The Subprocess Spectrum

Oliver Kopp, Hanna Eberle, Frank Leymann, Tobias Unger

Institute of Architecture of Application Systems, University of Stuttgart, Germany
Universitätsstraße 38, 70569 Stuttgart, Germany
{kopp,eberle,leymann,unger}@iaas.uni-stuttgart.de

Abstract: Using hierarchical structurings in process design is a frequent process modeling technique. Subprocesses are a common way to enable hierarchical structuring. Current approaches have a tight view on the syntactical restrictions of subprocesses and do not investigate different autonomy properties in detail. This paper fills this gap and broadens the current subprocess definition to a spectrum of possibilities of subprocess notations. Thereby, three classifications are introduced: subprocess autonomy, interaction between parent process and subprocess, and execution of subprocesses.

1 Introduction

"Sub" and "process" together form the term "subprocesses". "Sub" implies that something is below something else and that something is on the bottom, which is closely related to hierarchies. For instance, hierarchies are introduced within organizations in order to form an ordering between a set of entities. Hierarchies are relations of power, i. e., the entity superordinate can control the subordinate entity concerning certain properties. For example, within a company a manager has managerial authority over his staff. In the context of processes, subprocesses also introduce a hierarchy between a set of processes. The aim of this work is to present the properties of the relation between two processes, where one process is below another process and hence is called "*sub*process".

Subprocesses are introduced in [LR00] as a modeling construct used for modeling reusable business processes. Subprocesses are required to have a single logical entry and a single logical exit. They are not allowed to communicate with the calling process inbetween. Subprocesses also have a strong lifecycle dependency on the caller. This coupling effectively means that the subprocess has to give up his lifecycle autonomy: the subprocess has to be terminated if the calling activity is terminated. If the calling activity is compensated, the entire subprocess has to be compensated.

In this paper, we call those subprocesses "traditional subprocesses" and use the term "subprocesses" for extended traditional subprocesses, which have more defined properties than traditional subprocesses. We present in Sect. 2 taxonomies of these properties to enable a classification of subprocesses. The taxonomies differentiate autonomy properties, the interaction between parent process and subprocess, and the execution of subprocesses. Using the taxonomies, we provide a new subprocesses definition and a comparison to

choreographies in Sect. 3. Related work in the field of subprocesses is presented in Sect. 4. Finally, Sect. 5 provides a conclusion and provides and outlook on future work.

We focus on business processes executed fully on a workflow engine (called "workflow" in [LR00]). Internal activities may be executed by humans, which is usually implemented using human tasks. The coordination between actions of the workflow engine and actions of humans is handled by the task manager. An example of such an action is termination of an activity. The description of these issues is out of scope of this paper.

2 Taxonomies

This section provides three taxonomies for subprocesses: autonomy (Sect. 2.1), interaction (Sect. 2.2) and execution (Sect. 2.3). These taxonomies can be used to classify subprocesses and provide a definition for extended traditional subprocesses as presented in Sect. 3.

2.1 Autonomy

Autonomy properties describing the autonomy of the subprocess can be distinguished into five different subclasses (Fig. 1), each regarding a different aspect of autonomy and its renunciation. In the following, we describe each autonomy class and assign existing approaches to a class. As future research may find new approaches, the presented taxonomy is necessarily not complete.

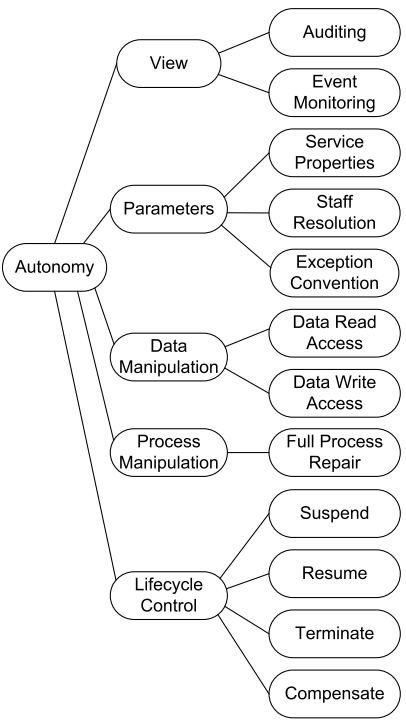


Figure 1: Autonomy Taxonomy

View Providing different views is key for business process execution and business process compliance checking [SLS10]. For instance, a part of a business process may be outsourced [KL06]. Then, the outsourcing company may still have the obligation to show that the outsourced parts follow a compliance rule. Another example are external services that may only be bound to a business process if they offer to track its audit trails from the caller's side. Hence, a service has to state whether it offers access to its *audit* trails and how long these audit trails are kept [LR00]. External *event monitoring* may be needed if cross-organizational process metrics have to be calculated [WKK+10].

Parameters Available parametrization possibilities of subprocesses include changing of service properties, parametrization of staff resolution and promising the raise of certain exceptions. Changing of *service properties* is described in [KLN+06]. There, strategies are described to change the WSDL port type and WSDL operation of a process at both, deployment and runtime. For example, these parameters may be set at the call of the subprocess to set specific services the subprocess has to use. In the case of *staff resolution*, the parent process sets staff resolution parameters [LR00] such as the organizational data base to be used or the group of people to select from. A subprocess may fault because of an erroneous situation. In this case, it does not reply with a result message, but with a fault message. Using an *exception convention*, the subprocess assures that it will rise certain exceptions in the respective situations. This is used to ensure that (i) all known exceptions are raised and (ii) the raised exceptions can be handled by the parent process.

Data Manipulation A subprocess may read variables and correlation sets of the caller and the caller may read variables and correlation sets of the subprocess. This enables parameter passing by reference in contrast to value passing in traditional subprocesses (*data read access*). Data manipulation also includes *data write access*, where data is shared and may be updated by the other party. Note that "data" excludes "parameters": "data" denotes data where the process has direct access to during execution, whereas "parameters" denote data used by the workflow engine to execute the process.

Process Manipulation A process may be adapted to fulfill new requirements or to enable *full process repair* in erroneous situations. In case values of variables have to be changed to repair the process, data write access has also be enabled. An overview on process manipulation and adaption is given in [RRMD09].

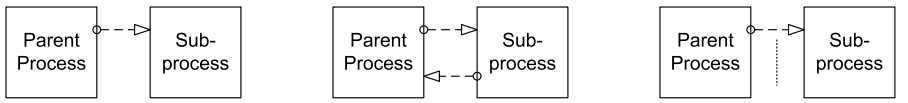
Lifecycle Control In traditional subprocesses there exist lifecycle dependencies between parent process and subprocess. Lifecycle dependency consists of suspending and resuming a subprocess, termination of the subprocess and compensation of a completed subprocess.

The concrete implementation of these properties is not presented in this paper. The autonomy properties may be handled by software or by organizational measures. In the case of manual processes the property data sharing may manifest itself by a company regulation that how folders of a manual parent process can be accessed. For instance, a written form to request the folder may be used.

2.2 Interaction between Parent Process and Subprocess

This section investigates all possible interaction patterns and the possible internal structures of subprocesses and parent process. The outcome of this investigation can be used in future work to define collaboration and coordination protocols between parent and subprocess.

Message Exchange Patterns (MEPs) denote a set of messages and the order in which these messages are sent or expected to be received [NvLL08], also called "interaction". Autonomy properties may influence the MEPs. For instance, a termination of a subprocess also leads to a stop of the MEP as no further messages are exchanged. There are two solutions to this issue: (i) the MEPs model the "happy path" through the process. The exceptional behavior has to be derived from the process. (ii) All possible message exchanges have to be enumerated. In this case, operating guidelines may be used [LMW07]. In this paper, we



(a) send only (b) send and receive (c) send and arbitrary subsequent interactions

Figure 2: Possible Message Exchanges

follow the first approach, as it follows the widely used approach to separate the normal flow from the exception flow.

2.2.1 Parent process / Subprocess Message Exchange Patterns

The process interaction between a parent and a subprocess is restricted such that the subprocess always gets started by receiving a message sent by the parent process. Figure 2 presents the three different kinds how the interaction may continue: send only, send and receive, send, arbitrary interactions and optionally a final reply message.

Figure 2a depicts the *send only* pattern. A subprocess gets spawned by the parent process and no further synchronization between the two processes is taking place. This interaction pattern is also called "kick off and forget", "connected discrete" and "chained services model" [LR00, Hol95]. The pattern used in the case of traditional subprocesses is shown in Fig. 2b: the subprocess first receives a message and finally replies with a message. This pattern is also called "hierarchical" [LR00, Hol95] or "functional de-composition" [HZ07]. Figure 2c presents the general interaction with a subprocess: the parent process sends a message to the subprocess. Then, they communicate arbitrarily with each other.

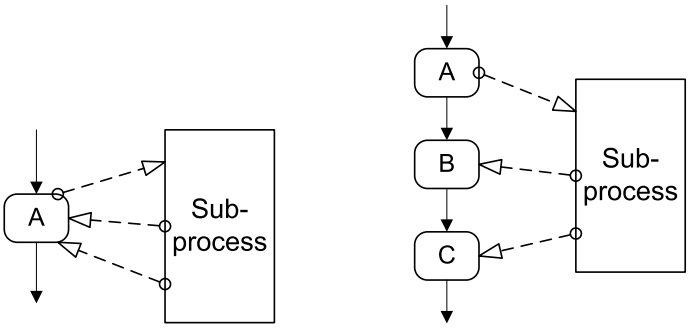
In case there is no final reply message, a coordination protocol has to provide the parent process the information that the subprocess is finished. This enables the calling process to drive the lifecycle autonomy implementation: in case a subprocess is running, it may only be terminated and not be compensated. In case a subprocesses is completed, it may only be compensated and not be terminated.

2.2.2 Parent Process MEP Implementations

The interaction with the subprocess can be implemented in the parent process in two ways: It is possible to model the interaction with the subprocess using a single activity implementing the MEP (Fig. 3a) or using multiple activities to implement the MEP (Fig. 3b). There can be also variants between these two possibilities: Parts of a MEP can be realized using single activities and the remaining parts using activities implementing multiple message exchanges.

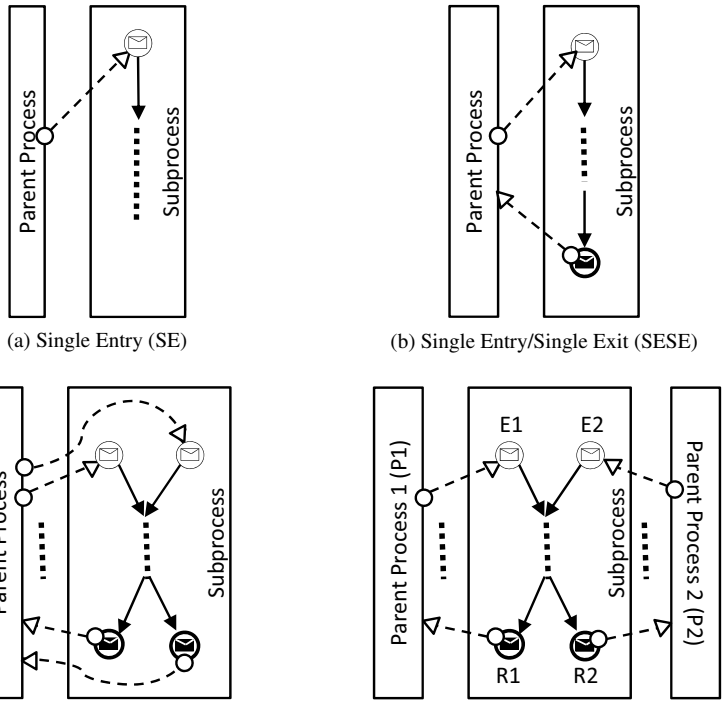
2.2.3 Subprocess Structures

The structure of a traditional subprocess is Single Entry/Single Exit (SESE). Subprocesses, however, may be structured differently as presented in Fig. 4. The simplest structure of



(a) single activity (b) multiple activities

Figure 3: Parent Process Structures to implement MEPs



(a) Single Entry (SE)

(b) Single Entry/Single Exit (SESE)

(c) Multiple Entries/Multiple Exists (MEME), one partner

(d) Multiple Entries/Multiple Exists (MEME), multiple partners

Figure 4: Subprocess Structures: Possible Entry and Exit Patterns

a subprocess contains a single entry and no exit (Fig. 4a). The traditional subprocess interaction pattern SESE is presented in Fig. 4b. A subprocess may also have multiple entries and multiple exists (MEME) as presented in Fig. 4c.

Multiple entries may be mutual exclusive. This transforms the entry to a logical single entry: the parent process may only use one of the multiple entries. After using the entry, the subprocess is instantiated and does not wait for the second message. Similarly, multiple exists may be mutually exclusive which transform them to a single logical exit. In case a subprocess has a single logical entry and a single logical exist, it falls into the category "SESE", since the caller cannot distinguish between the exits.

In case a subprocess has multiple entries, these entries may be used by different calling processes as shown in Fig. 4d. Here, process P1 uses entry E1 and exit R1, process P2 uses entry E2 and exit R2. Activities on the path w1 from P1 to R1 and on the path w2 from P2 to E2 fall into three categories: (i) only on the path w1, (ii) only on the path w2 and (iii) both on the path w1 and w2. Depending on the autonomy, the whole subprocess has to fulfill the autonomy requirements of the calling process or the activities on the execution path of the calling process. This might have influence on the response time of the process to other partners. The concrete relationship of autonomy and service level agreements [UMLS08] has not been investigated yet. A concrete classification and investigation of the interplay between different autonomy degrees is out of scope of this paper.

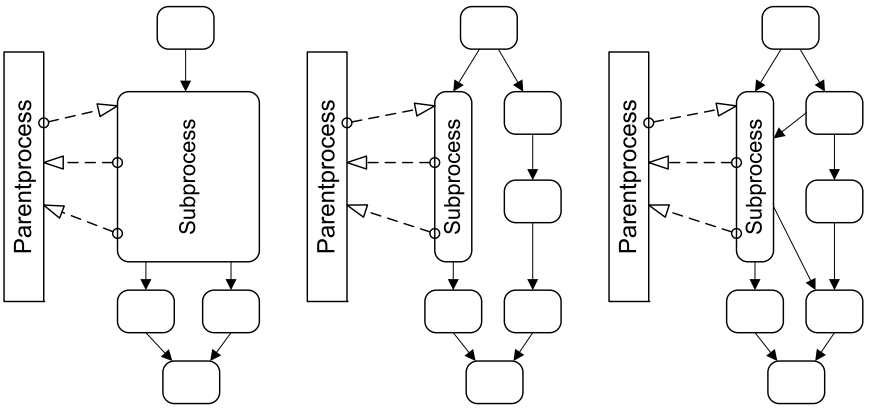
2.2.4 Subprocess Embedding

The subprocess may be a part of a larger process as illustrated in Fig. 5. Figure 5a presents the case where the subprocess is nested in a process without parallel activities. In this case, execution of the subprocess is not influenced by any other activity in the process, as no parallel execution happens outside. Figure 5b presents a case with activities executing in parallel to the subprocess. On the one hand, the subprocess may be aborted due to a fault in parallel activities. Suppose an activity A runs in parallel to a subprocess S. Assume A and S nested in the same parent activity P. Then, a fault in A is propagated to P, which causes S to terminate [CKLW03, BPE07]. This is behavior is independent of existing control links between A and S. On the other hand, a fault in the subprocess may in turn abort parallel activities. Figure 5c presents the case where control links cross the boundary of the subprocess. Thus, the autonomy of the subprocess also influences the execution behavior of the other activities. For instance, if the subprocess is suspended and subsequent control links reach from the subprocess to activities not being in the subprocess, the navigation there has to wait for the subprocess to be resumed.

2.2.5 Recursive Subprocess Calls

Figure 6 illustrates a case, where a subprocess calls another process. This process is then a subprocess to the calling subprocess and a "grand-subprocess" to the parent process. In this situation, the autonomy degree of the sub-subprocess must not be higher than the autonomy degree of the subprocess. For instance, if the subprocess has no lifecycle autonomy it must be the case that the sub-subprocess must have no lifecycle autonomy, too. More subordination for the sub-subprocess is possible, but lifecycle subordination is mandatory.

The subprocess is typically not an instance of the same process model as the calling process.



(a) subprocess nested in a process

(b) subprocess with a parallel process branch

(c) subprocess with cross-boundary links from parallel process branch

Figure 5: Subprocess Structures: Embedded Subprocesses

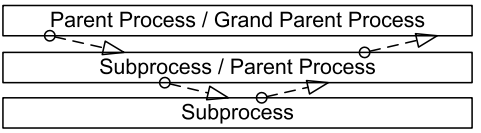


Figure 6: Recursive Subprocess Calls

The word "recursive" refers to the metamodel level: a process of the type "subprocess" calls another process of the type "subprocess".

2.3 Execution of Subprocesses

Subprocesses can be executed on the same workflow management system as the caller or on a different system. The first kind of subprocesses is called *local subprocess*, whereas the second kind is called *remote subprocess* [LR00].

3 Definition of Subprocess

The preceding section provided three classifications. Using these classification, traditional subprocesses and extended traditional subprocesses can be compared as presented in Fig. 7. There, the different execution possibilities of subprocesses have been dropped as traditional subprocesses as well as subprocesses allow both a local and remote execution. The Y-axis presents different autonomy degrees (cf. Sect. 2.1). Here, all different combinations of the autonomy properties are sketched. A traditional subprocess is always fully lifecycle dependent, whereas subprocesses allow a flexible choice of autonomy properties. The

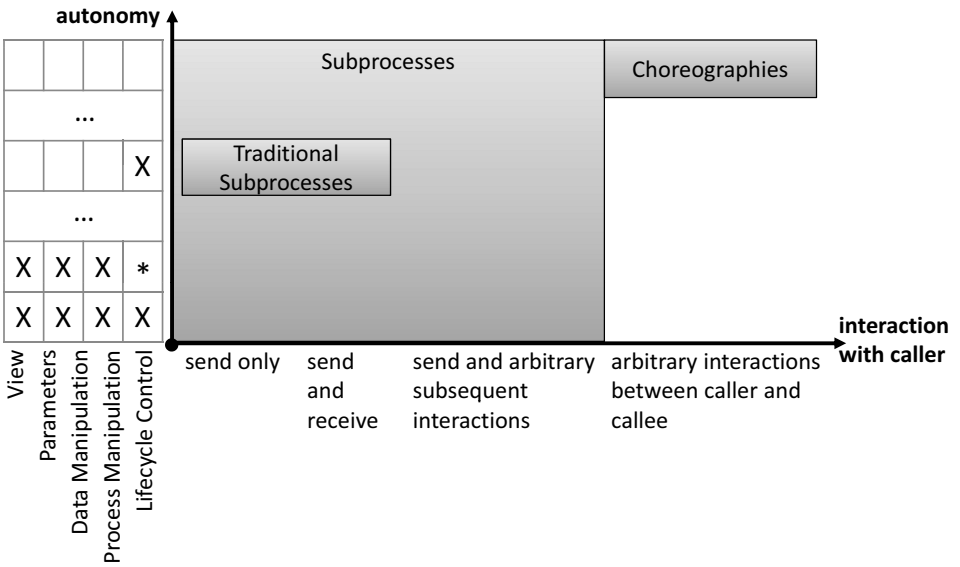


Figure 7: Subprocess Spectrum Overview. "X" denotes all autonomy properties of the respective class. "*" denotes the subset "suspend, resume and terminte" of the class "lifecycle control". "···" denotes that there are more combinations of autonomy properties, which are not explicitly shown in the table.

X-axis presents different interactions (cf. Sect. 2.2). In the case of subprocesses, a parent process always triggers interaction with a subprocess and an interaction is never started by a subprocess. In case arbitrary interactions between the parties are allowed, we are in the field of choreographies. A choreography describes the interaction between multiple processes, focusing on the interactions only or modeling internal behavior, too [DKLW09]. Current choreography languages do not offer to model autonomy degrees. Thus, choreographies are placed in the top of the diagram. Processes described in choreographies may also be executed on the same machine. Thus, choreographies also allow both local and remote execution.

Having regarded the classifications, a subprocess is defined as follows: *A subprocess is a process with a single logical entry and zero or more autonomy properties*. The concrete autonomy properties can be chosen from the autonomy degrees presented in Sect. 2.1.

4 Related Work

Related work on subprocesses discusses modeling aspects and runtime aspects of subprocesses. First, we present approaches mainly dealing with modeling aspects of subprocesses (Sect. 4.1) followed by Sect. 4.2 discussing runtime aspects. The overall distinction to our work is, that all these works discuss these aspects under the assumption of a traditional subprocess definition.

4.1 Model

The work of Vanhatalo et al. [VVK08] detects single-entry-single-exist regions (SESE regions) in workflow graphs. They define subprocesses as one SESE region and plan to use their work to identify common SESE regions across multiple processes to create a repository of reusable subprocesses.

YAWL [vdAtH05] supports modeling tasks as composite tasks. The execution semantic of a subprocess is that it gets started as soon as the composite task is started. The subprocess is canceled in case the task is canceled. As YAWL does not support compensation, the subprocess cannot be compensated.

Kiepuszewski et al. [KtHvdA03] focus on the control flow in workflows and defines subnets. They do not regard interactions between partners and lifecycle subordination.

Fragments are parts of business processes, which are stored in a repository and used at business process modeling [LD99, KWK05, ML09, EUL09] or executed as a part of a complete process [KL06, BD99, EUL09]. Whereas subprocesses have a defined entry and a defined exit, fragments are arbitrary parts of business processes. Lindert et al. [LD99] use autonomy in the sense that "the organizational unit can autonomously describe the fragment and enact the fragment". Lifecycle, data sharing or other autonomy degrees are not regarded. Up to now, fragments for business process modeling are not annotated with autonomy properties, which hampers reuse of SESE fragments as subprocesses. Fragments used for execution have predefined properties stating that they are part of a global model and thus implicitly define their autonomy.

The workflow patterns [vdAtHKB03] use "subprocess/activities" in the context of parallel branches of a workflow. In pattern 12 "Multiple Instances Without Synchronization", the authors describe the possibility to "kick-off" a new workflow instance without additional synchronization and control by the caller, which corresponds to "send only" in Fig. 2.

Both Service Interaction Patterns [BDtH05] and message exchange patterns [NvLL08] present how multiple services may interact which each other. Both works do not treat autonomy.

Reijers et al. [RMD10] investigate whether using subprocess during business processes design helps to foster understanding of complex processes. The authors provided evidence that this is true. They regard SESE subprocesses, but do not investigate the influence of different autonomy degrees.

Regarding process autonomy, von Riegen et al. [vRZ07] discuss the supervision of partner processes. The work lists different levels of enforcement, formats and communication patterns. Offering such properties introduces subordination to a participant. The properties itself are part of the view category in the autonomy taxonomy presented in Sect. 2.1. Leymann and Roller [LR00] outline a spectrum "from the subprocess being absolutely autonomous to the subprocess being totally controlled by the parent process", but do not provide a concrete spectrum. The WfMC reference model [Hol95] does not provide information about subprocess autonomy.

Veijalainen [Vei07] discusses autonomy in mobile P2P environments. There, autonomy is defined as the impossibly to control the behavior of an entity in the context of interactions.

In our work, that kind of autonomy is reflected in the control flow definition of a subprocess. Starting from that definition, an operating guideline can be generated [LMW07]. Using this operating guideline, the autonomy with respect to communication can be derived. Our model of autonomy (presented in Sect. 2.1) goes beyond the pure interactions and lists properties such as life-cycle dependency.

BPEL4WS 1.1 [BPE03], the predecessor of WS-BPEL 2.0 [BPE07], allowed a compensation handler to be specified at `process` level. This property has been removed in WS-BPEL 2.0, as the committee decided to remove any dependency on specific coordination protocols [Ark04]. These coordination protocols are defined in the BPEL subprocess extension (BPEL-SPE [IBM05]). There, the subprocess capability is added to BPEL: a BPEL subprocess is a process with a single start activity and a logical single reply activity with a compensation handler enabling compensation of the whole process instance.

4.2 Runtime

The workflow data patterns [RtHEvdA05] present different possible variants of data sharing between a process and the called subprocess. The approaches range from no sharing to full access of the data of sibling subprocess instances and the parents's process data. The subprocess definition used by the work states, that a subprocess may have multiple entries and multiple exists.

The approach presented by Hagen et al. [HA99] uses events instead of a shared database to exchange data between a parent process and a subprocess. The events are also used to propagate exceptions across processes. These events trigger compensation of finished activities. Thus, the subprocess has full autonomy besides exception handling.

There are several late binding approaches, selecting and binding subprocesses at runtime. Marconi [MPS+09] proposes an approach refining activities of a process at runtime with a subprocess. Worklets [AtHEvdA06] use subprocesses as implementation of tasks, where the selection strategy depends on context data.

In case a subprocess is a standalone process, it is instantiated by a single or multiple messages. In this paper, we did not focus on different instantiation semantics. A detailed discussion about process and subprocess instantiation is available in [DM09].

The work of Grefen et al. [GLDA06] deals with business process outsourcing and proposes an interface "CTRL" to control the behavior of an outsourced process. They list pause process, resume process, abort process and compensate process as example. These properties are part of our lifecycle control autonomy. The infrastructure presented in [GLDA06] can be used to realize our different degrees of autonomy.

Regarding the interplay between autonomy degrees, Kopp et al. [KML09] present that current coordination protocols are not enough to handle the case of an activity being on a path of two callers (MEME with multiple partners, case (iii) in Sect. 2.2.3) and having the lifecycle control dependency "Terminate" and "Compensate".

WebSphere Process Server 7.0 [IBM10] enables the execution of subprocesses, whose lifecycle is bound to the caller.

5 Conclusion and Outlook

In this paper we presented a spectrum of subprocesses fathoming the area between traditional subprocesses and choreographies. We presented three classifications for subprocesses: autonomy, interaction and execution. These classifications can be used to classify existing works as well as classifying future work on process modeling. We excluded concrete implementations and concrete coordination protocols in this work. Our future work is to provide a survey on existing implementations and possible coordination protocols.

The main focus of our future work is on the interplay of different autonomy properties and the autonomy modeling in choreography models. In case a subprocess itself is a part of a process, the most interesting point is the interplay between compensation requests from callers and compensation of the subprocess by the process itself. Additionally, the relation between process autonomy and service level agreements has not yet been investigated. The presented autonomy classification provides a basis to start investigation of relations between service level agreements and process autonomy.

Acknowledgments

This work is partially funded by the ALLOW project[1] and partially funded by the Tools4BPEL project. ALLOW is part of the EU 7th Framework Programme (contract no. FP7-213339). Tools4BPEL is funded by German Federal Ministry of Education and Research (project number 01ISE08B).

References

[Ark04] Assaf Arkin. Issue 53: Consistent enablement of compensation handlers, 2004. `http://www.choreology.com/external/WS_BPEL_issues_list.html` , version of 2006-08-28.

[AtHEvdA06] Michael Adams, Arthur H. M. ter Hofstede, David Edmond, and Wil M. P. van der Aalst. Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In *On the Move to Meaningful Internet Systems*. Springer, 2006.

[BD99] Thomas Bauer and Peter Dadam. Efficient Distributed Control of Enterprise-Wide and Cross-Enterprise Workflows. In *Enterprise-wide and Cross-enterprise Workflow Management*, volume 24 of *CEUR Workshop Proceedings*, 1999.

[BDtH05] Alistair Barros, Marlon Dumas, and Arthur ter Hofstede. Service Interaction Patterns. In *3rd International Conference on Business Process Management*. Springer, 2005.

[BPE03] BEA, IBM, Microsoft, SAP, Siebel Systems. *Business Process Execution Language for Web Services Version 1.1*, 2003.

[BPE07] OASIS. *Web Services Business Process Execution Language Version 2.0 – OASIS Standard*, 2007.

[1] `http://www.allow-project.eu/`

[CKLW03] Francisco Curbera, Rania Khalaf, Frank Leymann, and Sanjiva Weerawarana. Exception Handling in the BPEL4WS Language. In *BPM 2003: Business Process Management, International Conference*. Springer, 2003.

[DKLW09] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. Interacting services: From specification to execution. *Data & Knowledge Engineering*, 68(10):946–972, 2009.

[DM09] Gero Decker and Jan Mendling. Process Instantiation. *Data & Knowledge Engineering*, 68:777–792, 2009.

[EUL09] Hanna Eberle, Tobias Unger, and Frank Leymann. Process Fragments. In *CoopIS 2009 (OTM 2009)*. Springer, 2009.

[GLDA06] Paul W. P. J. Grefen, Heiko Ludwig, Asit Dan, and Samuil Angelov. An analysis of web services support for dynamic business process outsourcing. *Information & Software Technology*, 48(11):1115–1134, 2006.

[HA99] Claus Hagen and Gustavo Alonso. Beyond the Black Box: Event-based Inter-Process Communication in Process Support Systems. In *International Conference on Distributed Computing Systems*. IEEE Computer Society, 1999.

[Hol95] David Hollingsworth. *The Workflow Reference Model*. Workflow Management Coalition, Jan 95. Document Number TC00-1003.

[HZ07] C. Hentrich and U. Zdun. Service Integration Patterns for Invoking Services from Business Processes. In *Proceedings of 12th European Conference on Pattern Languages of Programs (EuroPLoP 2007)*, 2007.

[IBM05] IBM and SAP. *WS-BPEL Extension for Sub-processes – BPEL-SPE*, 2005.

[IBM10] IBM. WebSphere Process Server 7.0, 2010. `http://www.ibm.com/software/integration/wps/`, version of 2010-07-28.

[KL06] Rania Khalaf and Frank Leymann. Role-based Decomposition of Business Processes using BPEL. In *International Conference on Web Services*. IEEE Computer Society, 2006.

[KLN+06] Dimka Karastoyanova, Frank Leymann, Jörg Nitzsche, Branimir Wetzstein, and Daniel Wutke. Parameterized BPEL Processes: Concepts and Implementation. In *Business Process Management*. Springer, 2006.

[KML09] Oliver Kopp, Ralph Mietzner, and Frank Leymann. The Influence of an External Transaction on a BPEL Scope. In *CoopIS 2009 (OTM 2009)*. Springer, 2009.

[KtHvdA03] B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of control flow in workflows. *Acta Informatica*, 39(3):143–209, March 2003.

[KWK05] Kwang-Hoon Kim, Jae-Kang Won, and Chang-Min Kim. A Fragment-Driven Process Modeling Methodology. In *Computational Science and Its Applications - ICCSA*. Springer, 2005.

[LD99] Frank Lindert and Wolfgang Deiters. Modelling Inter-Organizational Processes With Process Model Fragments. In *Enterprise-wide and Crossenterprise Workflow Management: Concepts, Systems, Applications*, CEUR Workshop Proceedings, 1999.

[LMW07] Niels Lohmann, Peter Massuthe, and Karsten Wolf. Operating Guidelines for Finite-State Services. In *Petri Nets and Other Models of Concurrency – ICATPN*. Springer, 2007.

[LR00] Frank Leymann and Dieter Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000.

[ML09] Zhilei Ma and Frank Leymann. BPEL Fragments for Modularized Reuse in Modeling BPEL Processes. In *International Conference on Networking and Services (ICNS)*. IEEE, 2009.

[MPS+09] Annapaola Marconi, Marco Pistore, Adina Sirbu, Frank Leymann, Hanna Eberle, and Tobias Unger. Enabling Adaptation of Pervasive Flows: Built-in Contextual Adaptation. In *International Joint Conference on Service Oriented Computing (ICSOC 2009)*. Springer, 2009.

[NvLL08] Jörg Nitsche, Tammo van Lessen, and Frank Leymann. WSDL 2.0 Message Exchange Patterns: Limitations and Opportunities. In *3rd International Conference on Internet and Web Applications and Services (ICIW)*. IEEE Computer Society, 2008.

[RMD10] Hajo A. Reijers, Jan Mendling, and Remco M. Dijkman. On the Usefulness of Subprocesses in Business Process Models. Technical report, BPM Center Reports, 2010. BPM-10-03.

[RRMD09] Manfred Reichert, Stefanie Rinderle-Ma, and Peter Dadam. Flexibility in Process-aware Information Systems. *ToPNoC*, 2:115–135, 2009.

[RtHEvdA05] Nick Russell, Arthur H.M. ter Hofstede, David Edmond, and Wil M.P. van der Aalst. Workflow Data Patterns: Identification, Representation and Tool Support. In *24th International Conference on Conceptual Modeling (ER 2005)*. Springer, 2005.

[SLS10] David Schumm, Frank Leymann, and Alexander Streule. Process Views to Support Compliance Management in Business Processes. In *EC-Web*. Springer, 2010.

[UMLS08] Tobias Unger, Stephanie Mauchart, Frank Leymann, and Thorsten Scheibler. Aggregation of Service Level Agreements in the Context of Business Processes. In *Enterprise Distributed Object Conference (EDOC)*. IEEE, 2008.

[vdAtH05] WMP van der Aalst and AHM ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[vdAtHKB03] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[Vei07] Jari Veijalainen. Autonomy, Heterogeneity and Trust in Mobile P2P environments. *International Journal of Security and Its Applications*, 1(1):57–71, July 2007.

[vRZ07] Michael von Riegen and Sonja Zaplata. Supervising Remote Task Execution in Collaborative Workflow Environments. In *Kommunikation in Verteilten Systemen – 15. ITG/GI-Fachtagung (KiVS 2007)*, 2007.

[VVK08] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The Refined Process Structure Tree. In *BPM'08: Business Process Management, 6th International Conference, BPM 2008*. Springer, 2008.

[WKK+10] Branimir Wetzstein, Dimka Karastoyanova, Oliver Kopp, Frank Leymann, and Daniel Zwink. Cross-Organizational Process Monitoring based on Service Choreographies. In *25th Annual ACM Symposium on Applied Computing (SAC 2010)*. ACM, 2010.