Institute of Architecture of Application Systems

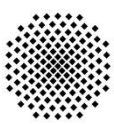
Rewinding and Repeating Scientific Choreographies

Andreas Weiß, Vasilios Andrikopoulos, Michael Hahn, Dimka Karastoyanova

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{andreas.weiss, vasilios.andrikopoulos, michael.hahn, karastoyanova}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{INPROC-2015-40,
  author    = {Andreas Wei{\ss} and Vasilios Andrikopoulos and Michael
               Hahn and Dimka Karastoyanova},
  title     = {{Rewinding and Repeating Scientific Choreographies}},
  booktitle = {On the Move to Meaningful Internet Systems: OTM 2015
               Conferences},
  year      = {2015},
  pages     = {337--347},
  series    = {Lecture Notes in Computer Science (LNCS)},
  publisher = {Springer-Verlag}
}
```

University of Stuttgart
Germany

Rewinding and Repeating Scientific Choreographies

Andreas Weiß, Vasilios Andrikopoulos, Michael Hahn, and Dimka Karastoyanova

Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Stuttgart, Germany
{andreas.weiss,vasilios.andrikopoulos,michael.hahn,
dimka.karastoyanova}@iaas.uni-stuttgart.de

Abstract. Scientists that use the workflow paradigm for the enactment of scientific experiments need support for trial-and-error modeling, as well as flexibility mechanisms that enable the ad hoc repetition of workflow logic for the convergence of results or error handling. Towards this goal, in this paper we introduce the facilities to repeat partially or completely running choreographies on demand. Choreographies are interesting for the scientific workflow community because so-called multi-scale/field (multi-*) experiments can be modeled and enacted as choreographies of scientific workflows. A prerequisite for choreography repetition is the rewinding of the involved participant instances to a previous state. For this purpose, we define a formal model representing choreography models and their instances as well as a concept to repeat choreography logic. Furthermore, we provide an algorithm for determining the rewinding points in each involved participant instance.

Keywords: Ad Hoc Changes, Choreography, Workflow, Flexibility

1 Introduction

The goal of eScience is to provide generic approaches and tools for scientific exploration and discovery [6]. The workflow technology, in this context known as *scientific workflows*, is one approach for supporting data processing and analysis. However, scientists have different requirements on workflow modeling and enactment than users in the business domain. eScience experiments often demand a trial-and-error based modeling [2] that allows the extension of incomplete models after they have already been instantiated, or their partial repetition with different sets of parameters for the convergence of results. In this context, scientists are both the designers and users of a workflow model. To support these requirements the *Model-as-you-go* approach has been introduced in [13]. The approach uses workflow technology from the business domain and adapts it for the requirements of scientists while keeping the technology's benefits such as standardization and automated error handling. One aspect of the approach is the definition of two operations allowing scientists to influence the execution of a workflow in an ad hoc manner without relying on pre-specified facilities

in the workflow model [14]. The *iterate* operation allows repeating workflow logic without undoing previously completed work. This is helpful for scientists to enforce the convergence of results by repeating some steps such as building a Finite Element Method grid with a different set of parameters. The *re-execute* operation also allows repeating parts of already executed workflow logic, however, completed work is compensated (undone) beforehand. This allows scientists to reset the execution environment e.g. in case of detected errors.

A limitation of the Model-as-you-go approach is the lack of support for multi-scale (e.g. space or time scales) and multi-field (e.g. physics and biology or chemistry), so-called *multi-**, experiments in cases where the experiment's mathematical models have not been merged. Such merging typically approximates descriptions of one or more scales/fields onto another scale/field, while sacrificing accuracy of the simulation results. In order to support the modeling and execution of multi-* experiments, in previous work [17] we proposed the use of choreographies and introduced the notion of *Model-as-you-go for Choreographies*. Choreographies are a concept known from the business domain. They provide a global view on the interconnection of collaborating parties such as business organizations. Unlike service orchestrations, choreographies do not have a centralized coordinator but represent the peer-to-peer-like interconnection between services or orchestrations of services [3]. However, flexibility features as provided for single scientific workflows are still missing. Scientists should be able to select a point in the choreography up to which the execution of the simulations has to be rewound before applying any desired changes and repeating the execution of the experiment. Repeating the execution instead of discarding all intermediate results saves a lot of time especially in case of long running scientific experiments.

Towards this goal, this work supports the notion of Model-as-you-go for Choreographies by providing the following contributions: based on the work of [7] and [14], we provide a *formal description of choreography models and instances* (Sec. 2). Subsequently in Sec. 3, we discuss the concept of repeating the execution of choreography instances and the *rewinding of choreography instances* to a previous state as preparatory step, and introduce an *algorithm to determine the rewinding points*. Sec. 4 compares our approach to related ones and Sec. 5 concludes the paper with an outlook on future work.

2 Formal Model

In this section, we define the underlying formal model for our approach corresponding to the life cycle phases *modeling* and *execution* of scientific choreographies [16].

2.1 Modeling Phase

A choreography model consists of at least two participants, which are represented by service orchestrations/process models. A process model is a directed, acyclic graph whose nodes represent activities. Control flow is explicitly modeled by control flow connectors linking activities. Data flow is implicitly described through the

manipulation of variable values as input and output of activities. The participants communicate with each other via message links. For the purposes of this work we do not consider loops inside the process models. Formally speaking, a *process model* is a DAG $G = (m, V, i, o, A, L)$, where $m \in M$ is the name of the process model (M is the set of all names), V is the set of variables, i is the map of input variables, o is the map of output variables, A is the set of activities, and L is the set of control flow connectors (control flow links). A control flow connector $l \in L$ is a triple $l = (a_{source}, a_{target}, t \mid a_{source}, a_{target} \in A, t \in C \wedge a_{source} \neq a_{target})$ connecting a source and a target activity, while its *transition condition* (where C is the set of all conditions) is evaluated during run time [7, 14]. Based on that, we define:

Definition 1 (Choreography Model $\mathfrak{C}$). *A choreography model is a directed, acyclic graph denoted by the triple $\mathfrak{C} = (m, P, ML)$, where $m \in M$ is the name of the choreography model, P is the set of choreography participants, ML is the set of message links between the choreography participants.*

A *choreography participant* $p \in P$ is a triple $p = (m, type, G)$, where $m \in M$ is the name of the participant, $type \in T$ is the type of the participant (T is the set of types), and $G \in G_{all}$ (G_{all} is the set of all process models) is a process model graph. A message link is a tuple $ml \in ML = (p_s, p_r, a_s, a_r, t)$, where $p_s, p_r \in P$ are the sending and receiving participants. For the sending and receiving participants the following holds: $p_s \neq p_r$, i.e., the sender and the receiver must not be identical; $a_s \in \pi_5(p_s.G)$ and $a_r \in \pi_5(p_r.G)$, where π_i is the projection operation on the i-th element of a tuple, are the sending and receiving activities for which holds: $a_s \neq a_r$. The transition condition $t \in C$ is evaluated during run time.

2.2 Execution Phase

Choreography models are typically not directly instantiable [3]. Instead, the process/workflow models implementing the choreography participants are instantiated. Together they form an overall virtual choreography instance. The virtual choreography instance at a given point in time can be created by reading monitoring information. We use the definitions of activity and process instances from [14] and extend them for choreography instances. Note that we describe states with the following abbreviations: (S=scheduled, E=executing, C=completed, F=faulted, T=terminated, Cmp=compensated, D=dead, Sus=suspended). Formally, a *process instance* is a tuple $p_g = (V^I, A^A, A^F, L^E, s_g)$, where V^{I} is the set of variable instances, $A^A \subseteq A^I$ is the set of active activity instances, $A^F \subseteq A^I$ is the set of finished activity instances, L^E is the set of evaluated links, and s_g is the state of the process instance. In general, the set of activity instances is defined as $A^I = \{(id, a, s, t) \mid id \in ID, a \in A, s \in \mathcal{S}, t \in \mathbb{N}\}$. During execution, an activity instance is identified by its id. The set $\mathcal{S} = \{S, E, C, F, T, Cmp, D\}$ contains the execution states an activity instance can take at any point in time t. The state of an activity instance $a^i \in A^I$ can be determined by the function $state(a^i)$, whereas its

model element is retrieved by the function $model(a^i)$. For the set of evaluated control flow links the following holds: $L^E = \{(l, c, t) \mid l \in L, c \in \{true, false\}, t \in \mathbb{N}\}$. Evaluated links are the links that already have a truth value c assigned at an execution time t. The truth value determines if the link is followed during execution. The process instance as a whole may be in one of the following states: $s_g \in \{E, C, Sus, T, F\}$. It then follows:

Definition 2 (Choreography Instance $\mathfrak{c}^i$). *A choreography instance is the triple $\mathfrak{c}^i = (P^I, ML^E, s_c)$, where P^I is the set of participant instances, ML^E the set of evaluated message links, and s_c the state of the choreography instance.*

The set of participant instances P^I contains pairs of the form $p^i = (m, p_g)$, where m is the name of the participant instance and $p_g \in P_g^{all}$ (P_g^{all} is the set of all process instances) is a process instance. For ML^E the following holds: $ML^E = \{(ml, c, t) \mid ml \in ML, c \in \{true, false\}, t \in \mathbb{N}\}$, i.e., ML^E contains the instantiated message links having a truth value c indicating the outcome of the transition condition evaluation and an execution time t. The choreography instance may be in one of the following states: $s_c \in \{E, C, Sus, T, F\}$.

3 Repetition of Choreographies

In this section we discuss our approach for rewinding a choreography instance to a previous state and we introduce an algorithm for identifying rewinding points.

3.1 Concept

A basic assumption for the *Model-as-you-go for Choreographies* approach is the existence of a monitoring infrastructure as introduced in [18] capturing the execution events, providing information about instance states of the process models distributed across different execution engines, and correlating these states with the corresponding choreography model (in the graphical modeling environment a scientist uses). For our purposes, it is sufficient that only the events related to elements already described in the choreography model are published. Figure 1 shows an example of a choreography instance in which a part of its logic is to be repeated, and summarizes the relevant terms that are used in the following. Repeating parts of the logic of choreography instances is triggered by manually choosing a *start activity instance* in a *start participant instance* (activity instance c_1 of Participant 1 in the example) during run time via a graphical modeling tool. The *choreography wavefront* contains all currently active or scheduled activity instances, control flow connector instances, and message link instances. The execution must be suspended before starting the rewinding and the repetition of logic in order to avoid race conditions [14].

An important concept is the notion of the *iteration body*. In [14], the iteration body is defined as the activity and evaluated link instances reachable from a user-selected start activity of an individual process instance. Activities that are not yet scheduled, i.e., that are located in the future of the process instance, are

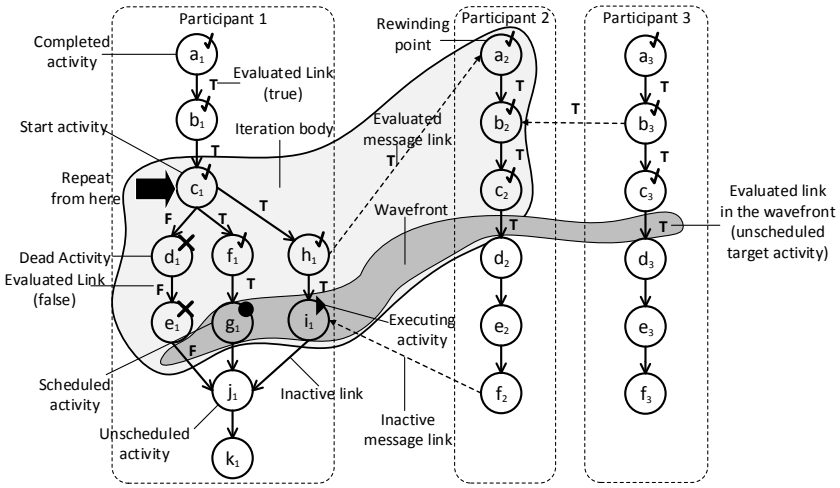


Fig. 1: Example of a choreography instance

not contained in the iteration body. In Fig. 1, the iteration body of Participant 1 are the activities c_1, d_1, e_1,f_1, g_1, h_1, i_1, and the control flow connectors between them. Note that the path $c_1 \rightarrow d_1$ has not been chosen during execution and is marked as *dead*. However, it may be chosen after starting a repetition from activity c_1. We extend the notion of iteration body for choreography instances. Here, the iteration body spans across process instances and includes message links between them. In Fig. 1, additionally to the already enumerated ones, the activities a_2, b_2, c_2, the control flow links between them, the control flow link $c_2 \rightarrow d_2$, and the message link $h_1 \rightarrow a_2$ are part of the *choreography iteration body*. The repetition of logic starting in one particular participant instance affects at least all participant instances that are part of the choreography iteration body. Already finished activities in the choreography iteration body must be *rewound*, i.e., either be *reset* (for iteration purposes), or *compensated* (to enable re-execution). *Iteration* in a choreography instance is the repetition of logic in the enacting workflow instances without undoing already completed work. The workflow instances participating in the choreography instance are collectively reset as described for individual workflow instances in [14]. *Re-execution* in choreography instances is the repeat of choreography logic after compensating already completed work.

In general, two cases can be distinguished. In the first case, the start participant instance is connected to other participant instances, which are reachable from the manually selected start activity instance (c_1 in Fig. 1). The start participant instance at this point contains completed sending activities that have sent messages to other participant instances. While the activity instance identified by the user is the *rewinding point* in the start participant instance, the rewinding points in the connected instances have to be identified separately (cf. Sec. 3.2). In the example, activity a_2 is the rewinding point of the Participant 2

instance. Rewinding points can also be found in participants that are transitively connected and reachable from the start activity instance. Each rewinding point indicates where the resetting or compensation of activities has to stop, i.e., how the choreography wavefront has to be moved to the past of the choreography instance. In the second case, participants that are not reachable from the start activity instance may still be affected by the repetition of logic in other participant instances. Messages that are not the reply of previous requests and transmitted over incoming message links to the affected participant instances must be available again in the case of repetition. This can either be done by also determining rewinding points in the sending participant instances and rewinding and repeating logic, or by storing and replaying previously sent messages by the workflow engine responsible for the respective participant instance. The determined rewinding points are sent to the involved workflow engines to trigger the iteration or re-execution of choreography logic using a transactional protocol. This is not presented here due to space constraints.

3.2 Determining the Rewinding Points

In the following an algorithm is presented to determine the rewinding points in a choreography iteration body. Therefore, a set of auxiliary functions is defined.

Definition 3 (Function *succ*). *The* successor function *succ is defined as* $succ : A^I \times A^I \rightarrow \mathbb{B}$*, where* A^I *is the set of activity instances and* $\mathbb{B}$ *is the set of boolean values* $\mathbb{B} = \{true, false\}$*.*

The function determines if the second activity instance is reachable from the first activity instance, i.e., a successor in the process instance graph. Definition 4 describes the function for finding the set of rewinding points.

Definition 4 (Function ρ). *The* Determine Rewinding Points function ρ *is defined as* $\rho : \mathfrak{C}^I \times A^I \times P^I \times RP^{all}_{\mathfrak{C}} \rightarrow RP^{all}_{\mathfrak{C}}$*, where* $\mathfrak{C}^I$ *is the set of choreography instances,* A^I *is the set of activity instances,* P^I *is the set of participant instances, and* $RP^{all}_{\mathfrak{C}}$ *is the set that contains all* $RP_{\mathfrak{C}}$ *sets.*

$RP_{\mathfrak{C}} \subseteq P^I \times \mathcal{P}(A^I)$ is a set of pairs $\{(p^i, A^I_{rp}) \mid p^i \in P^I, a^i_1, ..., a^i_k \in A^I_{rp} \subseteq A^I\}$ consisting of a participant instance and a set of rewinding point activity instances. The reason that a participant instance can have more than one rewinding point is the existence of parallel paths in the process model graph. A participant may receive messages in parallel that result in independent rewinding points.

Definition 5 (Function χ). *The* Handle Sending Activities function χ *is defined as* $\chi : \mathfrak{C}^I \times A^I \times RP^{all}_{\mathfrak{C}} \rightarrow RP^{all}_{\mathfrak{C}}$*, where* $\mathfrak{C}^I$ *is the set of choreography instances,* A^I *is the set of activity instances, and* $RP^{all}_{\mathfrak{C}}$ *is the set of pairs containing the assignment of participant instances to their rewinding points.*

Algorithm 1 and the sub-routine described in Algorithm 2 show the realization of functions ρ (Definition 4) and χ (Definition 5), respectively. The main idea of the algorithm is the following: Beginning from the user-selected start activity

Algorithm 1: determineRewindingPoints, ρ

```
1  input  : Choreography instance 𝔠^i, activity instance a^i_start, participant instance
            p^i, set of pairs RP_𝔠 = (p^i, A^I_rp)
2  output : RP_𝔠
3  begin
4      if RP_𝔠 = ∅ then
5          RP_𝔠 ← RP_𝔠 ∪ (p^i, {a^i_start})
6      end
7      Stack S ← ∅
8      S.push(a^i_start)
9      while S ≠ ∅ do
10         Activity Instance a^i ← S.pop()
11         if a^i is not marked as visited ∧ state(a^i) = completed then
12             mark a^i as visited
13             if model(a^i) is sending activity then
14                 RP_𝔠 ← handleSendingActivities (𝔠^i, a^i, RP_𝔠)
15             end
16             foreach
               l^i = (l^x, c^x, t^x) ∈ π_4(p^i.p_g) | l^x.a^x_source = a^i ∧ state(c^x) = true do
17                 S.push(l^x.a^x_target)
18             end
19         end
20     end
21     return RP_𝔠
22 end
```

instance, the start participant instance graph is traversed in a depth-first manner. Every activity instance with the state *completed* is marked as visited and its outgoing links, provided they have been evaluated to true, are followed. For each completed activity instance it is checked if it is a sending activity. If so, the sub-routine *handleSendingActivities* (χ) as defined in Definition 5 is invoked. The attached message link instance $ml^i_{traversed}$ of the sending activity instance a^i is retrieved by evaluating the following conditions: (i) it has been evaluated to true and (ii) there exists a receiving activity instance a^i_r in the *completed* state, i.e., a message has been sent and consumed. If $ml^i_{traversed}$ exists, the algorithm retrieves its receiving participant instance. For the receiving participant instance it is checked if it has already been (partly) traversed by the algorithm and a (preliminary) rewinding point has been found. If this is not the case, ρ is invoked recursively with the current receiving participant. If there exists already a rewinding point, it is checked if (i) the old rewinding point would be a successor of the new one or if (ii) both are in parallel branches. In case (i) the old rewinding point activity instance is removed before the new rewinding point is added and in case (ii) both are kept. In both cases, ρ is invoked recursively afterwards. The recursion in one participant instance stops when all reachable completed activity instances have been marked as visited.

Algorithm 2: handleSendingActivities, χ

input : Chor. instance $\mathfrak{c}^i$, activity instance a^i, set of pairs $RP_{\mathfrak{c}} = (p^i, A^I_{rp})$
output : $RP_{\mathfrak{c}}$
begin
  Message Link Instance $ml^i_{traversed} \leftarrow (ml^x, c^x, t^x) \mid (ml^x, c^x, t^x) \in ML^E \wedge$
  $ml^x = (p^i_s, p^i_r, a^i_s, a^i_r, c) \wedge a^i = a^i_s \wedge state(c^x) = true \wedge state(a^i_r) = completed$
  if $ml^i_{traversed} \neq \perp$ **then**
    Participant Instance $p^i_r \leftarrow ml^i_{traversed} \cdot p^i_r$
    if $\nexists (p^x, A^x_{rp}) \in RP_{\mathfrak{c}} \mid p^x = p^i_r$ **then**
      $RP_{\mathfrak{c}} \leftarrow RP_{\mathfrak{c}} \cup (p^i_r, \{a^i_r\})$
      $RP_{\mathfrak{c}} \leftarrow \rho\ (\mathfrak{c}^i, a^i_r, p^i_r, RP_{\mathfrak{c}})$
    end
    else if $\exists (p^x, A^x_{rp}) \in RP_{\mathfrak{c}} \mid p^x = p^i_r$ **then**
      Boolean $recursion \leftarrow false$
      foreach $a^x \in A^x_{rp}$ **do**
        if $succ(a^i_r, a^x)$ **then**
          $A^x_{rp} \leftarrow A^x_{rp} \setminus a^x$
          $recursion \leftarrow true$
        end
        else if $\neg succ(a^x, a^i_r) \wedge \neg succ(a^i_r, a^x)$ **then**
          $recursion \leftarrow true$
        end
      end
      if *recursion* **then**
        $A^x_{rp} \leftarrow A^x_{rp} \cup a^i_r$
        $RP_{\mathfrak{c}} \leftarrow \rho\ (\mathfrak{c}^i, a^i_r, p^i_r, RP_{\mathfrak{c}})$
      end
    end
  end
  return $RP_{\mathfrak{c}}$
end

4 Related Work

There are several areas related to our work, such as ad hoc repetition in process instances and adaptation of choreographies during modeling and run time. In literature, the concept of ad-hoc repetition in process instances is well studied. For example, in [10] concepts and algorithms for pre-modeled or ad hoc backward jumps, which enable the repeat of logic in process instances enacted by the ADEPT Workflow Management System are presented. The Kepler system supports the concept of smart reruns [1] enabling scientists to repeat parts of a scientific workflow with a different set of parameters. Previously stored provenance information is used to avoid the repetition of parts of the workflow that do not change the overall outcome of the scientific experiment. Similarly, in [8] a system is proposed that supports scientific workflows as well as the concept of

reruns of workflow logic for the validation of scientific results but not for enabling an explorative modeling approach as in our work. In [12], process flexibility types are classified. Our concept for rewinding and repeating choreography instances could be classified as *Flexibility by Deviation* – deviating from the specified control flow in the model. Similarly, our repetition is one form of the *Support for Instance-Specific Changes* as described in [15] for individual process instances. However, none of these works consider choreography instances.

Several works exist on the adaptation of choreography models. For example, in [11] the propagation of changes appearing in the private process of one choreography participant to the affected business partners is enabled without considering already running choreographed workflow instances. Formal methods are used to calculate the necessary changes and if the new message interchange is deadlock free. In [4], a generic approach for propagation of local changes to directly as well as transitively dependent business partners is shown. Reichert and Bauer [9] introduce a variant of the ADEPT system that combines the distributed execution of a partitioned workflow model with ad hoc modifications of the workflow instances. Changes on the model are efficiently transmitted to the involved execution engines. In [5], a concept for the evolution of distributed process fragments during run time is proposed identifying change regions and applying changes to the process fragment instances. The major difference of these works to our approach is that we do not start with changes on the level of the model but change running instances in an ad hoc manner.

5 Conclusions and Future Work

In this paper, we motivated the need for the capability to repeat partially or completely the logic in a choreography with a clear focus on the eScience community. Toward this goal, we presented a formal model for describing choreography models and instances. Based on the formal model, we introduced the concept of repeating logic in choreography instances, which involves the rewinding of process instances as a preparatory step. We distinguished between iteration, which executes logic again without undoing already completed work, and re-execution, which aims at the compensation of already completed work before executing it again. We proposed an algorithm that is able to identify the rewinding points for each involved participant instance.

In future, we plan to extend our formal model and proposed algorithm to also consider loop constructs and variable values. The integration of our proposal into the scientific Workflow Management System (sWfMS) [17] is ongoing work. While we currently do support the identification of rewinding points in the graphical environment of our sWfMS in an automated manner, the capability to rewind enacted choreographies through the environment is work in progress. As part of this effort, we also plan to evaluate our proposed algorithm in combination with monitoring data collected from executing simulation workflows in the context of the SimTech project.

Acknowledgment

This work is funded by the project FP7 EU-FET 600792 ALLOW Ensembles and the German DFG within the Cluster of Excellence (EXC310/2) SimTech.

References

1. Altintas, I., Barney, O., Jaeger-Frank, E.: Provenance Collection Support in the Kepler Scientific Workflow System. In: Provenance and Annotation of Data, pp. 118–132. Springer (2006)
2. Barga, R., Gannon, D.: Scientific versus Business Workflows. In: Workflows for e-Science, pp. 9–16. Springer (2007)
3. Decker, G., Kopp, O., Barros, A.: An Introduction to Service Choreographies. Information Technology 50(2), 122–127 (2008)
4. Fdhila, W., Rinderle-Ma, S., Reichert, M.: Change propagation in collaborative processes scenarios. In: Proceedings of CollaborateCom'12. IEEE (2012)
5. Hens, P., Snoeck, M., Poels, G., De Backer, M.: Process Evolution in a Distributed Process Execution Environment. Int. J. Inf. Syst. Model. Des. 4(2), 65–90 (2013)
6. Hey, T., Tansley, S., Tolle, K. (eds.): The fourth paradigm: data-intensive scientific discovery. Microsoft Research (2009)
7. Leymann, F., Roller, D.: Production Workflow - Concepts and Techniques. PTR Prentice Hall (2000)
8. Lu, S., Zhang, J.: Collaborative Scientific Workflows. In: Proceedings of ICWS'09. pp. 527–534. IEEE (2009)
9. Reichert, M., Bauer, T.: Supporting Ad-Hoc Changes in Distributed Workflow Management Systems. In: Proceedings of OTM'07, pp. 150–168. Springer (2007)
10. Reichert, M., Dadam, P., Bauer, T.: Dealing with forward and backward jumps in workflow management systems. Software and Systems Modeling 2(1), 37–58 (2003)
11. Rinderle, S., Wombacher, A., Reichert, M.: Evolution of Process Choreographies in DYCHOR. In: CooplS'06. pp. 273–290. Springer (2006)
12. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.: Process flexibility: A survey of contemporary approaches. In: Advances in Enterprise Engineering I, vol. 10, pp. 16–30. Springer (2008)
13. Sonntag, M., Karastoyanova, D.: Model-as-you-go: An Approach for an Advanced Infrastructure for Scientific Workflows. Grid Computing 11(3), 553–583 (2013)
14. Sonntag, M., Karastoyanova, D.: Ad hoc Iteration and Re-execution of Activities in Workflows. Int. J. On Advances in Software 5(1 & 2), 91–109 (2012)
15. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data Knowl. Eng. 66(3), 438–466 (2008)
16. Weiß, A., Karastoyanova, D.: A Life Cycle for Coupled Multi-Scale, Multi-Field Experiments Realized through Choreographies. In: Proceedings of EDOC'14. pp. 234–241. IEEE (2014)
17. Weiß, A., Karastoyanova, D.: Enabling coupled multi-scale, multi-field experiments through choreographies of data-driven scientific simulations. Computing pp. 1–29 (2014)
18. Wetzstein, B., Karastoyanova, D., Kopp, O., Leymann, F., Zwink, D.: Cross-Organizational Process Monitoring based on Service Choreographies. In: Proceedings of SAC'10. pp. 2485–2490. ACM (2010)