between both models deserves further research.

Petri net theory has developed a number of models that allows operationally changing Petri nets at run-time by putting the net to be changed as a token into a high-level net that does the change [17]. Adaptations have been formalized as direct operations on nets [12] and as graph transformations based on rules that allow constructing system behavior from scenarios [7]. Dynamically adaptive systems can also be expressed with various sorts of process algebras, for instance χ [2] or the π-calculus [20], including communication and changes in topologies. But like in the approaches mentioned above, the operations to adapt process behavior are an explicit part of the process definition. The "worklet" concept of YAWL allows adding encapsulated sub-workflows ("worklets") during process execution in a hierarchical workflow model [3].

Our result suggests that adaptive process dynamics need no hierarchies and no explicit adaptation operation to be part of the process model. The modeler declares the behavior she wants to include or exclude together with a precondition; the operational behavior is synthesized at run-time. The advantage or disadvantage of having no hierarchies depends on the actual process.

In this direction, Leoni et al [18] propose a situation calculus based on second-order logic to model processes. Their approach includes a control-loop scheme where the process correctly changes its behavior based on actual observations, including unforseen events. This is not covered by any other approach, including ours. A possible drawback is its text-based specification language that might obfuscate an intuitive understanding of the process.

We already mentioned that the concepts of our approach originate in live-sequence charts [14] which are an entirely declarative technique, while we provide an operational model. We consider a detailed comparison as future work.

We also see another contribution of our approach. The results of Mendling et al [19] suggest that the size of a process model and '*the number of arcs [has] an important influence on understandability.*' They also found that '*small variations between models can lead to significant differences in their comprehensibility.*' We take this as an indicator that our approach of modeling with well-conceivable process fragments has taken a reasonable direction.

Future Work We are currently implementing our concepts and the algorithms in a proof-of-concept run-time environment for adaptive processes. This tool will help us in validating our approach. We are researching the formal properties of our approach that help in verifying such processes. We are also investigating which further concepts we need to include in our model to achieve more expressivity. Benchmarks are our case study from the disaster response domain and practically relevant adaptation patterns as identified in [24].

References

1. W.M.P. v.d. Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.*, 270(1-2):125–203, Feb 2002.

2. J.C.M. Baeten abd D.A.v. Beek and J.E. Rooda. *CRC Handbook of Dynamic System Modeling*, chapter 19: Process algebra, pages 19.1–21. Chapman & Hall, 2007.
3. M. Adams, A.H.M. t. Hofstede, D. Edmond, and W.M.P. v.d. Aalst. Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In *OTM Conferences (1)*, pages 291–308, 2006.
4. A. Agostini and G. De Michelis. Improving Flexibility of Workflow Management Systems. In *LNCS*, volume 1806, pages 218–234. Springer-Verlag, 2000.
5. W. Nick Carter. *Disaster Management: A Disaster Manager's Handbook*. Asian Development Bank, 1991.
6. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 438–455, 1996.
7. H. Ehrig, K. Hoffmann, and J. Padberg. Transformations of Petri Nets. *Electr. Notes Theor. Comput. Sci.*, 148(1):151–172, 2006.
8. C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *COCS'95*, pages 10–21. ACM Press, 1995.
9. J. Engelfriet. Branching processes of Petri nets. *Acta Inf.*, 28(6):575–591, 1991.
10. J. Esparza and K. Heljanko. *Unfoldings - A Partial-Order Approach to Model Checking*. Springer-Verlag, 2008.
11. D. Fahland. Oclets - a formal approach to adaptive systems using scenario-based concepts. Informatik-Berichte 223, Humboldt-Universität zu Berlin, 2008.
12. B. Farwer. Recovery and reset in object petri nets with process markings. In *Proceedings of CS&P 2006*, pages 47–57, Sept. 2005.
13. D. Harel and H. Kugler. Synthesizing State-Based Object Systems from LSC Specifications. In *LNCS*, volume 2088, pages 1–33. Springer-Verlag, 2001.
14. D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., 2003.
15. K.M.v. Hee, A. Serebrenik, N. Sidorova, M. Voorhoeve, and J.M.E.M.v.d. Werf. Modelling with history-dependent petri nets. In *LNCS*, volume 4714, pages 320–327, 2007.
16. S. Jul. Who's Really on First? A Domain-Level User, Task and Context Analysis for Response Technology. In *ISCRAM 2007*, pages 139–148, 2007.
17. M. Köhler and H. Rölke. Reference and value semantics are equivalent for ordinary object Petri nets. In *LNCS*, volume 3536, pages 309–328. Springer-Verlag, June 2005.
18. M. de Leoni, M. Mecella, and G. De Giacomo. Highly dynamic adaptation in process management systems through execution monitoring. In *LNCS*, volume 4714, pages 182–197. Springer-Verlag, 2007.
19. J. Mendling, H.A. Reijers, and J. Cardoso. What makes process models understandable? In *LNCS*, volume 4714, pages 48–63, 2007.
20. R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, June 1999.
21. GFZ Potsdam. Mission statement of the "German Task Force Earthquakes". http://www.gfz-potsdam.de/pb2/pb21/Task_Force/index_e.html, 15th May 2008.
22. M. Reichert and P. Dadam. ADEPT$_{\text{flex}}$-Supporting Dynamic Changes of Workflows Without Losing Control. *J. Intell. Inf. Syst.*, 10(2):93–129, 1998.
23. W. Reisig. *A Primer in Petri Net Design*. Springer Compass International, 1992.
24. B. Weber, S. Rinderle, and M. Reichert. Change patterns and change support features in process-aware information systems. In *LNCS*, volume 4495, pages 574–588, 2007.