Coordinating Large Distributed Process Structures

Sebastian Steinau, Kevin Andrews, and Manfred Reichert

Institute of Databases and Information Systems, Ulm University, Germany
{sebastian.steinau,kevin.andrews,manfred.reichert}@uni-ulm.de

Abstract. Representing a business process as interacting small processes has become feasible with data-centric business process management paradigms. These small processes have relations and, thereby, form a relational process structure. The interactions of processes within this relational process structure must be coordinated to arrive at a meaningful overall business goal. However, relational process structures may become arbitrarily large and, with cloud technology, they may additionally be distributed over multiple nodes. Coordination processes have been proposed to coordinate relational process structures, where processes have one-to-many and many-to-many relations at run-time. This paper shows how multiple coordination processes can be used in a decentralized fashion to coordinate large, distributed process structures. The main challenge is to effectively realize the coordination responsibility of each coordination process. Key components of the solution are the subsidiary principle and the hierarchy of the relational process structure. Moreover, from these key components and the technical properties of coordination processes, an implementation based on microservices was developed, which allows fast and concurrent enactment of multiple, decentralized coordination processes in large, distributed process structures.

Keywords: Process interactions, relational process structure, coordination process, distributed process execution, BPM in the cloud

1 Introduction

Several approaches to business process management advocate to represent business processes as collections of interacting, interdependent small processes. Examples include the artifact-centric and object-aware approaches [4,5,9], where the collaboration of artifact or object lifecycle processes forms an entire business process. Principal challenges of these approaches are to determine which processes exist and how they relate to other processes, as well as the coordination of this structure of interdependent processes. Recently, the *relational process structure* [11] and *coordination processes* [10] have been proposed to tackle these challenges. A relational process structure captures processes and their relations in a hierarchical construct, which is used by a coordination process to specify and enforce *coordination constraints*. This allows the interactions of different processes to be guided towards a meaningful overall business process.

However, fundamental challenges still remain. A relational process structure may become arbitrarily large, i.e., it may comprise hundreds or more types of processes. At run-time, hundreds or thousands of instances of these different process types are created, as well as their interrelations, compounding the problem. Furthermore, interacting small processes are particularly suited to be employed in a distributed instead of a monolithic system. In consequence, some processes may be located on one node of the distributed system, whereas other processes are located on different nodes. Existing approaches to coordinate such large process structures propose employing a single *central coordinator* (e.g., a master artifact [13]). The term *coordinator* is hereby intended as an umbrella term for any kind of process coordination model, independent of paradigm or exact specification, e.g., choreography, coordination process, or Proclet [14]. A single, central coordinator for a vast process structure is however unsuitable. The coordinator has to incorporate all coordination requirements for all processes in its model. As a result, a central coordinator model can become overloaded, inflexible, costly to maintain, and difficult to understand. As another drawback, all distributed processes must communicate with the central coordinator, creating a huge communication overhead and, more importantly, a single point of failure. Additionally, as process structures become larger, several independent substructures may emerge, where each requires an individual coordination. For example, in the automotive industry, cars may be highly customized, requiring varying constraints on the production, assembly, and testing of the parts for each car.

As process structures may become very large and different substructures may be distributed across the nodes of a server cluster, it is beneficial to distribute and split up the coordination of processes as well. While a coordination process can serve as a central coordinator, the concept is flexible so that *multiple coordination processes* may be used to coordinate a relational process structure. Thereby, several coordination processes collaborate to achieve an overall coordination of the entire process structure. However, the challenge of *coordination responsibility* must be solved, i.e., the question which coordinator is responsible for which processes. Coordination processes are uniquely suited for a decentralized application due to leveraging the hierarchical nature of the relational process structure. This allows implementing the subsidiary principle, where a coordination process only coordinates a subset of processes, defining its coordination responsibility. The result are more flexible and smaller coordination models, a clear coordination responsibility of each coordination model, and a superior maintainability. This paper contributes the decentralized and distributed application of coordination processes and modeling guidelines to effectively model coordination processes in large, distributed relational process structures.

The remainder of the paper is organized as follows. The challenges and benefits of decentralized and distributed process coordination are elaborated in Section 2. Section 3 introduces background information on the relational process structure and the coordination processes. Section 4 presents the key concepts of effectively using coordination processes in a large and distributed relational process structure. Furthermore, an implementation of decentralized coordination

processes is presented, based on microservices. Section 5 discusses related work before Section 6 concludes the paper with a summary and an outlook.

2 Challenges and Benefits

The coordination of a multitude of different, interdependent processes is a complicated and challenging endeavor. Processes and their relations have to be identified and, based on these connections, suitable *coordination constraints* have to be specified and enforced. The different processes and their relations are summarized under the term *process structure*. A coordination constraint denotes a dependency that exists between two or more processes [10]. Generally, approaches for coordinating process structures involving multiple process types advocate the use of a single entity with the purpose of coordinating all involved processes. This entity is denoted as a *central coordinator*.

Central coordinators of any kind (e.g., a master artifact) are capable of properly coordinating different processes. Their main disadvantage is poor scalability in regard to the process structure. As the number of processes in a process structure grows, central coordinators must accommodate these additional processes in their coordination description. Moreover, additional coordination constraints must be incorporated into the coordination descriptions as well. This generally leads to the central coordinator model becoming large and possibly overloaded. With increasing complexity, flexibility suffers, the central coordinator model becomes more difficult to change, and the understandability of the model is impaired as well. Furthermore, performance of the central coordinator may degrade due to the large number of processes and the resulting communication overhead. As a consequence, the central coordinator might become a bottleneck for the overall performance of the business process structure.

From a functional perspective, relying on one central coordinator for coordinating everything is neither the intuitive nor the most effective way of providing process coordination for large process structures. Consider the following example of a recruitment business process.

Example 1. (Recruitment Business Process)
In the context of recruitment, applicants may apply for job offers. The overall process goal for a company is to determine who of the many applicants is best suited for the job. Applicants must write their application for a specific job offer and send it to the company. The company employees then evaluate each application by performing reviews. To reject an application or proceed with the application, a sufficient number of reviews need to be performed, e.g., the majority of reviews determines whether or not an application is rejected. If the majority of reviews are in favor of the application, the applicant is invited for one or more interviews, after which she may be hired or ultimately rejected. In the meantime, more applications may have been sent in, for which additional reviews are required, i.e., the evaluation of different applications may be handled concurrently, as well as the conduction of interviews.

Various interdependent process types can be identified in Example 1: *Job Offer*, *Application*, *Review*, and *Interview*. Each *Job Offer* is largely independent of other *Job Offers*, having its own set of applications and reviews. Consequently, a single central coordinator is tasked with coordinating each *Job Offer* independently from others. The central coordinator must recognize and keep track of different executions states of processes, decision results made during the execution as well as enforcing the appropriate coordination constraints for the *Job Offers* and their connected processes, e.g., *Applications*. This constitutes an enormous complexity for the model of the central coordinator, especially when the run-time is concerned. Moreover, the central coordinator acts as a single point of failure, as problems that might occur with any *Job Offer* may affect all other *Job Offers* as well.

As different *Job Offers* are conceptually independent from each other, a sensible solution would be to arrange that each *Job Offer* is coordinated individually with its connected other processes such as *Applications* or *Reviews*. This means that there is one model of a coordinator that is instantiated multiple times at run-time, once for each *Job Offer*. This is denoted as *stage-1 decentralized coordination.* This shift reduces model complexity, as the logic for distinguishing different *Job Offers* may be omitted due to the coordination happening on a *per-Job Offer*-basis, which in turn benefits understandability and maintainability. The additional complexity of having to instantiate a model multiple times may generally be neglected, as this is one of the core ideas of a process-oriented system. Another advantage is that this also eliminates the single point of failure. If the coordination of one *Job Offer* fails for some reason, other *Job Offers* should remain unaffected. Stage-1 decentralized coordination is inherently supported in coordination processes (cf. [10]).

The distribution of coordinators has plenty of advantages while at the same time only small costs incur. Adding more decentralized coordinators may still yield more benefits.

Example 2. (Unsolicited Application)

Consider the recruitment scenario of an "unsolicited application", i.e. an applicant sends in an *Application* without a prior *Job Offer* from the company. In case the unsolicited *Application* is accepted, a specific *Job Offer* will be created for the application.

As the coordinator that coordinates *Applications* with *Reviews* and *Interviews* is tied to a *Job Offer*, the unsolicited *Application* cannot be processed correctly without a link to a *Job Offer*. Thus, it is reasonable to add another coordinator and transfer responsibilities to it from the *Job Offer* coordinator: The new coordinator coordinates *Applications* with *Interviews* and *Reviews*, and is tied to the respective *Application*. The existing *Job Offer* coordinator is subsequently only responsible for coordinating the *Job Offer* with its related *Applications*. As a result, an unsolicited *Application* may be handled correctly in addition to the usual recruitment procedure. This further reduces the complexity of the individual coordinator models.

Employing multiple coordinators, also denoted as *stage-2 decentralized coordination*, is also advantageous in a distributed environment. Processes may run on different *nodes* in a distributed *cluster*, e.g., servers of different departments of the same company. The nodes and their communication paths are referred to as the *layout* of the cluster. As basic premise, communication within a node is performant and cheap, whereas communication between nodes is more costly. While the primary goal is the proper coordination of all involved processes, a secondary goal is to minimize communication between nodes due to its associated cost. A single central coordinator, running on one node, is forced to communicate with processes on other nodes. By distributing coordinators among nodes, e.g., one coordinator for each node, communication between nodes can be minimized, resulting in more efficient and performant communication.

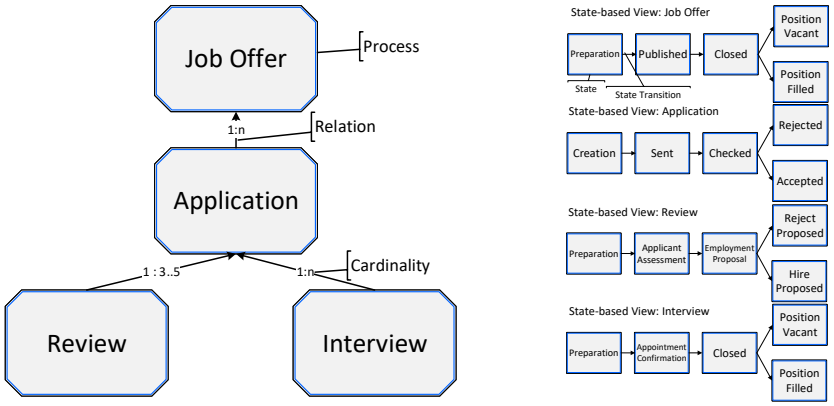
To realize the benefits from the use of decentralized coordinators in process structures, several challenges must be addressed. First, it must be determined how many coordinators are necessary for a given process structure, taking the layout of a potential cluster into account. Second, the processes that require coordination must be assigned to a suitable coordinator, i.e., the *responsibility* of the coordinator must be defined. The responsibility includes that redundancies in the coordination must be avoided. Processes should be assigned, if possible, only to one coordinator, i.e., the overlap between coordinators should be minimal. Otherwise, superfluous work would be performed, or communication costs cannot be reduced compared to a single coordinator. Dividing the responsibility among several coordinators is the primary challenge of decentralized coordinators.

Coordination processes have been designed with a decentralized application in large process structures in mind, and can therefore provide a solution to enable the discussed benefits. This paper contributes new applications of coordination processes and elicits a modeling guideline to effectively utilize the potential of coordination processes.

3 Relational Process Structures and Coordination Processes

For the purposes of this paper, a process is represented in an abstract, simplified manner, which is called a *state-based view* [12]. In a state-based view, each process model is partitioned into different states that are relevant for process coordination. This allows accommodating processes modeled in different paradigms. e.g., artifact-centric or activity-centric processes. The current execution status of a process is determined by the *active state* of the state-based view. Furthermore, process types are design-time entities, from which process instances can be created at run-time. Figure 1b shows the state-based views of the processes from Example 1.

The basis for using coordination processes is the *relational process structure.* It captures all process types relevant for the specific business process [11]. Figure 1a shows the design-time relational process structure of the Recruitment Business Process (cf. Example 1). A relational process structure not only comprises



(a) Relational Process Structure at design-time

(b) State-based views of the processes in the Relational Process Structure

Fig. 1: Relational Process Structure and State-based Views

the different process types, but also includes *relations* between the process types, forming a directed acyclic graph. A relation indicates that the corresponding process types have one or more dependencies between them. A dependency may also exists transitively over a path of relations between two process types. Relations further have cardinalities, restricting how many process instances of one type at run-time may be related to an instance of another process type. Of course, this implies that processes are in one-to-many or many-to-many relationships. In order to enforce these cardinalities at run-time, the relational process structure tracks every created process instance of each process type and monitors each relation that is established between two instances. Thereby, full transparency over process instances and their relations is achieved (cf. Figure 2), allowing a coordination approach to effectively specify constraints on process interactions at design-time and enforce them on all process instances at run-time.

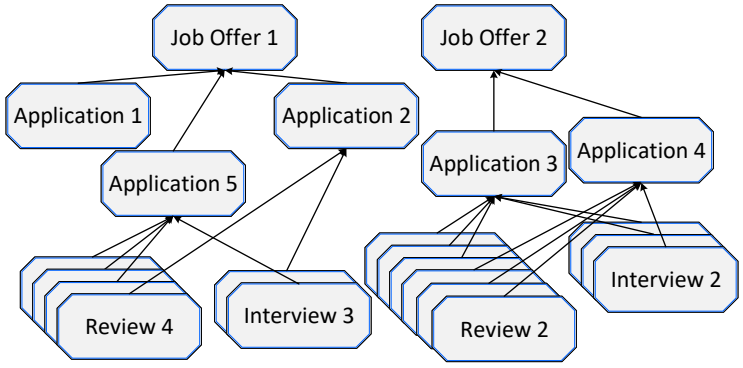


Fig. 2: Run-time Relational Process Structure, tracking Every Process Instance and Relation (simplified view)

Coordination processes [10] constitute an approach for managing process interactions based on the features of the relational process structure. Both coordination processes and relational process structure have their origins in the object-aware process management approach [5]. A coordination process specifies coordination constraints between process types in terms of *semantic relationships*. Semantic relationships are basic interaction patterns of processes in a one-to-many or many-to-many relationship [12]. In a coordination process, processes are represented by *coordination steps*. A semantic relationship, and consequently, a coordination constraint between two process types, is created by establishing a *coordination transition* from one coordination step to another. Figure 3 shows a coordination process that coordinates the recruitment business process (cf. Example 1).

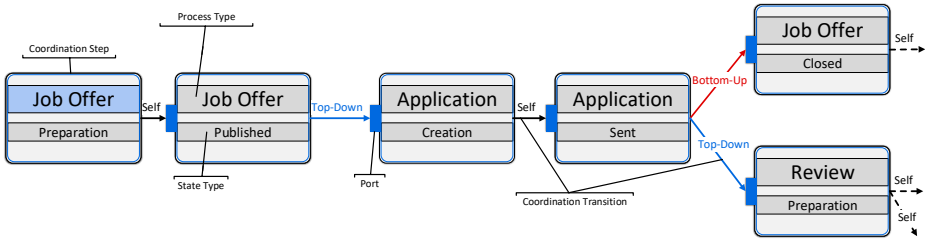


Fig. 3: Coordination Process for the Recruitment Business Process, Part 1

Coordination steps specify a process type and a state of the respective process type. Each incoming semantic relationship of a coordination step represents a condition that must be fulfilled before the respective process is allowed to activate the specified state. Knowing the relations of processes from the relational process structure, fine-grained coordination of the processes becomes possible.

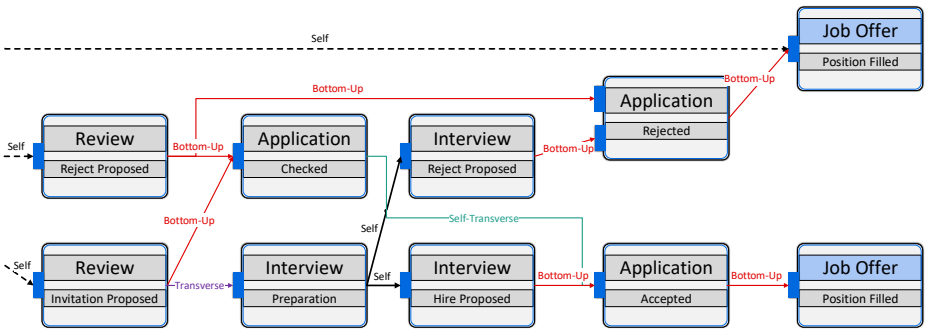


Fig. 4: Coordination Process for the Recruitment Business Process, Part 2

Coordination processes conceptually require a *coordinating process type* to function. In the example from Figure 3, the respective coordination process is attached to a *Job Offer*. This means that each instance of *Job Offer* comes with its own coordination process that coordinates the *Job Offer* with its corresponding *Applications*, *Reviews*, and *Interviews*. Coordination processes already rep-

resent decentralized coordinators, as they are instantiated together with the coordinating process type. A central coordinator can be realized by instantiating a coordinating process type only once, with one coordinating process type per process structure. In the following, it is shown how further decentralization can be achieved with coordination processes, i.e, realizing stage-2 decentralized coordination.

4 Decentralized Coordination Processes

When coordinators are decentralized, one of the primary challenges concerns *responsibility*, i.e., deciding which coordinator shall be responsible for which processes. In particular, coordinators may share responsibility for several processes, i.e., they enforce the same or different coordination constraints on the same processes. Consequently, it is crucial that coordinators do not model contradicting constraints, e.g., a combination of constraints states exactly the opposite of another constraint. With decentralized coordinators, this challenge gains importance as coordinators are modeled individually, i.e., contradictions may not be spotted easily. Consequently, the relational process structure offers a way to address this challenge, i.e., avoiding the possibility for contradictions altogether by clearly defining the responsibility of each coordinator. In particular, responsibilities must overlap as little as possible. Fundamental for the solution, the relations in a relational process structure are directed, which means that processes can be arranged hierarchically. This hierarchy is an integral part of how semantic relationships work, the cornerstone of the coordination process concept. Additionally, the hierarchy of a relational process structure offers advantages when using multiple coordination processes to coordinate a relational process structure.

4.1 Coordination Process Scope

For clearly defining responsibilities, the concept of *scope* of a coordination process is essential. A coordination process is attached to a coordinating process type, and its scope determines which other processes the coordination process is allowed to coordinate, i.e., its *responsibility*. The coordinating process can be easily identified from a coordination process model. By convention, the start and end steps of a coordination process must refer to the coordinating process type [10]. The hierarchy of the relational process structure provides an easy and intuitive solution for defining the scope. The scope of a coordination process is defined as all *lower-level process types* of the coordinating process type. Lower-level processes are all process types that have a (transitive) relation to one particular process type. Regarding the relational process structure in Figure 1a, *Review* and *Interview* are both lower-level processes of *Application*, which in turn are all lower-level processes of *Job Offer*. Attaching a coordination process to the *Job Offer* consequently allows coordinating the entire relational process structure in Figure 1a, i.e., *Reviews*, *Interviews*, *Job Offers* and *Applications*.

The scope of a coordination process achieves that the responsibility of a coordination process is not arbitrary, but clearly defined. This provides a great advantage when modeling decentralized coordination processes, as arbitrary responsibilities of multiple coordinators create unnecessary redundancy as well as potentially contradicting constraints, and decrease the maintainability and understandability of the overall model.

While the scope defines the responsibility of a coordination process, in a relational process structure, the scopes of multiple coordination processes may still overlap. For example, when a coordination process is attached to the top-level process in the hierarchy of the relational process structure, its scope overlaps with the scopes of coordination processes attached to lower-level processes. Consider the unsolicited application from Example 2. *Application* is a lower-level process type of *Job Offer* (cf. Figure 1a). An unsolicited application requires its own coordination process in absence of the coordination process from a *Job Offer*. However, in the end, if an unsolicited application is accepted, a *Job Offer*, together with its associated coordination process, will be created. The *Job Offer* coordination process has the *Application*in scope.

4.2 Subsidiarity

As shown with this example, simply attaching a new coordination process to the *Application* process type creates overlapping scopes with the *Job Offer* coordination process. The required coordination constraints to coordinate *Reviews* and *Interviews* would have to be be replicated in the *Application* coordination process, creating redundancy. In addition to redundancy, contradicting constraints in multiple coordination processes may, in principle, inadvertently be specified. However, the hierarchy of the relational process structure allows additional measures to remove overlap: The application of the *subsidiarity principle*. The Oxford dictionary defines subsidiarity as follows:

> **Subsidiarity** (noun)(in politics) the principle that a central authority should have a subsidiary function, performing only those tasks which cannot be performed at a more local level.[1]

Transferring this principle to both coordination processes and the relational process structure, subsidiarity means that a coordination constraint should be modeled in the lowest coordination process whose scope comprises all process types involved in the constraint. Regarding the unsolicited application, modeling any coordination constraints involving only *Application*, *Review*, and *Interview* in the *Job Offer* coordination process is a clear violation of subsidiarity. By moving corresponding coordination constraints to the *Application* coordination process, subsidiarity is fulfilled. Only the coordination constraints for *Application* and *Job Offer* are kept in the *Job Offer* coordination process. Figures 5 and 6 show *Application* and *Job Offer* coordination processes after the application

[1] https://en.oxforddictionaries.com/definition/subsidiarity

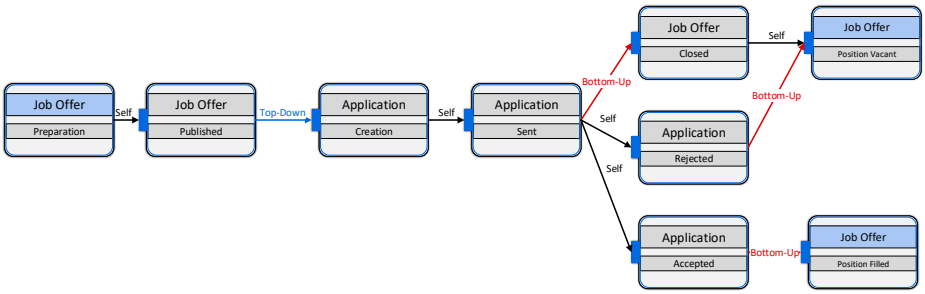


Fig. 5: Job Offer Coordination Process

of the subsidiarity principle. Benefits include the proper support of the unsolicited application variant and the elimination of redundancy between coordination processes. Further, note that the correct coordination of an unsolicited application is only possible with two coordination processes. Moreover, each coordination process model is smaller, simpler and more understandable. Altogether, the subsidiarity principle and scopes enable the proper decentralized coordination of small sections of a relational process structure with coordination processes, which, in turn, collaborate as well to provide coordination for the entire relational process structure.

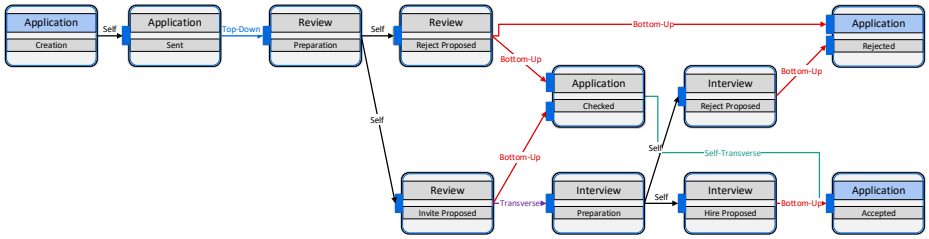


Fig. 6: Application Coordination Process

4.3 Coordination Processes in Distributed Environments

Distributing coordination processes across different hierarchy levels yields significant benefits for the simplicity of the coordination process models. In settings where multiple processes collaborate to achieve a business goal, it is not unreasonable to assume that these processes are not all executed on the same machine. With the advent of cloud computing, distributed applications are gaining even more momentum, as scalability is becoming an important issue [1,2]. For that matter, it is possible to distribute a relational process structure over different nodes in a distributed cluster (e.g., a cloud). Coordination processes and relational process structures originate in object-aware process management and are implemented in the PHILharmonicFlows prototype, which comprises a process execution engine based fully on microservices [1,5]. As such, the issue of distribution of processes across nodes is highly relevant not only for object-aware processes, but in the general sense as well.

Figure 7 shows an example of a feasible distribution of a relational process structure over three nodes. It assigns process types to a specific node, e.g., process A is assigned to Node 1. Each instance of A at run-time is placed onto Node 1 as well. The abstract example is chosen here instead of Example 1 due to its larger size and, therefore, a better illustration of the distribution across nodes.

In regard to process coordination in distributed environments, performance and scalability are the main challenges in addition to a correct coordination. Specifically, communication between processes, and, consequently, communication between nodes, has an important impact on the overall performance of the distributed relational process structure. In general, communication within a node is considered cheap, whereas communication between nodes is costly in terms of time and performance. This holds regardless of any specific metrics, and communication between nodes should therefore be reduced to a minimum.

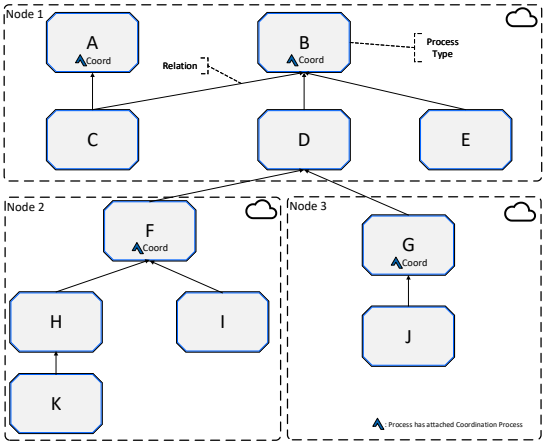


Fig. 7: Relational Process Structure distributed across different Nodes

Obviously, communication between nodes cannot be totally avoided, as processes need to be coordinated across nodes. Coordination processes, however, allow minimizing the communication between nodes significantly. By attaching coordination processes to process types where the scope encompasses the entire node, the communication is kept within a node. Note that further coordination processes within a node are still possible. In Figure 7, process type F has a coordination process that comprises all process types of Node 2. Coordinating the process types F,H, K, and I therefore requires no communication between nodes. Process type F still requires communication with the coordination process of B on Node 1, but the communication amount of Node 2 with the coordination process of B is significantly reduced.

Altogether, coordination processes allow for the decentralized coordination of large process structures. The relational process structure hierarchy, scope, and subsidiarity principle provide clear responsibilities for each coordination process, facilitating modeling and reducing modeling errors. In particular, the

coordination approach no longer contains a single point of failure. By using multiple coordination processes for the same large process structure, the individual coordination process models become smaller and simpler, resulting in greater understandability and maintainability of the models. As shown, these advantages also translate well to a distributed cluster, where a coordination process can be used for each node, significantly reducing communication overhead and, therefore, increasing performance.

4.4 Implementation

Both coordination processes and the relational process structure have been implemented in the PHILharmonicFlows prototype. This prototype is based on the object-aware process management approach and has been developed in the PHILharmonicFlows[2] project at Ulm University. The tool supports a modeling GUI, a run-time GUI where processes and their execution can be visualized, and a server with a REST-enabled interface connected to both GUIs (cf. Figure 8).

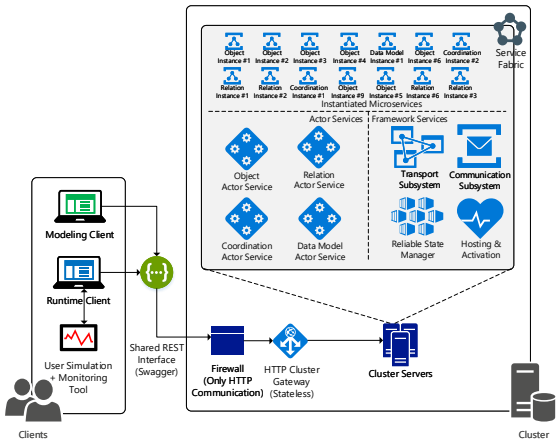


Fig. 8: PHILharmonicFlows prototype architecture

Based on this implementation, an extension of the server and GUIs has been developed to support more than one coordination process in a relational process structure. This development has been conducted based on the concepts presented in this paper. Figure 9 shows the *Application* coordination process from Figure 6 modeled in the PHILharmonicFlows tool.

The run-time engine of the server is also able to handle multiple coordination processes in a relational process structure without any adjustments. This becomes possible since the initial design of the engine considered multiple coordination processes as a future extension. Furthermore, the PHILharmonicFlows server is based on a microservice architecture [1]. Each process instance is realized as one microservice. Microservices may be organized in clusters. Therefore,

[2] For more details on the prototype visit `https://bit.ly/2KYvyT9`

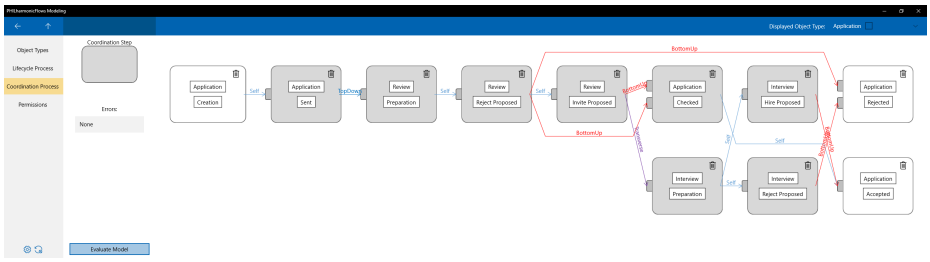


Fig. 9: Modeling the Application Coordination Process with the PHILharmonicFlows Modeling Tool

a relational process structure consists of microservice processes and can be distributed in a cluster, benefiting from decentralized coordination processes as described in Section 4.3.

Process hierarchy and subsidiarity are simple principles, but they create challenges regarding their support *at both design and run-time*. The fact that these principles can be applied with coordination processes in a straightforward fashion to achieve decentralized process coordination in a distributed environment required extensive backing by concepts and implementations. The straightforwardness of using the principles for an intricately complex process coordination solution is the result of foresight as well as careful design and engineering. Both conceptual design and implementation in software of the relational process structure, the coordination process with its semantic relationship, and the microservice engine architecture had to converge to enable the frictionless application of decentralized coordination processes in both modeling and run-time.

5 Related Work

Artifact-centric process management [9,4] operates with artifacts that represent business entities. A business process in the artifact-centric paradigm is constituted by the interactions of the involved artifacts. As such, artifact-centric process management has been dealing with the same challenges as coordination processes in object-aware process management. Traditional activity-centric process management paradigms have investigated the interactions of different processes in one-to-one relationships, where no process structures emerge at run-time. Therefore, activity-centric process coordination is not considered close related work and is therefore not discussed here.

For artifact-centric process management, [3] recognizes the need of process structures, especially regarding many-to-many process relationships. Proclets [14] are chosen to represent an artifact lifecycle as well as their interactions. An approach is developed to represent the interactions of artifacts by means of a new, meaningful artifact acting as a coordinator between two artifact instances. However, open challenges include the specification which artifact instances interact at run-time and defining a coordinator artifact for each two artifact instances. The relational process structure solves the problem of knowing which instances

interact with which other instances. Coordination processes are able to coordinate processes in any relationship, reducing the complexity compared to having a coordinator between any two processes.

Again in the context of artifact-centric process management, [13] proposes declarative artifact choreographies to coordinate the interactions of different artifacts. The declarative choreographies operate on a type-instance schema and involve many-to-many relationships between artifacts, sharing similarities with the coordination process approach. The artifact instances and their relations are captured in a correlation graph, which shares the same responsibility as a relational process structure, but is not hierarchically organized. Based on this correlation graph, declarative rules and constraints may be specified, implementing the declarative choreography. It is however unclear whether a declarative choreography acts only as a central coordinator or is capable of decentralized coordination of a correlation graph.

Finally, [6] introduces the concept of agents and location-aware artifacts. More precisely, an artifact knows which agent it (currently) belongs to. The general idea consists of agents acting upon artifacts and eventually passing artifacts on to other agents. This approach requires an interaction model between agents, i.e., a choreography. The approach synthesizes this interaction model from the lifecycles of all artifacts, which, in essence, represents a central coordinator. The approach is tailored towards artifact-based inter-organizational processes.

The coordination of large process structures not rooted in artifact-centric process management, but with focus on the engineering domain, is considered in [7,8]. The COREPRO approach explicitly considers process relations with one-to-many cardinality, thereby exhibiting the concept of a process structure. A Lifecycle Coordination Model acts as a central coordinator of a process structure. Decentralized lifecycle coordination models are not considered in this approach.

Finally, for an overview and a discussion of coordination approaches in general, that do not necessarily act on process structures, please refer to [10].

6 Summary and Outlook

With coordination processes, the conceptual, technical and methodological capabilities exist to successfully implement decentralized process coordination in large process structures. The concepts of scope, hierarchy of the relational process structure, and the principle of subsidiarity make the complexity of it all manageable and, therefore, the whole approach feasible. On the benefits side, large-scale coordination of large process structures becomes feasible, while at the same time, the complexity and size of individual coordination process models is reduced compared to a central coordinator. As has been shown, this also applies to distributed relational process structures. However, in a different sense multiple coordination processes are more complex than a central coordinator. Again, subsidiarity and hierarchy are central to managing this complexity, enabling modelers to model the coordination of large process structures.

In future work, a thorough empirical investigation and evaluation of the coordination process concept, including large relational process structures, will be conducted. This investigation is challenging due to the inter-linked nature of the concepts of coordination process, semantic relationships and relational process structure. Individual evaluations of each concept are therefore rather pointless, as they must be seen and evaluated in a broader context.

Acknowledgments. This work is part of the ZAFH Intralogistik, funded by the European Regional Development Fund and the Ministry of Science, Research and the Arts of Baden-Württemberg, Germany (F.No. 32-7545.24-17/3/1)

References

1. Andrews, K., Steinau, S., Reichert, M.: Engineering a Highly Scalable Object-Aware Process Management Engine Using Distributed Microservices. In: CoopIS'18, Part II. pp. 80–97. Springer
2. Baeyens, T.: BPM in the Cloud. In: BPM'13. pp. 10–16. Springer (2013)
3. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Many-to-Many: Some Observations on Interactions in Artifact Choreographies. In: 3rd Central-European Workshop on Services and their Composition (ZEUS), 2011. CEUR Workshop Proceedings, vol. 705, pp. 9–15. CEUR-WS.org (2011)
4. Hull, R., Damaggio, E., de Masellis, R., Fournier, F., et al.: Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events. In: DEBS'11. pp. 51–62. ACM (2011)
5. Künzle, V., Weber, B., Reichert, M.: Object-aware Business Processes: Fundamental Requirements and their Support in Existing Approaches. International Journal of Information System Modeling and Design (IJISMD) **2**(2), 19–46 (2011)
6. Lohmann, N., Wolf, K.: Artifact-Centric Choreographies. In: ICSOC'10. pp. 32–46. Springer (2010)
7. Müller, D., Reichert, M., Herbst, J.: Data-driven Modeling and Coordination of Large Process Structures. In: CoopIS'07. pp. 131–149. LNCS, Springer (2007)
8. Müller, D., Reichert, M., Herbst, J.: A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures. In: CAiSE'08. pp. 48–63. LNCS, Springer (2008)
9. Nandi, P., Kumaran, S.: Adaptive Business Objects-A new Component Model for Business Integration. In: ICEIS (3). pp. 179–188 (2005)
10. Steinau, S., Andrews, K., Reichert, M.: Modeling Process Interactions with Coordination Processes. In: 26th Int'l Conf. on Cooperative Information Systems (CoopIS). pp. 21–39. Springer (2018)
11. Steinau, S., Andrews, K., Reichert, M.: The Relational Process Structure. In: CAiSE'18. pp. 53–67. Springer (2018)
12. Steinau, S., Künzle, V., Andrews, K., Reichert, M.: Coordinating Business Processes Using Semantic Relationships. In: CBI'17. pp. 33–43. IEEE Computer Society Press (2017)
13. Sun, Y., Xu, W., Su, J.: Declarative Choreographies for Artifacts. In: ICSOC'12. pp. 420–434. Springer (2012)
14. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A Framework for Lightweight Interacting Workflow Processes. International Journal of Cooperative Information Systems **10**(04), 443–481 (2001)