FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Laufzeitverifikation von Geschäftsprozessen unter Verwendung der Blockchain

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Christoph Prybila, BSc BSc
Matrikelnummer 0925463

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Dr.-Ing. Stefan Schulte

Wien, 15. Dezember 2016

Christoph Prybila

Stefan Schulte

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Runtime Verification for Business Processes utilizing the Blockchain

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Christoph Prybila, BSc BSc
Registration Number 0925463

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Dr.-Ing. Stefan Schulte

Vienna, 15th December, 2016

Christoph Prybila	Stefan Schulte

Erklärung zur Verfassung der Arbeit

Christoph Prybila, BSc BSc
Weimarer Straße 70, 1180 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. Dezember 2016

Christoph Prybila

Danksagung

Ich möchte mich zuallererst bei meinem Betreuer Dr.-Ing. Stefan Schulte bedanken. Es passiert schnell, dass man sich bei einem umfangreichen wissenschaftlichen Thema in Details verrennt. Hier hat seine konstruktive und präzise Kritik mich sehr dabei unterstützt auf das Wesentliche kontenzentriert zu bleiben. Auch hat es sehr geholfen, dass ich direkt in den Büroräumen der Distributed Systems Group (DSG) arbeiten durfte. Auf diese Weise konnte ich jederzeit und unkompliziert Feedback zu offenen Fragen bekommen. Dies war eine ungemein wertvolle Hilfestellung.

Auch möchte ich bei den vielen Menschen am DSG bedanken die mich während dem verfassen meiner Arbeit in vielen kleinen Dingen unterstützt haben. Besonders erwähnen möchte ich dazu Christoph, Michael, Olena und Philipp. Obwohl ich eigentlich keine Anstellung am DSG hatte, haben sie mich sofort aufgenommen und ein sehr angenehmes Arbeitsumfeld geschaffen.

Bedanken möchte ich mich zum Schluss bei meiner Familie und den Menschen die mich während meinem Studium begleitet haben. Die viele menschliche (und auch finanzielle) Unterstützung welche ich erhalten habe hat mir dabei geholfen dran zu bleiben und bis zum Ende durchzuhalten. Ich habe lang genug dafür gebraucht danke, dass ihr für mich da seid.

Acknowledgements

I want to thank my adviser Dr.-Ing. Stefan Schulte for his excellent supervision. His constructive and precise criticism enabled me to remain focused in an extensive scientific field of study. By being allowed to work directly at the distributed systems group (DSG) office I was able to collect feedback whenever needed.

Furthermore, I want to thank all people at DSG that have supported me during the course of this thesis. Especially Christoph, Michael, Olena and Philipp created a very friendly and supportive workplace environment. They kept me motivated and provided a lot of help even though it was not their responsibility.

At last, my deepest gratitude goes to my family and people that accompanied through the time of my study. Their emotional (and also financial) support helped me to keep going and to see it through to the finish. It took me long enough, thank you for being there for me.

Kurzfassung

Zentral orchestrierte Workflow-Managementsysteme sind nur eingeschränkt skalierbar. Wissenschaftliche Arbeiten forcieren daher einen dezentralen Ansatz zur Workflow-Choreographie. Hierbei wird die Kontrolle einer Workflow-Instanz zwischen unabhängigen Teilnehmern einer Choreographie aufgeteilt. Für diese geteilte Kontrolle wird ein unabhängiger Mechanismus benötigt mit welchem Workflow-Instanzen dokumentiert und verifiziert werden können.

Um als kryptographische Währung dezentral und unabhängig zu bleiben, bestehen für das Bitcoin-Projekt ähnliche Anforderungen. Bei solchen digitalen Währungen kommt dabei die Blockchain-Technologie zum Einsatz, welche als verteiltes und unabhängiges Medium zur Kontoführung genutzt wird. Im Rahmen dieser Diplomarbeit wird von uns die Eignung der Blockchain für eine verteilte Laufzeitverifikation erforscht. Dazu werden zuerst bestehende Lösungen im Bereich der verteilten Laufzeitverifikation diskutiert und die Eigenschaften von verschiedenen Blockchains beleuchtet. Basierend auf den daraus gewonnenen Erkenntnissen wird ein neuartiger Ansatz zur Laufzeitverifikation abgeleitet und dieser in einem Prototyp umgesetzt.

Der entwickelte Prototyp wird zuerst mit anderen existierenden, verteilten Laufzeitverifikationsansätzen verglichen. Basierend auf übergreifenden Kriterien für Choreographien wird ein funktioneller Vergleich durchgeführt. Es zeigt sich, dass der Einsatz der Blockchain ein nahtloses Monitoring der verteilten Ausführung ermöglicht. Gleichzeitig können Anonymität und Unabhängigkeit der Choreographieteilnehmer gewahrt werden. Weiters ermöglicht unser Prototyp das bedarfsorientierte Einbinden von neuen Choreographieteilnehmern. Es bleiben aber auch Nachteile. So können die Vertraulichkeit der Workflowdaten nicht gewährleistet und eine vorgegebene Ausführungssequenz nicht erzwungen werden.

In einer Leistungsanalyse wird der Overhead unseres Ansatzes ermittelt. Die Verwendung des Prototyps kann zu einer signifikanten Erhöhung der Laufzeit führen. Der größte Einflussfaktor dafür ist die Transaktionsbestätigungszeit (TBZ) in der Bitcoin-Blockchain. Die TBZ betrug während der Analyse im Median 7,741 Minuten. Zusätzlich hat die TBZ auch eine sehr hohe Standardabweichung. Manche Transaktionen haben daher auch wesentlich länger benötigt um bestätigt zu werden.

Es ist möglich den Prototyp mittels eines weniger sicheren Arbeitsmodus zu beschleunigen. Generell ist unser Ansatz jedoch am Besten für Geschäftsprozesse mit zeitintensiven Aktivitäten geeignet, zum Beispiel für Logistik oder Supply Chain Prozesse.

Abstract

To address the scalability limitations of orchestration-oriented workflow management systems, scientific contributions propagate workflow choreographies. The control over a workflow instance is shared between independent participants. Accordingly, an independent mechanism to document and verify the execution of a workflow instance is required.

In the unrelated scientific field of cryptocurrencies, the Bitcoin project utilizes the Blockchain technology as distributed ledger to record payment transactions. This thesis explores the suitability of the Blockchain to create a novel approach to runtime verification. Existing approaches to distributed runtime verification are discussed. Next, the properties of different operating Blockchains are highlighted. Based on these findings a novel approach to runtime verification that utilizes the Bitcoin Blockchain is developed.

The developed prototype is evaluated in a functional comparison. Based on selected criteria, runtime verification approaches are categorized and discussed. Findings show that our Blockchain-based approach enables a seamless execution monitoring while at the same time preserving anonymity and independence of the participants. Some downsides remain. Our proposed prototype enables flexible on-demand participant selection but is not able to provide data confidentiality or to enforce an execution sequence.

At last, the prototype is evaluated in a performance analysis. The usage of the runtime verification prototype can significantly increase workflow duration. The greatest influence factor is the *transaction confirmation time* (TCT) in the Bitcoin Blockchain. A median TCT of 7.741 minutes is recorded in the evaluation. The TCT also exhibits a very high standard deviation, indicating that single transactions take even longer to confirm.

While it is possible to reduce the induced execution overhead of the framework with a less safe but greedy approach, the results suggest that the prototype is most suited for business processes with long running activities like logistic or supply chain processes.

Contents

CHAPTER 1

Introduction

Process-based management is one of the latest trends in business management. Companies document and define how they generate value by defining a set of processes through modelling languages such as ***B**usiness **P**rocess **M**odel and **N**otation* (BPMN) [1, 59, 72]. After the definition, processes have to be correctly enacted in order to generate value. These executions are referred to as *instances* [81]. Controlling large numbers of instances require automated management systems, called ***B**usiness **P**rocess **M**anagement **S**ystems* (BPMSs) [93] respectively ***W**orkflow **M**anagement **S**ystems* (WfMSs) [63]. There are serveral tasks involved in process based management which are summarized under the term ***B**usiness **P**rocess **M**anagement* (BPM) [96].

The term BPM is used to describe both modelling and execution tasks, resulting in ambiguity in many scientific publications. Software systems which are designed to support them (e.g. modelling tools or execution engines) are commonly referred to as BPMS. One approach to provide a distinction is to split the term BPM into the terms *process management* and *workflow management*, where the first describes the modelling task and the second the execution task [87, 96]. An execution engine, responsible for managing the running workflow instances, is then referred to as WfMS. The used terminology for this thesis is further illustrated in Figure 1.1.

First, modeled activities of a process have to executed as tasks. Therefore they have to be mapped to fitting services, available in the system. Classic WfMS implementations follow the centralized approach of service orchestration [95]. In these systems, a central coordinator is responsible for managing the enactment of all workflow instances. The coordinator receives the output of a service and forwards it to the mapped service of the next activity. Therefore, the message interaction of instances is routed in their entirety through the coordinator.

To address the scaling and cooperation limitations resulting from this approach, many scientific contributions propose WfMSs which implement the approach of service chore-

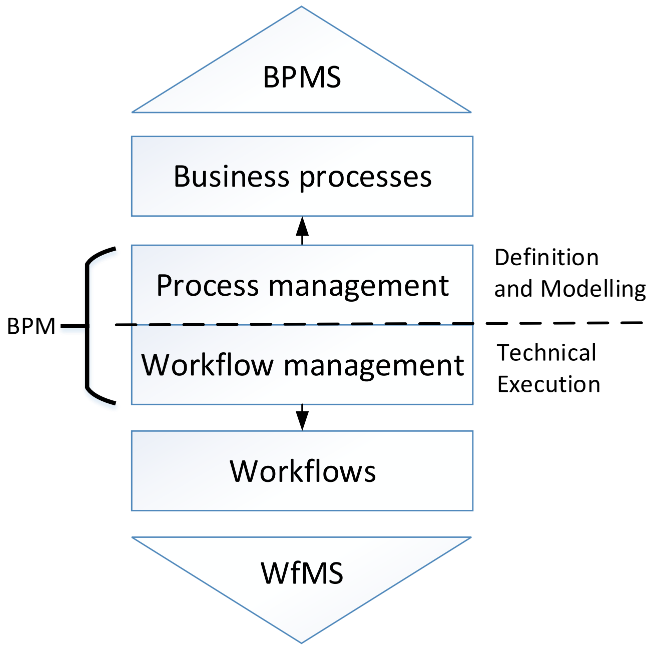


Figure 1.1: Terminology Definition Business Processes in contrast to Workflows. *Source:* Adapted from [96].

ography [75, 77]. Service choreography distributes the control of the enacted workflow instances over different independent partners. Each cooperation partner receives information about the negotiated terms and requirements of a designated part of the choreography and then acts independently. Messages are exchanged directly between the involved partners instead of a central coordinator. This design removes single points of failure and messaging bottlenecks [58]. Scalability is improved and cooperation is simplified. Choreography-oriented WfMSs especially benefit use cases involving *Business to Business* (B2B) cooperation [93].

Through the division of labor, companies strive to focus on their core competences. Therefore, techniques to support cooperation between independent companies are required. Many business use cases, e.g. manufacturing or transportation, can be improved through this approach.

On the other hand cooperation between independent partners creates a lot of coordination effort. Each company manages its services and resources on its own. Global optimization becomes very challenging. The required coordination effort should be mitigated by the proposed choreography-oriented WfMSs. But there are challenges which hinder the adoption of choreography-oriented approaches. The industrial and scientific research about choreography-oriented WfMSs is still in its infancy [2]. There are no commonly agreed techniques to model, convert or execute choreography-oriented processes. For choreography-oriented WfMS there are no commonly agreed designs or architectures. To the best of our knowledge, there exist no mature choreography-oriented WfMS implementations.

Beside the technical challenges, companies also hesitate to join choreographies because of the inherent information sharing discrepancy of the approach. When joining cooperations, companies want to share as little information as possible about their internal operations. All participants are still independent companies and potential competitors. The internal technical facilities of each participant should remain private. Data which is required for the execution of a workflow should only be disclosed to participants that require it. At last, the identities of all participants of a workflow should only be known to the workflow initiator (i.e. process owner).

At the same time, companies that initiate workflows require reliable information about their remotely enacted workflow instances. They want to be able to determine the execution path of an instance and which companies are involved in enacting it.

In a choreography-oriented WfMS, the control over a workflow instance is distributed. Therefore, the documentation of the remote enactment becomes crucial. Since the foundation of each *Business to Business* (B2B) interaction is a contract [71], the records about the distributed enactment of a workflow instance are the only basis for contract verification. This documentation of the distributed enactment of a workflow instance must be indisputable and accepted by all choreography participants [13]. It can then be used to enforce the contract of a choreography. Based on this, a process owner can penalize a choreography participant in case of an incorrect execution of a task. At the same time a company can claim payment from the process owner for the participation in the enactment of a workflow instance.

To enable this kind of documentation, a choreography oriented WfMS must incorporate an end-to-end verification mechanism for its enacted workflow instances. A feature like this boosts the overall trust into the robustness of the choreography and the acceptance of the overall cooperation.

In the unrelated field of digital currencies, one implementation approach already provides a sophisticated mechanism to perform distributed documentation and verification. The cryptocurrency Bitcoin documents and verifies its conducted payment transactions in a distributed ledger, called the Blockchain [99]. Through cryptographic safety measures the funds of a single actor are protected. In the process of paying another actor, both parties must have undeniable proof that the correct amount of money was indeed sent. The Blockchain itself is not maintained by a single financial institution but by a large number of small and independent peers, called *miners* [5]. This boosts the trust in the Blockchain as an independent institution.

Both choreography-oriented WfMSs and Bitcoin face similar challenges when performing verification. The actors involved in Bitcoin transactions are all independent and mostly even anonymous. Payment transactions which have been submitted must be permanent and indisputable [85]. The companies involved in choreography enactment are also independent, even potential competitors. At the same time, the performed tasks of a workflow instance must be permanently documented in a distributed and trusted way.

Therefore, the utilization of the Blockchain for choreography-oriented WfMSs appears promising.

The goal of this thesis is to determine the suitability of the Blockchain to serve as trust basis for distributed and indisputable runtime verification in the context of choreography-oriented WfMSs. In order to achieve this, it is highlighted in which ways a Blockchain can be utilized for choreography-oriented WfMSs. In addition, the prototype created in this thesis is analyzed and compared to other already existing runtime verification variants.

To accomplish this, different existing runtime verification variants are discussed in detail. These runtime verification proposals are selected according to their relevance for choreography oriented WfMSs. In addition to the discussion of existing runtime verification proposals, the properties of different existing operating Blockchains are described. The selection of the Blockchain has great impact on the development of a Blockchain-based runtime verification prototype. A Blockchain may be specialized to serve limited amount of use cases only. New implementations of Blockchains offer a broader range of possibilities but have not yet been extensively reviewed.

Based on the findings of these two sections, a Blockchain-based runtime verification approach is developed in this thesis. The characteristics of the developed prototype are further evaluated in a *functional comparison* similar to the scientific contributions of [8, 45, 88] and a *performance analysis* similar to the work of [70].

In order to highlight and discuss the functional differences between the found runtime verification proposals and the prototype of this thesis, a functional comparison is conducted. Different criteria which are crucial for the flexibility and acceptance of a choreography-oriented WfMS are extracted from the findings of this thesis. The listed runtime verification proposals are categorized by these criteria and discussed. The benefits and trade-offs are highlighted and compared. Through this analysis a positioning of the proposed prototype in respect to other already existing prototypes can be defined.

The performance analysis aims to determine the runtime overhead of the proposed prototype. When utilizing runtime verification in a choreography-oriented WfMS, this mechanism might impact the overall execution duration. First an optimal baseline for the average verification-less execution duration is established. These results are then compared to the execution duration of runtime verification enhanced workflow instances. This provides a good estimate of the impact of the prototype on the execution duration. It is further evaluated and discussed which components of the runtime verification prototype have the greatest responsibility on the produced impact and why.

The remainder of this work is organized as follows: Section 2 describes the different approaches to WfMS architectures and how workflows are currently monitored. Furthermore the concept of Bitcoin and its operated Blockchain is explained. The research challenges of this thesis are defined in Section 3. Section 4 outlines the motivational scenario this thesis operates on. Different Blockchain implementations and variants of runtime verification are described in Section 5. This includes the Blockchain-based runtime verification proposal of this thesis. Section 6 provides a functional comparison on the listed runtime verification proposals of the previous section. Furthermore, a performance analysis is conducted for the proposed prototype. At last, Section 7 summarizes the proposed prototype and the findings of the evaluation.

CHAPTER 2

Related Work

2.1 Workflow Management Systems

2.1.1 Introduction

The enactment of workflow instances must be managed by WfMSs. Different variants of workflow compositions, which categorize WfMSs, are described in Subsection 2.1.2. In Subsection 2.1.3 and Subsection 2.1.4, the individual benefits and drawbacks of the centralized WfMS approach and the decentralized WfMS approach are outlined.

2.1.2 Methods of Service Composition

Classic approaches of WfMSs depict the instance execution engine as a central component, responsible to coordinate the control and data flow of the running instances [95]. It then becomes the single contact point for all instances where messages are sent to and from [84]. Recent scientific approaches propose a different solution where multiple services communicate directly with each other and share this coordination responsibility [2, 36]. The contrary viewpoints can also be described with the methods of *orchestration* and *choreography* [76]. These two methods originate from the concept of service (or workflow) composition [93].

Orchestration

Orchestration prescribes the usage of a central coordinator which oversees the whole execution. This coordinator calls the services in the correct order and forwards the results. The services involved in the composition do not have any information about the overall orchestration, they may not even be aware if they are part of one. Business processes and their workflows which solely are in the control of the same organizational entity (e.g. pool in BPMN 2.0) can therefore be defined as examples of orchestrations

[95]. A classic centralized WfMS is an execution engine for orchestrations. An example business process which is also an orchestration is illustrated in Figure 2.1.

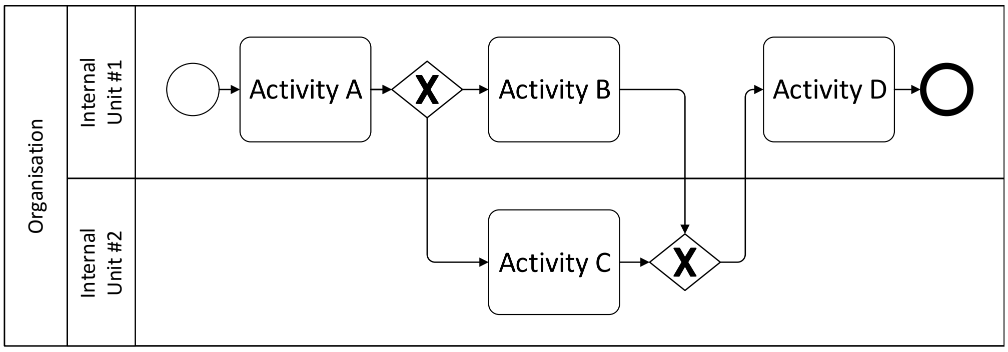


Figure 2.1: Business Process modeled as Orchestration

Choreographies

Alternatively compositions can be described as choreographies [95]. In this method a composition is realized through the collaborative effort of different independent actors (e.g. multiple pools in BPMN 2.0) and their services. There is no central controlling entity. Diagrams describing a choreography focus on the interaction of the actors and services. Instead of communicating through a central WfMS, either the involved actors or their services directly communicate with each other in a P2P fashion.

Even though many scientific contributions describe the great robustness and the scalability potential of this method [75, 77], it also creates a number of challenges. It increases the responsibility of the individual actors and their individual services. They must have at least partial knowledge about the globally agreed choreography. Services must know where to expect which kind of messages from and where to forward their generated results to. Each one must be able to react to unexpected events and adapt the choreography correspondingly.

Implementing an execution environment which is flexible enough to support the enactment of choreography-oriented process modells is a challenging task. Different scientific contributions address this topic by proposing prototype architectures for execution engines [2, 58]. To the best of our knowledge there currently exists no commercial WfMS which supports full workflow choreography. The modelling notation for this method has also not been standardized yet. Different proposals like WS-CDL [90], BPEL4Chor [32] or the choreography enactment part of BPMN 2.0 [72] have not yet found broad acceptance [9]. An example business process modeled with distributed control is illustrated in Figure 2.2.

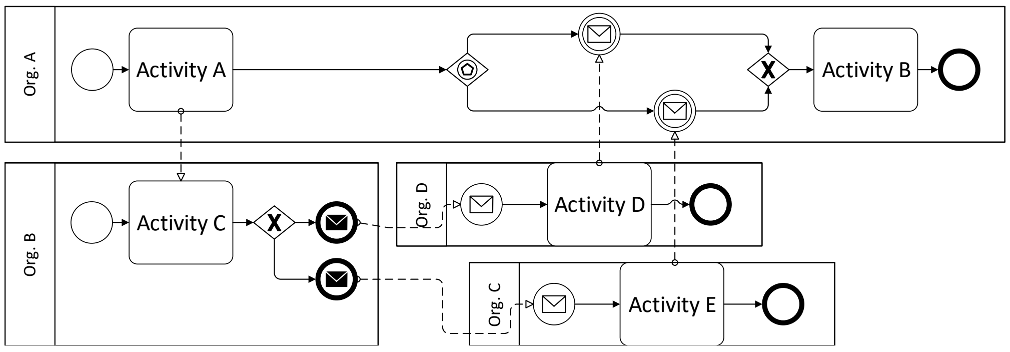


Figure 2.2: Business Process modeled as Choreography

2.1.3 Centralized Approach

Some of the most important characteristics in BPM are reusability and maintainability. Centralized WfMS benefit from increased flexibility. In such systems there is only one controlling entity. The WfMS is in full control of scheduling and resource allocation. Furthermore, tight monitoring can be implemented in order to provide a global and very detailed overview of the state of the running instances [50]. Following this description, a centralized WfMS is an execution engine for service orchestrations.

On the downside many argue that centralized WfMSs do not scale well [75, 77, 84]. Since those systems only have a single coordinator, they become a bottleneck. Messages as well as input and output data have to be routed through the central coordinator.

Additionally, it is sometimes not possible to cover every activity within a single company. This division of labour forces companies to enter inter-organizational cooperation with partner companies [11, 71, 86, 94]. Such cooperation would be suited to be modeled as a choreography but can also be modeled as an orchestration. In the later case one company calls services of other companies only when needed. This can be referred to as *subcontracting*. It suffers from the same bottleneck problems as intra-organizational orchestration [98]. An example business process involving subcontracting is illustrated in Figure 2.3.

2.1.4 Decentralized Approach

Decentralized WfMSs aim to enable the distributed management of workflows. This means a workflow is co-managed by a set of different software systems. Such co-management can either take place on the service level or on the enterprise level. These systems can be seen as an implementation for the service choreography approach. They can be implemented on two different levels.

On the service level, the choreography takes place entirely between services [16, 35, 92]. The management responsibility is assigned to the involved services. Services communicate directly with each other in true P2P fashion. Co-management on the enterprise level

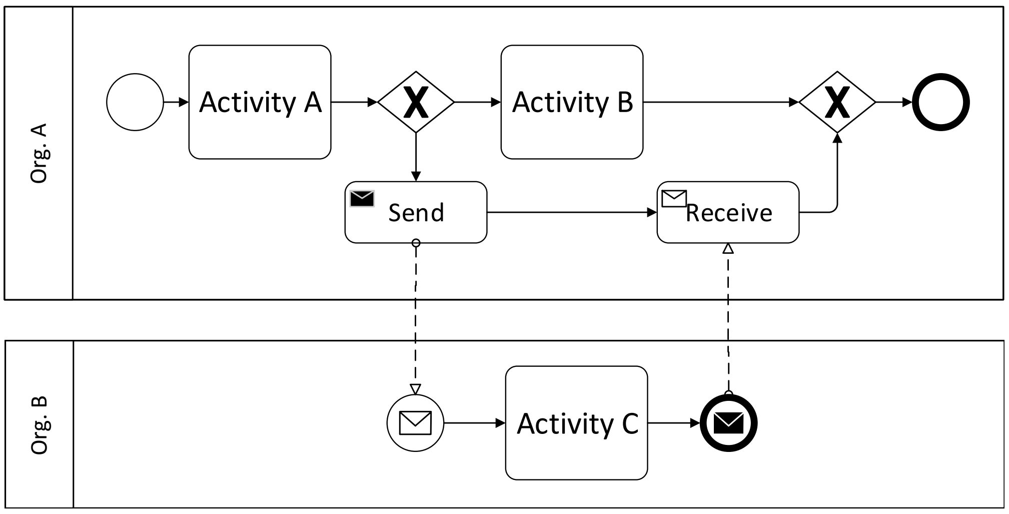


Figure 2.3: Business Process modelled as Orchestration with Subcontracting
Source: Adapted from [86, 98].

places the management responsibility at the involved companies [36, 93]. The message exchange between the companies is regarded as the essence of the choreography. These two viewpoints enable different design variants for decentralized WfMSs.

The majority of available process models are designed using the orchestration method [51]. In order to meet increasing demand, companies require such orchestrations to be executed in a distributed WfMS. The conversion of these orchestrations can be challenging. It is even possible that they fail if modelled constructs are not suited to be executed in a distributed fashion [96]. The goal is to convert a orchestration oriented process model into independent fragments enriched with additional control logic. These fragments can then be executed independently across the decentralized WfMS. There is no common technique to achieve this transformation but various proposals have been made [3, 15, 51, 64].

The distributed approach creates lots of opportunities in comparison to the centralized one. By distributing control and enactment, scalability and robustness are improved [58]. There is no single point of failure or messaging bottleneck. Instead of relaying the messages through a single coordinator they are now sent directly between the cooperating software systems. On the other hand new challenges are created through this approach. Cooperation creates an increased demand for coordination, in economic sciences also referred to as transaction cost [80]. It is the task of software systems like WfMSs to keep these transaction cost as low as possible by automating coordination efforts and by providing means for distributed information sharing.

2.1.5 Summary

Orchestration-oriented workflow engines have reached their limit in terms of scalability and cooperation. These engines are not suited for large numbers of instances and

Business to Business (B2B) cooperation. Scientific contributions propose decentralized choreography-oriented engines as the solution. This research is still in its infancy. There are no commonly agreed WfMS designs, and therefore no mature implementations. The main challenge is the decentralized coordination between independent actors.

2.2 Monitoring of Workflow Enactment

2.2.1 Introduction

WfMSs have to monitor the instances and services operating within their system. The data collected is required to assess the health of the overall system, to react to arising problems and to further optimize the system in real-time. This real-time feedback process is also referred to as *business activity monitoring* (BAM) [50].

The remainder of this section is organized as follows: Subsection 2.2.2 provides an overview about the general feedback loop of monitoring systems. Complex event processing, a prominent implementation of the feedback loop, is described in Subsection 2.2.3. Finally Subsection 2.2.4 describes the specific challenges and possible solutions for monitoring of B2B choreographies.

2.2.2 The Monitoring Feedback Loop

There is a multitude of data that can be collected in a WfMS. Important artifacts for monitoring are the enacted workflow instances, the services they utilize and the servers those services operate on. In a first step basic facts have to be recorded. Workflow instances and services are software constructs, therefore only logical data may be monitored. Facts regarding those two types of artifacts are for example the start timestamp and the end timestamp of a workflow instance or a service call. For the provisioned servers, physical data like RAM or CPU usage can be recorded.

The next step in monitoring is the aggregation of the collected facts into more complex data sets [81]. Metrics specify how basic facts can be used to produce such metadata. For instance the start timestamp and end timestamp can be used to calculate the workflow instance duration. A number of CPU usage observations can be used to calculate the average CPU usage over a specified timespan.

In order to react upon this vast amount of information, companies specify *key performance indicators* (KPIs) [29]. Those KPIs are calculated from aggregations of large amounts of facts and metadata. An example for a KPI is the average duration of all workflow instances. A change in a KPI is the first indicator of possible problems within a WfMS. The operating software components of a WfMS can further analyze the related metrics and facts of an affected KPI. When the problem sources are identified, the system is able to react to them. This way the feedback loop is closed.

2.2.3 Complex Event Processing

The monitoring of WfMS, enabling the previously mentioned feedback loop, are summarized under the term BAM [50]. One approach of realizing such system is *complex event processing* (CEP).

Observed facts are published as *events*. It is the core element of CEP [62]. When something relevant is observed in a sub-system, a so called *event generator* is responsible for creating and publishing a new event [50]. Through this approach the recorded facts are saved in a standardized event format already enriched with identification, time and category information. How these generators are implemented is not relevant for the concept of CEP.

Through the definition of aggregation mechanisms (i.e. metrics), sometimes also referred to as *rules*, connected basic events (i.e. basic facts) can generate additional complex events [43]. The standardized format of events combined with additional meta information enable expressive and flexible rule definition languages. Through this, companies can define and manage large sets of rules for their system landscape. There is a basic multi-level architecture which describes common implementations of CEP [50]. On the lowest level reside the event generators. As mentioned, those generators can be implemented in various ways. They may observe data from physical sensors and servers or non-physical software systems. In order to observe the software constructs relevant in a WfMS (e.g. workflow instances and services) different approaches with varying invasiveness exist.

Most events of a WfMS are generated at the utilized services. Therefore event generators are likely to be placed in the same runtime environment or the same server as a service. The most invasive method of creating an event generator is integrating it into a service itself. Less invasive approaches monitor information about a service from the outside. By observing a service's process behaviour, its related databases, its related communication and its generated log files a process generator can anticipate the status of a service without being actually integrated [29, 50].

Event processors reside on the second level of the CEP architecture [50]. Those software components are responsible for collecting basic events and generating complex events according to defined rule definitions. They represent the logical core of the whole CEP system and may themselves again be separated into different processing agents performing different operations[62].

The desired KPIs of a company can be represented as complex events. Therefore as a result from the second layer, the KPIs of the companies are omitted to the next layer. The third and last layer consists of the *event consumers* [81]. These are systems which require KPIs as input, for example managing dashboard and most important the (possibly distributed) workflow coordinator component of the WfMS.

Each of the described components from the different levels of the CEP architecture can be operated on individual physical machines. The CEP itself therefore can be deployed as a distributed system and is very scalable [50].

2.2.4 Monitoring in B2B Choreographies

As previously described, CEP is suited for the application in distributed systems. But at the same time it is designed for monitoring within the organizational boundaries of a single business entity. During the workflow enactment of B2B scenarios new challenges have to be faced. Companies strive to externalize support activities through cooperation, they want to be able to participate in flexible and short term cooperation [11, 71, 86, 94]. Internal WfMSs and BAM systems should be extended to be used in such cooperation with as little effort as possible [78].

While workflow choreographies are enacted remotely companies still require monitoring information about them. At the same time, when locally hosting a remote workflow instance from another company, no insight about internal system states and events should be exposed to the remote company [71, 94]. Due to the lack of mature choreography execution environments, there currently also exist no mature choreography monitoring techniques [9]. Only a number of scientific proposals address these challenges. The remainder of this section describes a selection of them.

[71] cover the first step of cooperation, namely the contract negotiation. In order to form flexible, short term cooperation with a legal foundation, autonomous software agents must be able to agree on simple legal contracts on behalf of their companies. While important aspects of such contracts can be defined in ***S****ervice* ***L****evel* ***A****greements* (SLAs), other aspects still require standardization. [71] propose a markup language which supports the definition and exchange of such extended sourcing contracts. Besides simple SLAs, other organizational sections, like monitoring, are included in such contract. In the monitoring section it is defined what monitoring information should be provided by which participant and through which kind of interfaces the information is shared.

In addition to contractual definitions, monitoring in distributed workflow enactment must also be defined during the modelling of the service choreographies. [9] describe this aspect in a semi-static choreography scenario. As in the previously described work, they argue that the first step in cooperation is a contract. But before the actual enactment, participants also have to agree on a choreography diagram which serves as a common definition of the choreography. Monitoring aspects have to be taken into account in such diagram.

They claim that BPMN 2.0 enables the definition of choreographies but does not provide sufficient means to define monitoring for them. "BPMN [...] already supports including monitoring injection points with its monitoring and auditing element. However [...] the specification claims details are out of scope and are left to the implementing BPMN engines." [9] Therefore they extend these monitoring injection points to support choreographies. Most importantly they address the issue of information correlation in cross organizational monitoring through specific identification schemata.

BPEL4Chor is a WS-BPEL-based scientific choreography definition language. [94] propose an event-based choreography monitoring prototype through the definition language BPEL4Chor. They aim to create a choreography wide BAM system. It is argued that companies run their own intra-company WfMSs and BAM systems. As soon as outsourcing decisions are made, normal processes suddenly become B2B processes. As a common example they describe basic shipment processes.

In a centralized WfMS the enacted workflow instances are all managed by the same central coordinator component. Therefore all instance-related events can be created by simply placing an event generator at this central component. In the case of a distributed WfMS, realized as a company oriented choreography, each monitoring components of each company has to employ an event generator [94].

The monitoring aspects of choreographies have not yet been integrated into the language. Therefore, [94] extend BPEL4Chor with an event-oriented XML-based monitoring agreement. In this document, each cooperation partner defines the events she is willing to share. To address the privacy concerns of the participants, events can only be defined based on the publicly available choreography. How each participant maps the public choreography activity to internal processes remains hidden.

The challenge of event correlation is also addressed by [94]. Through common agreement on IDs, events can be correlated to specific activities and choreography instances. In order to form a choreography wide BAM system, the local BAM system of each participant is provided with the negotiated monitoring agreement document. Through this definition each system knows which basic events to record and which complex events to calculate. The document also specifies how the events should be published.

BAM systems not only aim to record events but also to enable WfMSs to adjust accordingly. The main goal is not to react to occurring SLA violations but to avoid them altogether by taking actions in advance. This violation prediction becomes especially challenging in the loosely coupled environment of B2B choreographies. [17] propose a prediction mechanism specialized for choreographies. For their prototype they utilize the choreography runtime from the CHOReOS EU project [18]. This runtime is able to enact QoS-aware choreographies, modelled in BPMN with the extension Q4BPMN. According to [17] choreography SLAs cannot be defined too specific because of the abstract and unspecific nature of B2B choreographies. Therefore their proposed prediction mechanism is able to extract implicit unspecified prediction rules during the enactment itself when opaque activities are mapped to concrete services.

2.2.5 Summary

WfMSs assess the healthiness of their instances and services through monitoring. Based on the collected data, WfMS react on problems and issue optimizations. CEP is a scalable monitoring technique, capable of handling vast systems. Events are collected from various sources and aggregated into interpretable KPIs. In B2B WfMSs, participants require monitoring for remote workflow instances. At the same time no internal information should be exposed. Therefore, B2B monitoring has to be explicitly addressed during the negotiation, the modeling and the implementation of a choreography.

2.3 Bitcoin

2.3.1 Introduction

The success of currencies and the payment transactions conducted with them always have been depending on trust. Instead of exchanging one good for another, it can be purchased through the transfer of tokens i.e. money. Another requirement for currencies is the limited supply of its tokens [99]. In order to retain value, it must not be possible to duplicate existing money tokens. Furthermore, the production of new tokens must be a controlled process.

Governments issue currencies to their citizens. Their federal banks control the production and distribution of fresh money. In addition great effort is conducted to prevent forgery by applying security features. Both these measures ensure the limited supply of modern physical money. Governments have to ensure and facilitate the trust into their currency through their financial and fiscal politic. The general healthiness of a state's economy is used as an monitor for the success of these politics [4].

Digital money must be handled differently than physical money. Since it can easily be replicated, the exchange of digital tokens is not a practical replacement for physical money. Known as the *double spending problem*, a malicious trader could copy digital money tokens and spend them multiple times for different transactions [85]. The amount of money one person holds is recorded in a ledger. The ledger must then be managed by a trusted institution. A digital payment transaction is simply conducted by reducing the balance in one person's ledger while increasing the balance of another person's ledger.

This puts great power into the hands of the book keeping institution. Customers have to fully trust such institutions to keep their accounts secure, to enact the transactions correctly and to keep their data anonymous [99]. In practice, only a small group of big financial institutions carry out most of the digital payment transactions over the internet today [44]. The performance of these institutions is not flawless. There exist for example a multitude of reports about problems and lost money from customers of Paypal [44].

The cryptocurrency Bitcoin aims to solve these mentioned problems by utilizing decentralisation and cryptography technology. From its original proposal in 2008 [69] to the present day the interest in Bitcoin has been growing steadily. It promises to become the first digital currency which truly reflects the properties of physical money. Like cash in a person's wallet, a Bitcoin can only be spent by its owner and cannot be duplicated. The digital storage, book keeping and validation of payment transactions is not controlled by a small group of big financial institutions, but by a vast number of smaller independent actors. The owning as well as the spending of a Bitcoin can be conducted completely anonymously. Though far away from perfection, Bitcoin is able to fulfil all of these features to a large extent [99].

The adoption has been slow yet steadily increasing [99]. At the same time the value of Bitcoin, measured in its exchange rate to fiat currencies, has been very volatile. Trust into the currency itself is derived from the robustness of its technological foundation, which withstood every breaking attempt till the present day [99].

The smallest unit in the Bitcoin currency is not the Bitcoin (BTC) itself, but the Satoshi (SAT). One Bitcoin can be split into one hundred million Satoshis ($1_{BTC} = 100,000,000_{SAT}$). This way the cryptocurrency can adapt to increasing value and an increasing user base [24].

The remainder of this section is organized as follows. How a Bitcoin transaction is performed is described in Section 2.3.2. The distributed ledger management system, which is the core innovation of Bitcoin, is explained in Section 2.3.3. Finally in Sections 2.3.4 and 2.3.5, the open challenges as well as possible enhancements and variants of Bitcoin are outlined.

2.3.2 Transactions

The Bitcoin equivalent to an account is called *Bitcoin address*. It is a identification string created from the hash of a public key [5]. Therefore it is necessary to create a private/public key-pair for every new address. The private key then commonly serves as the access mechanism for funds associated with the address. Every participant of the network can easily create as many Bitcoin addresses as desired. Payment transactions are issued between Bitcoin addresses.

A Bitcoin itself is represented as a chain of of transactions [69]. A common transaction is composed of an input section and an output section. The *owner* of a Bitcoin has access to the output of the latest transaction in which the corresponding coin was used. In order to spend it, the owner has to issue a payment transaction in which the output of the previously latest transaction becomes the input to the new transaction [85]. The payer specifies the new owner of the Bitcoin by directing the output of the new transaction to a specific Bitcoin address. Since the output of the previously latest transaction now has been used, it is considered spent and cannot be used as input for another transaction. After creating the overall transaction information, the data is signed by the private key of the payer's bitcoin address and broadcasted to the Bitcoin network [34]. Since Bitcoin can be split into Satoshis, transactions usually contain fractions of Bitcoins.

A standard Bitcoin transaction can have multiple input and output parts defined in its corresponding sections [99]. Thus, the input section of a single new transaction can be composed of multiple parts referencing the outputs of multiple old transactions. Likewise can the resulting amount of a transaction be split into multiple output parts. These output parts can then be distributed to multiple different Bitcoin addresses. This enables the payer to pay multiple people at once and receive change a the same time. Change becomes necessary since one output part of a transaction can only be consumed as a whole. Thus an output part of a transaction cannot be partially consumed. If the input of a transaction surpasses the desired payment value, one output part of the transaction can point back to an address of the Bitcoin's previous owner thus returning the change. Figure 2.4 outlines the output to input relations of standard transactions.

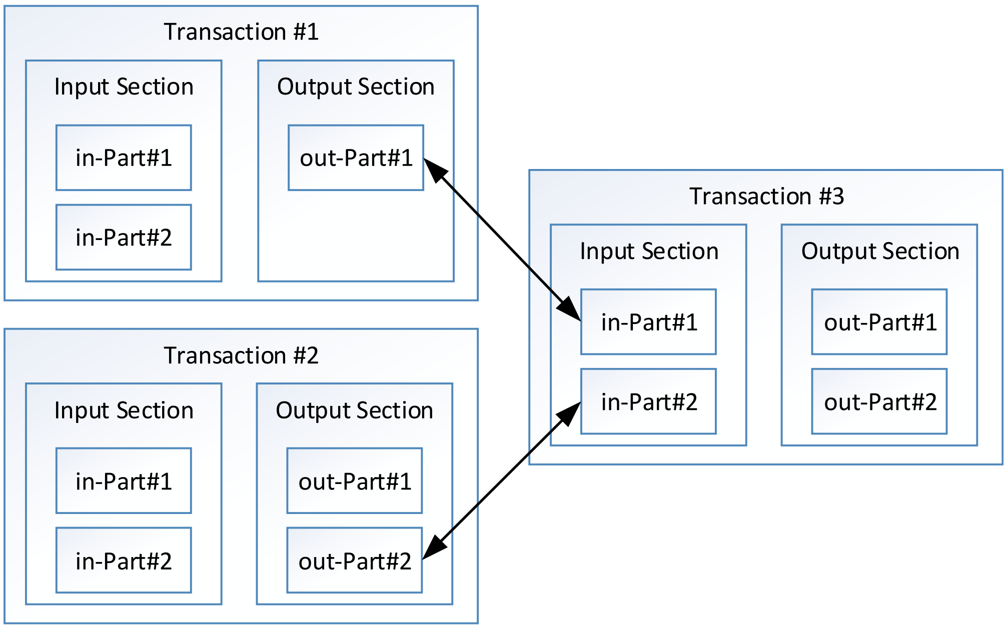


Figure 2.4: Output to Input Relation in Bitcoin Transactions

Bitcoins are created through special *coinbase* transactions which require no input. These transactions represent the start of a Bitcoin chain. Coinbase transactions are submitted during the mining process (see Section 2.3.3) as incentives [5].

The technical mechanics of a standard transaction are as follows. Output parts are sums of Bitcoin money paired with small scripts written in a custom Bitcoin scripting language. These scripts guard the funds associated with the output. In order to access an output part, the corresponding script must be supplied with a parameter that renders its result to *true* [99].

The most common script requires a signature as input, created from the payee's Bitcoin private key. A payer can very easily create this script since the payee's Bitcoin address and its associated public key are available. Only the holder of the corresponding private key is able to create the required signature, thus making the output only accessible to the owner of the destined Bitcoin address [5]. An input part of a transaction only contains a reference to its corresponding originating output part and the necessary parameters to render the script of the output to *true*. Through this mechanism, everybody receiving a broadcast transaction can verify if the transaction is really authorized to access the specified outputs since all necessary informations are available. Figure 2.5 illustrates the common access mechanic for Bitcoin funds.

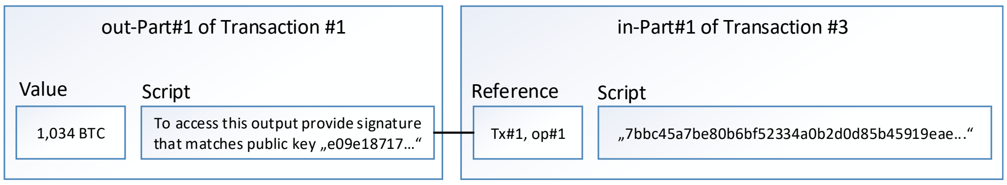

out-Part#1 of Transaction #1
Value
Script
1,034 BTC
To access this output provide signature that matches public key „e09e18717…“
in-Part#1 of Transaction #3
Reference
Script
Tx#1, op#1
„7bbc45a7be80b6bf52334a0b2d0d85b45919eae…“

Figure 2.5: Output Access Mechanism in Bitcoin Transactions

After the assembly, the transaction data is signed by the payer and broadcasted to the network. There exist other protocol variants where also the signature of the payee is required, thus requiring the agreement of both actors before issuing a transaction.

2.3.3 The Blockchain

One of the most important innovations of Bitcoin is the Blockchain mechanism. It enables the distributed, secure and undeniable book keeping of the Bitcoin transactions. In the Bitcoin system, all issued transactions are public [44]. This means everyone can fetch the historical transaction data and determine how much funds are associated with certain Bitcoin addresses and what payment transactions have been conducted. This is in fact a desired functionality. Anonymity is achieved through the abstraction of Bitcoin addresses. This ensures privacy while all transaction data is publicly available. It is advised to create a new Bitcoin address for every new transaction in order to increase anonymity even more.

As described in Section 2.3.2, the access to funds is stored in transaction outputs protected by access scripts. The commonly used script ensures that only the owner of the payee's Bitcoin address can access it. Therefore, a malicious trader is not able to access the funds of other people [99]. I.e. it could easily be verified that a newly created transaction is not valid since the scripts guarding the used funds would not render to *true*. Instead a malicious trader only has the possibility to use a personal transaction output multiple times, i.e. to spend the same Bitcoins over and over.

To avoid this problem, the public and distributed book keeping mechanism of Bitcoin becomes necessary. The Blockchain provides an unchangeable history of all issued Bitcoin transactions from the past. This way, it can be verified if a Bitcoin has been already spent thus denying the double spending of funds. Furthermore, timestamping of newly issued transactions is enabled. When a new transaction is broadcasted to the network, it is first verified and then added as a new record to the transaction history. Last but not least, the Blockchain also enables the controlled creation of new Bitcoins, thus slowly increasing the amount of existing Bitcoins until a certain threshold [85]. The operation and maintenance of the Blockchain is performed by a large set of individuals and companies. In the Bitcoin system, those individuals are referred to as *miners* [5].

As the name points out, the Blockchain consists of a series of interconnected data blocks. Each block contains a number of transactions as well as a link to the previous block, incentive information and a *proof of work* [5]. The most crucial feature of the distributed

Blockchain is the synchronization between the miners. It must be ensured that all participants of the network agree on the same Blockchain i.e. the same transaction history. To perform a double spending attack, a malicious trader would have to rewrite the Blockchain in order to delete or alter an old transaction containing previously spent Bitcoins.

To address this, the creation of new blocks requires a proof of work. In order to create a new block, a miner has to solve a computationally difficult problem. It should require some minutes to solve on up-to-date hardware. First, a miner collects new transactions which have been broadcasted throughout the network. The previous block in the chain is hashed. This hash is stored as link in the new block. As a reward for mining, the miner is allowed to add a coinbase transaction with a specified amount of Bitcoins to the block [85]. At last the block contains a *nonce* i.e. a field which is iterated in order to solve the problem. The miner must ensure that the hash of the newly created block has a specified number of leading zeroes [99]. This problem is essentially solved through try-and-error. After each attempt the nonce is iterated until a match is found. A newly created block is then broadcasted to the network.

The different miners compete against each other. If a new block is broadcasted, all miners first validate the contained transactions and then start *mining* on a new block [99]. Miners show their approval of the validity of a block by accepting it as the new head of the chain. The reward contained in each new block works as the incentive for the miners to keep trying. Currently the reward is made up of newly created Bitcoins and a percentage transaction fee. The number of newly created Bitcoins is determined by the Bitcoin protocol and continuously decreasing. In the year 2140 no more new Bitcoins will be created and the incentive will be made up from transaction fees alone [85]. To address hardware improvements, the difficulty of the mining problem (i.e. the number of leading zeroes) is periodically adapted.

If two new blocks are propagated at the same time, a conflict occurs. Every miner starts working on the first received block but keeps the alternative block in memory. Now the network is partitioned into two parts, mining on two versions of the Blockchain. By following the simple rule *"adopt the longest chain"*, the conflict is resolved as soon as the next block is created. Since the new block is placed on top of one of the two chains, the tie is broken and the longer chain is chosen [99]. In this case, the other chain is discarded.

The validity of the Blockchain is therefore protected by the majority of the miner's computing power. A malicious trader that wants to rewrite the transaction history would need to change the corresponding block in the chain and then recalculate all blocks which have been added on top of the changed block. While doing so, the attacker would have to out-race the entire network in order to create a chain longer than the valid chain. The more blocks are placed on top of the changed block, the less likely the attacker is going to succeed.

It is claimed that an attacker would require more than 50% of the networks total computing power to conduct this attack [5, 44, 69, 99]. The double spending attack therefore is reduced to the so called *50% attack*. This scenario becomes less likely the more miners participate in the mining process. It is also argued that an attacker who controls 50% of the network's computing power would earn more money through staying honest and collecting rewards than by double spending money [99].

2.3.4 Current Challenges and Proposed Solutions

There exist numerous challenges that Bitcoin has yet to solve. While not being perfect, [99] points out that Bitcoin only needs to compete with other forms of digital payment in order to be accepted. The rest of this section provides an overview about the most pressing challenges of Bitcoin.

Scalability

The size of the Bitcoin network has been steadily increasing over the past years. Up until now it was able to serve the increasing demand, but problems for future growth are already emerging. Currently it takes about ten minutes for a new block to be created [44]. In the worst case the payees have to wait the full timespan to get confirmation for their transactions. Even longer if they require stronger assurances [85].

At the same time, the Blockchain is already a highly wasteful mechanism [99]. All miners have to perform verification of published blocks and compete with each other in the creation of new blocks. A vast amount of work is performed redundantly which wastes a lot of energy. In addition to that, all broadcasted messages must be relayed repeatedly to ensure that they cover the majority of the distributed network.

The maximum size of a Block is currently set to 1_{MB} which limits the amount of transaction that fit into it. This results in an approximate throughput of one transaction per second (tps) [82]. For the required speed of a full-fledged global payment medium, the transaction rate of credit institutions provides a good benchmark figure. Visa processes on average $2,000_{tps}$ [21]. In order to increase the transaction throughput, Bitcoin must either reduce the block creation time or increase the maximum block size.

Increasing the maximum block size would pose new challenges to miners. Since a complete copy of the Blockchain is needed for in-depth verification, miners would have to handle an increase in storage of the magnitude of TB per month. The verification of bigger blocks requires more computing power. Both of these tasks introduce additional barriers for new miners.

The worst consequence is the increased propagation delay. Bigger blocks also take longer to propagate [31]. This significantly increases the likelihood of conflicting blocks which reduces the resilience to double spending attacks and the overall efficiency of the network [31].

In order to reduce the block creation time, the required proof of work would have to be reduced in complexity. This approach suffers from similar problems like the previous one.

A reduced proof of work results in reduced security against double spending attacks and increased block creation results in more conflicting blocks.

The transmission speed of the network cannot be increased since Bitcoin operates on an overlay with independent members. Conflicts through propagation delays are therefore unavoidable. Approaches addressing the scalability problem therefore focus on making the Blockchain more resilient to synchronization conflicts.

In their work [82] propose a change to the Bitcoin protocol, named *GHOST*, and evaluate its resilience during higher transaction throughput. They propose an alternative conflict resolution rule. When conflicts occur, miners should **G**reedily adopt the **H**eaviest **O**bserved **S**ub-**T**ree. The approval of blocks is demonstrated by miners through extending them with new ones. The GHOST protocol takes advantage of this and associates weight to blocks. A block's weight is defined through the number it is extending blocks (including forks). The chain with the heaviest sub-tree is chosen as the valid one.

Strategic Behaviour

Miners perform their work in order to collect a reward. But the Blockchain's incentive mechanism is designed as a "winner takes it all" mechanism [44]. All miners compete against each other, but only one miner is rewarded. Miners thus only receive an irregular income. One logical conclusion is to maximize those profits through strategic behavior.

Specialized hardware is used for the hashing problem, in order to gain an advantage [14, 99]. Common CPUs where quickly replaced with more potent GPUs. Nowadays, miners use specialized chip equipment which is specifically designed to perform hashing operations. This equipment of course is much more expensive than regular hardware. This leads to factual entry restrictions to the group of Bitcoin miners.

Miners form loose cooperatives, called *mining pools*, and share the collected profits [85]. This changes the irregular and high income to a regular and low income which reduces the overall risk for all participants.

Strategic delaying is another strategy to gain an advantage, especially for bigger pools [34, 99]. It has been observed that sometimes new transactions are not relayed to other participating miners to hinder others to collect the transaction fee of those transactions.

Mining pools have become so popular that almost every miner is member of a pool. Thus, the mining of Bitcoins is again fragmented into a small number of big collectives. Some pools repeatedly came close to cross the dreaded 50% computing power threshold [85].

In summary it can be stated that the economies of scale in the mining process reduce the overall decentralization of the network which threatens the underlying goal of Bitcoin.

Privacy

Through the abstraction of Bitcoin addresses, the trade with Bitcoins should be private. But in practice this privacy is weakened, especially for individuals. Since the complete trading history is public, an account must be exposed only once in order to follow all trading through it. Even if Bitcoin addresses are used only once, it is possible to deduce information from it [65].

This weak point starts at the Exchanges. If Bitcoins are not acquired through mining they need to be purchased at an Exchange against fiat currency. If not carefully conducted, one has already revealed the own identity at such Exchange, making it very easy for the corresponding company to follow all traffic through it [44].

To further improve privacy, various proposals have been made. Mixing services aim to harden the traceability of addresses and transactions by submitting original transactions as many smaller ones [27]. An original transaction is then hidden behind a convoluted mix of many different subsequent smaller transactions between newly generated addresses. The main benefit of this approach is that it requires no changes to Bitcoin.

Other proposals suggest protocol changes to improve privacy through advanced cryptographic methods. ZeroCoin is such an approach [66]. It prevents the traceability of transactions through zero-knowledge proofs and one-way accumulators. [7] further improve this approach by hiding the transaction amount and the involved Bitcoin addresses.

Legal

Up until now, Bitcoin has not yet attracted enough attention to be relevant for one nation's fiscal politics. Therefore, there are still very few legal foundation on how to treat the possession of Bitcoins [53]. Through Bitcoins design, governments will have significantly less control in comparison to fiat currencies. While this is one of the reasons Bitcoin was created, regulatory institutions could view this as a problem. This could lead to the general prohibition of Bitcoin.

The Bitcoin market is still relatively small and not controlled by central institutions. Different hypes around the system repeatedly cause the value of Bitcoin to greatly increase and decrease in short spans of time. While this was not enough to hinder the further growth of Bitcoin, it currently presents problems for practical sale applications like refunds [44].

Design

While the Bitcoin concept introduced lots of robust new inventions, it still contains some design flaws that cause problems.

As an decentralized and open source system there is no strictly enforced update policy. Still, all participants have to use compatible versions of the Bitcoin software. This makes the introduction of updates very difficult [99]. To facilitate continuous development, two version of the Bitcoin Blockchain are operated. A testing realm, called *testnet*, is operated with a block creation time that is reduced by 50% and relaxed requirements for published transactions. Testnet Bitcoins have no real world value and can be acquired for free by developers at so called *testnet faucets*. The operative Blockchain of Bitcoin where the real currency is traded is referred to as *mainnet*.

Incentives in the block creation process motivate miners to create new blocks. There are also other tasks which are vital for the system, but are not rewarded. Especially for the underlying communication tasks, critical in a decentralized network, there are currently no rewards for the participants [41]. This affects one of the most crucial tasks, the forwarding of broadcast transactions and blocks. As described in Section 2.3.4, this even leads to opportunistic behavior.

Since the output of a transaction is protected by a private/public key-pair, the loss of a corresponding private key is fatal. This way Bitcoins become unusable [14]. Since there will always be only a finite number of Bitcoins by design, this could lead to the slow but steady reduction of the overall volume of Bitcoins.

2.3.5 Alternative Usages

The Bitcoin protocol as well as its provided software is open source. Through this emerged many adoptions of the technology. Some introduce new variants of cryptocurrencies with changed functionality. Others utilize Bitcoin for completely new use cases. The adoptions can be categorized into approaches that build on top of the Bitcoin Blockchain and approaches that require the operation of a separate Blockchain. The rest of this section provides some examples of these adoptions.

Numerous alternative digital currencies already exist. Those so called *altcoins* often deviate only slightly from Bitcoin by replacing specific features. Most altcoins require their own Blockchain.

Litecoin [56] introduces an alternative hashing algorithm in order to break the dominance of specialized hardware during mining. Furthermore, the block creation rate is increased to approximate 2.5 minutes. Litecoin is specifically suited for large numbers of small value transactions.

In order to reduce the wastefulness of the mining process, Primecoin [52] adds some intrinsic value to it. The proof of work mechanisms of Primecoin involves the discovery of long chains of prime numbers. These chains can also be used in number theory, regardless of the mining outcome.

A completely different usage of the Bitcoin technology is the Namecoin [48] project. It provides a decentralized key-value store based on the Blockchain mechanism. On top of this store, Namecoin operates a decentralized Domain Name Service (DNS).

The Ethereum [37] project enhances the Bitcoin technology even further. It extends the Bitcoin scripting language and makes it Turing complete. This enables the creation and distributed execution of arbitrary contracts on top of Ethereum's Blockchain.

A less invasive alternative usage is the Originstamp [42] approach. It provides a decentralized general purpose timestamping service. Unlike the previous two approaches, this project operates on top of the Bitcoin Blockchain. It directly stores submitted hashes in the Blockchain by converting them into Bitcoin addresses. This service can be accessed through an open Web Service or an API.

2.3.6 Summary

The digital currency Bitcoin aims to incorporate the properties of physical money through cryptographic measures. Bitcoins cannot be duplicated or forged, they can be spent anonymously and they are not controlled by any financial institutions. Instead of digital tokens, Bitcoins are managed in an public distributed ledger, called the Blockchain. The Blockchain is maintained by a large number of independent peers, called miners. Distributed consensus is achieved through the majority voting of the available computing power. When issuing a payment, a payer submits a transaction to the mining network that transfers a Bitcoin's ownership. The transaction is then persisted into the Blockchain.

While being a successful and resilient cryptocurrency, Bitcoin faces a number of of challenges, the most difficult being scalability. In order to become a global cryptocurrency, the transaction throughput of Bitcoin has to be increased. Due to incorporated security measures in Bitcoin's design, this is currently not possible.

There exist lot of variants to the main Bitcoin approach that create alternative implementations of the cryptocurrency. The Blockchain itself is also utilized in other projects which are completely unrelated to cryptocurrencies.

CHAPTER 3

Research Challenges

Current orchestration-oriented WfMSs suffer from limited scalability and cooperation. The transition to choreography-oriented WfMSs is regarded as the solution to these challenges. The distribution of management for workflow instances across equal participants removes performance bottlenecks and single points of failure. Furthermore, choreography-oriented approaches are suited to define cooperation between independent companies.

At the same time, choreographies create new challenges. B2B choreographies have increased requirements for coordination and trust. The decentralized enactment requires companies to hand over the control of their workflow instances to remote partners. Process owners may not know which partners participate in the enactment of a choreography instance. In order to encourage companies to join choreography-oriented partnerships, different mechanisms to facilitate trust between the partners are needed. The formed cooperation contracts need to be verifiable in order to make them enforceable.

Therefore, companies which participate in choreographies need to be able to collect trusted information about their remotely enacted workflow instances. At first glance a monitoring system seems suitable to address this challenge. CEP-oriented monitoring systems can record logging data and distribute it to the relevant peers. Furthermore, these monitoring systems can be enhanced to include correlation-ids which enable the cross-organisational correlation of events [12, 94].

Still, this monitoring approach is not suited to ensure end-to-end runtime verification. Though CEP events can be enriched with security meta-data, certain security issues remain. Each CEP system operates under the control of one of the choreography's participants and can not be regarded as a trusted system for the others. Events are commonly broadcast asynchronously, therefore messages may be lost, intercepted or not sent at all. There is no shared agreement between the participants on which events did occur and which did not. CEP-oriented monitoring systems are therefore not suited to serve as trusted runtime verification system. A more detailed explanation about this matter can be found in Section 5.2.

Runtime verification in choreography-oriented WfMSs has to be provided through dedicated mechanisms. Process owners must be able to trace the execution path of a workflow instance across the boundaries of the different participating companies. At the same time, a company has to be able to proof its participation in a choreography. The collected information must be trustworthy enough to serve as legal basis for contract enforcement.

In the unrelated scientific field of digital currencies, the cryptocurrency Bitcoin is utilizing the Blockchain, a distributed ledger, for transaction verification. Similar to choreographies, Bitcoin transactions take place between independent partners and must be secure enough to be considered indisputable. Therefore, the Blockchain seems to be a promising technology to created new approaches of workflow runtime verification. The Blockchain's applicability in this domain will be highlighted in this thesis.

3.1 Research Challenges

At first glance, both choreography-oriented WfMSs and Bitcoin require similar verification characteristics. Both systems must moderate between completely independent participants. There is a strong emphasis on the fact that there is no central controlling entity in the system. In the case of Bitcoin, no central financial institution is controlling the cryptocurrency. Likewise, workflows which should be enacted as choreographies must not be controlled by a single company.

Furthermore, both technologies control very sensitive data, making the security measures of these systems critical features. The transfer of a Bitcoin from a payer to a payee must be permanently recorded and the record itself must be valid and indisputable. Equivalently, the handover of a workflow instance from one company to another must be permanently and undeniably documented. Process owners must then be able to validate this documentation. While there is no commonly agreed technique for choreography-oriented workflow enactment verification, Bitcoin solves this challenges through the Blockchain. To further investigate the applicability of the Blockchain in choreography-oriented WfMSs, the following research challenges need to be resolved.

RC-1 How can runtime verification be performed in choreography-oriented WfMSs by utilizing a Blockchain?

Operating a custom Blockchain is not reasonable. Instead, one of the existing operating Blockchains must be chosen for the workflow enactment verification approach. This selection must be done with great care. Each operating Blockchain is specialized for certain use cases limiting its adaptiveness. Furthermore, the quality of the Blockchain's miner base determines its security strength. Only a few different variants of the Blockchain are currently operated and accessible.

RC-2 How does Blockchain-based runtime verification compete against existing scientific proposals in terms of performance and flexibility?

The characteristics of the proposed Blockchain approach must be evaluated and compared to existing runtime verification approaches for workflow choreographies.

3.2 Evaluation Approach

In order to address *RC-1*, different variants to implement a runtime verification mechanism for WfMSs will be outlined. For the most promising approach, a prototypical implementation will be provided.

RC-2 will be addressed in two parts. After an initial description of existing scientific runtime verification proposals, a *functional comparison* between the discovered approaches and the developed prototype will be conducted. The functional comparison will analyze the flexibility of the listed approaches in respect to the requirements of choreography-oriented WfMSs and the companies that use them.

In particular, the comparison will assess the suitability of the select proposals to handle the unique characteristics of distributed B2B cooperation. Therefore, the term *flexibility* is used to measure the capacity of a software system to deal with the dynamic nature of workflow choreographies. A flexible runtime verification technique can be utilized for different choreography-oriented WfMS prototypes and their use cases.

The second part of *RC-2* will be addressed through a *performance analysis* of the proposed prototype from *RC-1*. This evaluation is carried out similarly to the overhead analysis described by [70]. During different workflow executions, which include the proposed runtime verification, the runtime overhead and the transaction overhead (i.e., costs in terms of the number of additional transactions necessary) will be recorded. These executions are then compared to verification-less executions as baselines.

CHAPTER 4

Motivational Scenario

As described in Section 2.2, there is no commonly agreed implementation approach for workflow choreographies. Therefore, this section highlights possible choreography scenarios and assumptions which will be used as reference for further analysis in this thesis.

In order to address ever-changing market environments, companies require access to B2B cooperation. It must be possible to define business processes on demand and have them executed as choreographies by a pool of independent cooperation partners. In comparison to centralized orchestration-oriented WfMSs, choreography-oriented WfMSs operate within a distributed system. Therefore, these orchestration-oriented WfMSs must be able to address the highly dynamic nature of these systems [71, 98].

At the same time, the shared workflows must be enacted in a controlled fashion. The participating companies will be reluctant to share information about their identity, data or internal business structure [13, 89]. On the other hand, the process owner requires information about their enacted distributed workflows. They need to know which activities have been fulfilled by which partners and how long the enactment took [68, 91]. As the definition of workflow choreography describes, this control over the cooperation should, if possible, not be centralized. A centralized monitoring facility must be trusted by all choreography participants.

While scientific contributions agree that choreography-oriented WfMSs must exhibit a high degree of flexibility in order to deal with the dynamic nature of distributed B2B cooperation, basic assumptions about the characteristics of these cooperation differ.

One aspect which is often not explicitly discussed by scientific contributions is the selection mechanism for choreography participants. This mechanism has major impact on the overall system. Many contributions assume that the participants of a choreography are selected prior to the actual enactment and do not change during the course of the enactment [68, 91–93]. Alternatively the participants can be selected on-demand during the enactment [19, 92]. Pre-selecting choreography participants certainly reduces complexity during the actual enactment. On the other hand, participants that can be selected or changed on-demand increase flexibility during the enactment. Beside traceability, fault management becomes an important and complex topic for distributed choreographies [39, 58, 68].

Other characteristics in choreography-oriented scenarios seem to be commonly agreed on. For example, there always exists one process owner which initiates a business process and is paying for its successful distributed execution [71]. The different functional and non-functional execution constraints and monetary reward of an activity are predefined by the process owner, in many cases described as SLAs. This bundled workflow information is shared among the different cooperation partners. Activities can potentially be enacted by many different services. Some may be located directly at the process owner, others may be located remotely at one of the cooperating partners.

The challenge of this thesis and its proposed prototype is to ensure that these described choreography scenarios can be executed in a well-documented fashion. The handover of the control of a workflow instance must be documented in an undeniable way. This documentation must be accessible for the process owner.

Furthermore, certain assumtions are made. It is assumed that, the process owner initially hands over the enactment of the workflow to a suitable partner in order to have a specific task of the workflow executed. To accomplish this, the process owner first selects the next suitable choreography participant. This participants is either pre-defined or chosen based on the required service and the defined SLA. When select on-demand, the process owner and the selected potential cooperation partner negotiate the terms of the handover. This described search and negotiation steps are well covered and researched in the scientific field of ***S****ervice* ***O****riented* ***A****rchitecture* (SOA) [79]. After the cooperation partner has finished the execution of the defined task, the control over the workflow execution is passed along to another potential choreography participant. This is done by employing the previously described search and negotiation steps.

As described in Section 2.1.4, two characteristic variants of choreographies are discussed in the research community. In some discussions, choreography enactment is placed at the service level [16, 35, 92]. A workflow instance is directly passed along between the different services that enact it. The routing decision is also made directly at the service. It is not relevant who operates the services.

Other contributions outline the enactment of a choreography as *distributed orchestration*. Workflow instances are passed between choreography partners (i.e. companies) instead of services [36, 93]. Each partner enacts one or more steps of the choreography instance before passing it on to other partners. The internal execution of a workflow instance at a single participant has then to be managed by a local centralized orchestration engine. It provides the common tasks of a WfMS like mapping the tasks of the instance to services, scheduling executions and allocating resources for services. This described choreography setup is illustrated in Figure 4.1.

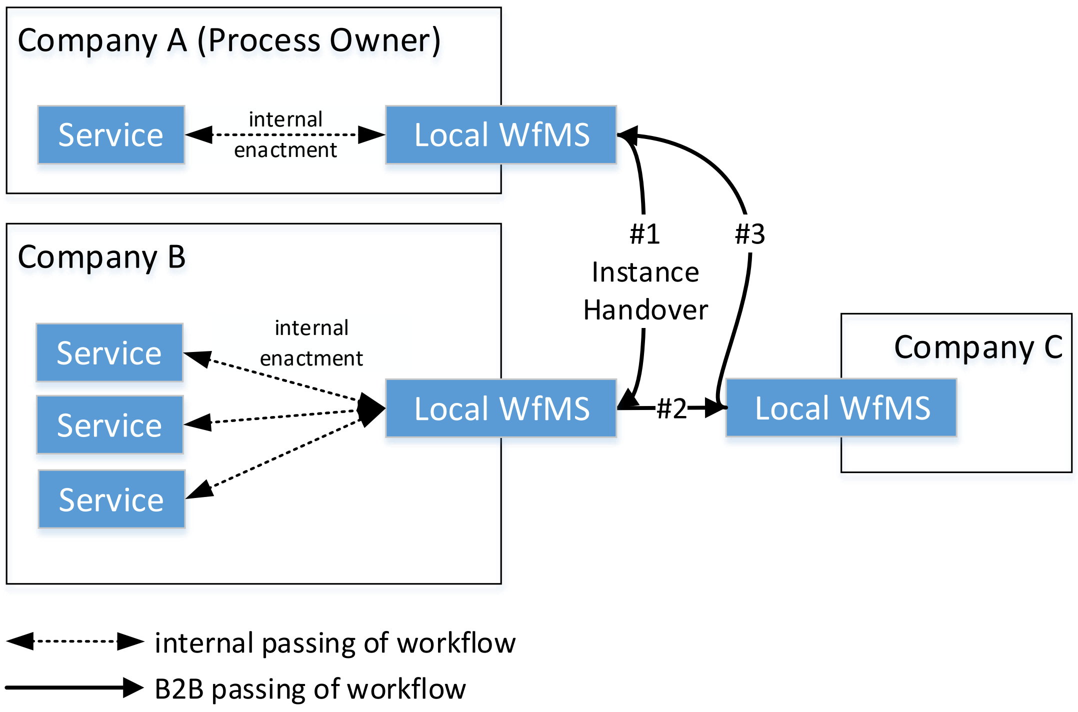


Figure 4.1: Choreography Setup

Since the progress documentation of a choreography is especially relevant when control is passed over to other companies, the described choreography scenario of this thesis will be outlined as cooperation between companies instead of services.

It has to be emphasized that all participants remain independent organizations and also potential competitors [89]. That is why one goal for runtime verification is to keep mutual dependencies to a minimum. Companies receive an incoming call for a workflow instance with all necessary execution data and work on it. After completion, they forward the instance to the next choreography partner. The handover of a workflow instance to another company together with the achieved progress must be documented. No centralized invasive monitoring service can be used. Such monitoring would introduce a tighter coupling and information sharing between the participants of a choreography, which is not desired. Any centralized authority must be avoided.

CHAPTER 5

Runtime Verification for Choreographies

As described in *RC-1*, the Blockchain technology offers a promising basis to implement independent, distributed and undeniable runtime verification for workflows. This section analyzes this approach in detail. In an initial step, a short introduction into the current three major operational Blockchain implementations *Bitcoin*, *Litecoin* and *Ethereum* is given in Section 5.1. Next, Section 5.2 describes existing approaches and implementations of runtime verification. Based on the findings of these two sections, the Blockchain-based runtime verification approach proposed by this thesis will be defined in Section 5.3. At last, Section 5.4 gives an overview about the prototypical implementation of the proposed runtime verification framework.

5.1 Current Blockchain Implementations

This section provides an overview about the most important current Blockchain implementations. The Blockchains are discussed in regard of their features and their quality. Blockchains can be categorized into first generation and second generation Blockchains [91]. The first operating Blockchains were all designed to serve a single main purpose and have limited adaptability for other use cases. They are referred to as first generation Blockchains. Representatives of first generation Blockchains are Bitcoin and Litecoin.

Recent implementations of Blockchains do not intentionally limit the use cases that can be addressed by them. By providing an open programming environment in their Blockchain, they aim to support and facilitate various use cases. These type of Blockchains are referred to as second generation Blockchains. The Ethereum project is a representative of second generation Blockchains.

A Blockchain's goal always is to serve as a shared distributed trust basis. When a Blockchain is selected for a certain project, not only its offered features but also its exhibited quality should be considered. This important characteristic is often not taken into account. Even though it can be a challenging task, the technical foundation and the supporting group of miners of a Blockchain should be analyzed, at least in a general fashion.

Synchronization decisions in a Blockchain are not achieved through classic per capita voting but through computing power voting. As long as no single entity controls more than 50% of the networks computing power, no single entity is able to execute malicious actions by rewriting contents of the Blockchain (i.e. rewriting the transaction history) [85]. Therefore, it is important that a Blockchain is supported by a large miner base. The more miners contribute their computing power, the less likely a single miner is able to accumulate enough computing power on its own to carry out a 50% attack. This is especially critical for newly founded Blockchains which do not have a large miner base yet.

Two other important characteristics of a miner base are decentralization and geographic distribution. In order to prevent a single entity from taking over the network, miners should operate independently and decentralized. On the other hand, miners may form mining pools in order to share revenues and to reduce their individual financial risk. At the same time these mining pools erase the independence between the participating miners. To the rest of the network, a mining pool becomes a new single mining entity, thus reducing the decentralization of the network [99].

At last, the operating miners of a Blockchain should be geographically distributed. This ensures that a Blockchain can be used globally with similar network delay. The distribution also ensures that miners operate from different nations and therefore different jurisdictions. This further emphasizes the independence of a Blockchain.

Based on the estimated market capitalizations of cryptocurrencies from [30], dating from the 28th of august 2016, the top three operating Blockchains were selected for a short analysis. These three Blockchains had the highest market capitalization at that date. Note that while the Ripple project ranks on the third place, it is not a cryptocurrency that relies on a Blockchain and is therefore omitted. In addition to these three Blockchains, the Counterparty project is also briefly explained [33]. This project does not maintain its own Blockchain but operates on top of the Bitcoin Blockchain. Therefore, it is worth mentioning in this context.

5.1.1 Bitcoin

The first and oldest Blockchain was established for Bitcoin. It is a first generation Blockchain with the purpose to support the exchange of digital currency. The technical foundation of Bitcoin is described in Section 2.3. Bitcoin and its Blockchain is still the most popular project among cryptocurrencies. It achieves the highest market capitalization and outranks the Ethereum project on the second place by a factor of about nine.

At the same time, this has drawn a lot of attention to the project and increased its user base. While its cryptography foundation remained solid and uncorrupted to the present day, Bitcoin currently suffers from major scalability problems. As described in Section 2.3.4, the transaction throughput of the network is not high enough. Reducing the block creation time or incrementing the block size both increase the threat of conflicting blocks occurring in the network.

Since Bitcoin operates a cryptocurrency which is not backed by banks or nations, its value is solemnly dictated by the trust of users into the security of its technical foundation. That is why new conflict resolution proposals, like GHOST [82], have not yet been integrated in order to lower the block creation time. Instead the community is trending towards a very conservative increase of the size of Bitcoin's blocks.

Because Bitcoin was the first cryptocurrency, it attracted a lot of attention from researchers around the world. Therefore, various evaluations about its network and miner base have already been published. A quantitative and qualitative analysis of the Bitcoin P2P network was conducted in 2014 by [34]. During 37 days of monitoring they identified more than 872,000 Bitcoin nodes homogeneously spread across the world, making the Bitcoin overlay a global network. The majority of all active nodes where discovered in United States and China. Together the nodes found in these countries made up 37% of all nodes.

Another evaluated feature was the propagation time of published transactions and blocks. For this purpose more than 13,000 nodes where specifically monitored in greater detail. Some discrepancies were revealed between the propagation time of blocks and the propagation time of transactions. 50% of all blocks on average needed less than 22 seconds to be propagated to 25% of all the monitored nodes. At the same time 50% of all transactions needed up to 17 minutes to be propagated to the same amount of nodes.

As in every open P2P network the overall activity and reliability of the participating nodes varies a lot. The authors of [34] observed that approximately 6,000 nodes form the reliable core of the network. This reliability is also reflected in the propagation activity of the nodes. Only 20 nodes from the pool of 13,000 specifically monitored nodes serve as the first relay hop for more than 70% of all propagated blocks and transactions.

The evaluation of [97] two years later shows only a slightly changed picture. [97] operates an agent that contiguously crawls the Bitcoin network and publishes the results on a web page. It is important to note that, [97] and [34] employ two different monitoring techniques. While [34] conducted a detailed scientific examination of the network over a short timespan, [97] operates a crawler which continuously crawls the network in a general manner. The distribution of nodes to countries discovered by [97] on the 15th of September 2016 resulted in the following top four nations.

1. United States (27.53%)
2. Germany (16.70%)

3. France (7.71%)
4. Netherlands (5.63%)

As already mentioned, Bitcoin operates on top of a first generation Blockchain. Still it is possible to adapt Bitcoin transactions for other use cases to a certain degree. Since Bitcoin resulted in the first implementation of a Blockchain, various possibilities for alternative usages where initially incorporated in the design. Section 5.1.4 gives an example of an alternative use of the Blockchain. Also the Sections 5.3 and 5.4.1 describe in detail how an alternative use can be achieved.

5.1.2 Litecoin

The Litecoin project was started in 2011 as a payment alternative to Bitcoin [56]. It is fully based on the technology stack of Bitcoin and added just a few changes. Therefore, the project also operates a first generation Blockchain. Despite having a market capitalization of the size of only 2% of Bitcoin's market capitalization, it is still the third biggest operating Blockchain in terms of market worth.

The first change incorporated in Litecoin is a different proof of work hashing method. As already described in Section 2.3, Bitcoin mining has become very professional. Through the usage of specialized and expensive hardware, mining became unaffordable for private Bitcoin miners. Through the use of the scrypt hashing method, Litecoin aimed to reduce the effectiveness of specialized hardware. Unfortunately they were only successful to a certain extend [99].

As a second change, [56] reduced the block creation time to about 2.5 minutes. This reduction of the creation time is achieved by reducing the proof of work complexity. This reduction results in an increased rate of conflicting blocks and therefore an increased number of orphaned blocks. This alone might not be a problem. But other scientific evaluations point out that the required 50% computing power threshold to carry out a double spending attack is successively lowered when the network fails to fully synchronize in between the block creation intervals [31, 41, 85].

At last, no scientific evaluations about the P2P network of Litecoin could be found. To the best of our knowledge there currently exists no scientific estimation about the size, distribution and quality of Litecoin's mining pool. The only crude estimate about Litecoin's miner base is its market capitalization. A cryptocurrency that achieves a higher capitalization and therefore a higher potential to earn money as a miner, will attract a larger miner base. This general estimation metric was also proposed by [42].

5.1.3 Ethereum

The Ethereum project and its Blockchain was launched in 2015. It implements a second generation Blockchain. Instead of a singular purpose, Ethereum enables the use of so called *smart contracts* [73]. These smart contracts are program agents that are executed within the Ethereum environment. They are defined in a Turing complete scripting language and enriched with private storage and monetary balance. Through these features smart contracts can operate completely autonomous. The Ethereum project received a lot of attention in recent months and is currently the fastest growing Blockchain implementation. It has the second highest market capitalization of all operating Blockchains. Still the project reaches only about a 9^{th} of Bitcoin's market capitalization.

The project also defines its own currency to fuel its operations, called *Ether*. This currency can either be used directly for payment transactions or to fuel the execution of smart contracts. It is possible to define contracts in the scripting languages *Solidity* and *Serpent* and have them executed by all Ethereum mining nodes. Besides mining, these nodes operate a so called *Ethereum Virtual Machine* (EVM) which is a runtime environment for the smart contracts. Once submitted, a contract is able to send and receive messages. The exchange of these messages is performed through the transactions which are submitted to the Blockchain. The code of a contract is executed each time it receives a message. To compensate for this execution, each message defines an amount of Ether, so called *gas money*, to pay the nodes that run the contract inside their EVMs.

During the execution of its code, a contract is able to interact with its storage, send messages or even create other contracts. The private state of a contract is implemented as a key-value store. Since smart contracts are only executed upon receiving messages, the state is propagated through the exchanged messages. Beside new transactions, newly mined blocks include also the new current state. While this seems impractical, the implemented nodes only store one instance of the state and include references to the relevant parts inside the blocks. Since each block is likely to only change a small part of the state, this information does not take up much space.

The Ethereum project implements a very low Block Creation time of 15 second. To address the resulting increase in conflicting blocks, a simplyfied version of the GHOST [82] conflict resolution proposal is used [38].

While the proposal of [82] holds a lot of potential, the Ethereum project is its first integration in a major Blockchain implementation. There has yet to be a thorough investigation if the GHOST approach is able to handle a block creation time as low as 15 seconds. To the best of our knowledge it has not yet been evaluated if the ghost approach is effectively able to address synchronization conflicts in this frequency. Since the creation time is so low, the network most likely will not be able to synchronize in time. If not properly addressed, this situation could result in lowering the 50% threshold for double spending attacks [31, 41, 85].

Ethereum's mining network has not yet been throughtly analysed [6]. The evaluation of [6] revealed that up until now, only 15% of all documented transactions targeted smart contracts. The major portion of all transactions still consists of normal payment transactions. While being advertised as second level Blockchain, it seems that Ethereum is still mainly regarded as another cryptocurrency. A general scan of the network showed that the network of Ethereum is global with a strong tendency towards Northern America, Russia and China. Unfortunately [6] provide no estimates about the number of active miners.

5.1.4 Counterparty

The goal of the Counterparty project is to offer second generation Blockchain capabilities while operating on top of the Bitcoin Blockchain [33]. An extensive meta-framework has been developed which extends the main Bitcoin software[1] with additional components. Standard Bitcoin transactions are used to store miscellaneous data in the Bitcoin Blockchain. While this data does not have special meaning to classic Bitcoin nodes, specialized Counterparty nodes are able to interpret and execute the stored data, if required.

In order to regulate the usage of its various features, Counterparty defines its own currency, called *XCP*. Like the framework itself, XCP operates as a meta-currency on top of BTC. In order to obtain XCP, one has to *burn* Bitcoin. This is done by sending Bitcoins to specific fake Counterparty addresses. Since the address itself is fake, the coins that have been sent there can never be forwarded again and become unusable. At the same time, specialized Counterparty nodes in the Bitcoin network monitor these kind of transactions and issue XCP to the senders of these burn-transactions. While this seems highly wasteful and unnecessary, XCP are used to regulate the usage of Counterparty features, i.e. to avoid spam. Since small amounts of XCP are required to use most features of Counterparty, users have to demonstrate their dedication to the project by burning Bitcoins and spending XCP. Already created XCP can also be bought at exchange platforms.

While XCP are the native tokens of the Counterparty project, one of the first and simplest features offered by the project is the creation of custom tokens. By defining a dedicated token name and converting a certain amount of XCP into the newly created token, users and developers can use these tokens to control and fuel their applications. This enables various use cases, like voting, betting or access control.

The second and most advertised feature is the usage of Turing complete smart contracts. Instead of defining their own smart contract language and execution environment, the Counterparty project adopted the smart contract techniques of Ethereum. Specifically they support the scripting languages Solidity and Serpent. Therefore, contracts can be defined in the same way as Ethereum contracts and migration between the frameworks should be possible with only minor adoptions. Counterparty smart contracts are fuelled by XCP, and executed by the specialized Counterparty nodes. As compensations for these execution, XCPs are collected as fees. The smart contract feature of Counterparty has not yet been declared ready for productive use. Currently it is only possible to publish and use smart contracts in the Bitcoin testnet. Another downside are the required specialized Counterparty nodes. While the miner base of Bitcoin has been well evaluated,

[1] `https://github.com/bitcoin/bitcoin/`

the supporting node base of Counterparty has to be evaluated additionally. This node base must exhibit the same quality features as a classic Blockchain miner base.

The last advertised feature of Counterparty is the *lightning payment framework*. This payment framework addresses the problem of the long block creation time in Bitcoin. The resulting long transaction verification time represents a problem for many real world payment scenarios. By providing a side channel to the Blockchain, the lighting payment framework aims to speed up payment transactions, especially micropayments. The basic concept of such side channels relies on multi-signature transactions to establish shared wallets and sharing off-chain refund transaction with lock-times. As explained in Section 5.4.1, Bitcoin includes a transaction type which requires the signature of two or more parties to access funds. A transaction can have a lock-time associated upon creation. This lock-time defines a future date. Until this date is reached, the transaction is defined as invalid and can not be included in the Blockchain.

Assume that two parties, Alice and Bob, want to quickly exchange a series of micropayments. Bob wants to repeatedly send Alice money. First a wallet that is shared between Alice and Bob has to be created. To achieve this, Bob creates a transaction that locks a certain amount of his funds in an output that can only be accessed by providing a signature of both parties. Before Bob publishes this transaction, Alice creates and shares a refund transaction which returns all funds to Bob but is locked for n hours. Then Bob publishes the initially created wallet transactions. Alice and Bob now have a shared wallet to perform micropayments. If Alice stops responding, Bob can recollect his funds after n hours by using the refund transaction. In order to pay Alice, Bob simply has to create and share a new refund transaction which splits the contents of the wallet between Alice and Bob and has a lock-time of $n - 1$ hours. The refund transactions can be exchanged off-chain and directly between Alice and Bob. If Bob wants to perform another payment, he simply has to resent a new refund transaction with adjusted balances and a lock-time of $n - 2$ hours.

Similar to the smart contract feature, the lightning payment feature has not yet been completed and is therefore not yet available.

5.2 Existing Enactment Verification Approaches

5.2.1 Introduction

In choreographies process owners hand over the control over their workflow instances to remote partners. Depending on the design and flexibility of the system or the workflow, the process owner might not know which companies are participating or which activities they cover. In such systems process owners can only trust to receive the result of their workflow at some point in the future.

As described in [71], contracts are always the basis of a choreography or any other kind of B2B cooperation. They specify how cooperating companies are rewarded and how they are penalized. Contracts form the trust basis of choreographies. In order for such contracts to be enforceable, the distributed enactment of processes must be provable. Process owners have to be able to determine which company covered a task and how long this task took [62].

WfMSs collect information about their subsystems by using monitoring frameworks. The CEP approach can be utilized to implement monitoring features for centralized WfMSs and B2B WfMSs alike. Section 2.2 explained monitoring approaches in B2B choreographies. These enable a process owner to collect information about the status of remotely enacted workflow instances. While being suited for collecting information and calculating KPIs, most event-based architectures do not incorporate end-to-end security features [28].

Section 2.2 discussed CEP-based montoring as a theoretical solution approach for B2B workflow runtime verification. It is possible to enhance event messages with security measures. They can be enriched with identification information and their integrity as well as their privacy can be protected through cryptography [12]. CEP systems are expected to handle a vast amount of events. Adding security features to those events greatly increases the complexity of such systems. Furthermore, each CEP system operates under the control of one of the choreography's participants and can not be regarded as a trusted system for others. Malicious participants can still generate messages for events which did not take place. In addition, events are commonly broadcast asynchronously. Messages may be lost or intercepted. Without a centralized monitor component there is no shared agreement on which events did occur and which did not. CEP-oriented monitoring systems therefore are not suited to serve as trusted runtime verification systems.

This requires choreography-oriented WfMS to implement additional security mechanisms to ensure end-to-end integrity, authenticity and non-repudiation for workflow instances [54, 68].

The remainder of this section is organized as follows: Section 5.2.2 provides an overview over the different solution approaches from the scientific field of workflow runtime verification. The integration of verification mechanisms in actual implementations of choreography-oriented WfMS is discussed in Section 5.2.3. Runtime verification solutions based on Blockchains are described in Section 5.2.4.

5.2.2 Solution Approaches

The verification of workflow enactment does not present a problem in centralized WfMSs [54]. How this challenge can be solved in a choreography-oriented WfMS depends on the system's design. The more flexible a system is designed, the more difficult enactment verification becomes. Unfortunately, making a system more static is no solution. While verification becomes a lot easier in static choreographies, the overall robustness of the distributed WfMS is greatly reduced [19].

In a federation of independent partners, one cannot rely upon constant availability of all services. The used services or involved partners in a choreography are therefore not static. The same is true for the execution time of specific service calls. Some service execution times are predictable. Others operate on “best effort” basis and can only provide worst-case predictions of their execution time.

Process owners require the possibility to track the execution of an instance. Existing research addressing this challenges appears to be limited. “There seems to be a relatively small amount of work that examines basic security issues of workflow systems, particularly in terms of authenticity and integrity protection of workflow information and sequence” [60]. There seem to be two general approaches to provide enactment verification in choreographies. The first approach aims to control the message flow between the participants.

[89] propose the usage of multiple *Enterprise Service Buses* (ESBs) to handle all communication between the cooperating companies. In their work they suggest the usage of Web Service proxies which are supposed to intercept all communication. These proxies log all necessary information to a central logging component. Through metadata, which provide time and correlation information, the central logging component is able to sort and associate the information.

A similar approach is described by [13]. In their scenario, the cooperating participants of a choreography are already chosen at deployment time by the process owner. The initiating company is then able to generate choreography-specific messaging policies for each participant. Those policies individually specify accepted message patterns. In order to guarantee the enforcement of given policies, all participants must run the same communication gateways which intercepts all traffic. If deviations are observed, CEP fashioned events are emitted to notify the process owner.

The second general approach proposes token passing along the participants of the choreography. By enhancing it with cryptography features, the token becomes a proof for the path it travelled along. Through keeping a copy, each company can proof its participation in the corresponding instance. Upon receiving the corresponding tokens, process owners are able to verify the exact sequence of execution. Depending on the structure of a choreography’s required data sources, this technique can also be used to ensure data integrity and confidentiality. If the required data can be sent along the choreography as a single document, the document becomes the token upon which the security features are applied.

An early and simple variant is proposed by [19]. Through the usage of simple cryptographic signatures, the participants can integrate and sign information about their contribution into the choreography token. Upon receiving a token each participant performs the required service on the document and then appends information about their contribution to it. To enable process owners to interpret the contributions, [19] requires them to be formatted in a defined XML schema.

[54] further develop this approach by combining it with *Trusted Platform Modules* (TPMs). These hardware components provide the trusted platform the distributed network operates on. Each module contains a private key, is able to perform cryptographic operations

and is tamper-proof. As a security token a so called *process slip structure* is used. This structure contains the data relevant for the choreography. It is protected by cryptography measures. In order to access the required data, services have to use their associated TPMs. They will only return the data if its associated service fits the execution plan. After a service has been enacted, the slip structure is updated and forwarded to the next service call.

The two previously described approaches provide means to prove that a participant possessed the security token at one point during the enactment. [68] further enhance this approach with enforcement of sequence mechanics and privacy for participants. Their security token is called an *onion*, which is made up of several layers of encryption.

At first the process owner has to select the desired participants prior to the deployment of the instance. Then the security token is generated by layering encrypted information. Each layer can only be accessed by the destined participating partner of the choreography. Upon receiving, a service "peels" of the topmost layer of encryption. Inside this layer reside the necessary data required to enact the current process step. Upon completion the result is appended to the onion and the whole token is forwarded to the next participants. A business partner can be sure that the previous steps have been fulfilled correctly when it is possible to decrypt the top layer of the onion.

Through a sophisticated encryption key management system, each choreography uses unique keys for all layers. The process owner distributes the keys after the onion has been created. Through this technique, the privacy between the partners is preserved. Only the process owner maintains a global picture of the choreography. The design of the onion approach is clearly very static, as the participants have to be selected in advance and cannot change. Therefore [68] also include a recovery mechanism. Intermediate results are stored at the partners and used during re-enactment after a failure.

Inspired by this, [60] propose an alternative approach which promises more flexibility. By adapting the concept of hierarchical identity-based signatures they create *workflow signatures*. These signatures further enhance the functionality of traditional public key infrastructure (PKI) based signatures. Instead of key pairs, identity-based signatures are generated from a key-identity pair. A key, referred to as *private key*, is used for encryption while a publicly available String identifier is used for decryption.

The core enhancement of the concept is the fact that identity-based signatures can also be generated by using combinations of more than one private key. This enables the signatures to serve as evidence for the sequence of the fulfilled tasks as well as to reflect the logical paths of the workflow. Each task of a workflow instance is associated with an hierarchical identifier composed from the task-ids of the already taken path and the current task-id.

Upon forwarding a workflow instance to a cooperating partner, a company is able to determine the identifier for the next step and compute the associated private key to it. Upon merging a parallel execution, a participant simply has to wait until all preceding paths are finished and the corresponding private keys have been received. The signature of all private keys combined then serves as proof that all required prior paths have been completed.

5.2.3 Verification in Scientific WfMS Prototypes

The development of choreography-enabled WfMS is still in its infancy [2, 58]. There are no mature frameworks which support the enactment of choreographies. Different scientific proposals have been made on how such WfMS could be designed. Only few of those proposals cover security-related aspects let alone enactment verification. The following WfMS proposals incorporate security aspects related to the verification of workflow instance enactment.

[46] propose a distributed B2B WfMS which is optimized to run in clouds. The B2B and cloud aspects led to the incorporation of various security features to ensure authentication, confidentiality, data integrity, and non-repudiation. They employ a token-based verification approach. A XML document, including the relevant workflow data is passed along the choreography. The document is heavily secured with element-wise encryption, timestamps and digital signatures. To increase reliability, a distributed cloud storage is used to pool all active tokens. Through the limited but sufficient access regulations of the storage, the enactment of the workflow instance is further secured.

The coordination through distributed storages is also employed in the work of [64]. They propose the use of *Linda-based Tuplespaces* to coordinate the participants of a choreography. “A space can be seen as a synchronized container shared among all participants in a workflow [...]” [64]. The variables in those containers are accessed through templates. These templates outline the necessary conditions for different tasks. Each participant specifies a custom template. As soon as all conditions are met, a participant starts its execution. Through these distributed containers the enactment of each workflow instance can be observed and verified. Unfortunately the authors did not evaluate the scalability of the synchronization tasks required for the distributed containers.

There exist many other scientific WfMSs prototypes which are able to enact choreographies of different forms and variants, e.g. BPELCube [74], SwinDeW-C [61], Jadex WfMS [49], OSIRIS-SR [83] or MonALISA [57]. These WfMSs are not described in this section because they do not explicitly address security related topics.

5.2.4 Blockchain-based Verification Prototypes

Message controlling and token passing are two approaches for controlling and propagating the execution state of a choreography. Both impose different challenges when being applied to the motivational scenario of this thesis.

If loose coupling is a priority in the cooperation environments, the runtime verification approach of message controlling becomes difficult. Different communication frameworks, like message buses, can ensure that choreography messages passed between the participants are only exchanged by using the provided connectors. But in order to set up a such communication framework between different companies, a tight integration between their software systems becomes necessary.

Also the token-based approach introduces new problems. The found existing runtime verification implementations for WfMSs [46, 64] all follow this approach. Still they suffer from the major problem that the controlling token, which is passed along in the distributed system, might get lost. The described prototypes therefore save the tokens, which controls access to the choreography, in a shared storage. [46] uses a distributed cloud storage to save XML files which serve as access tokens. [64] describes the usage of *Linda-based Tuplespaces* which control access and provide navigation decisions. The tuplespace is describe as a shared container, which is synchronized among the participants. This shared storage then again becomes the controlling entity for the system. The shared storage must be operated by another party and be trusted by the choreography's participants.

New scientific proposals aim to solve the problems of these approaches by using a Blockchain as the trusted entity for the choreography. Through its design, the Blockchain can provide a shared trust basis which is not under the control of a single organization. Messages can be exchanged directly through Blockchain transactions and token information can be stored in the Blockchain by embedding them in transactions. The remainder of this subsection describes two different proposals which utilize the Blockchain for documentation, message passing or controlling purposes.

The simplest security application of the Blockchain is the public documentation of timestamp hashes. While this feature alone is not enough to serve as a token-based runtime verification system for choreographies, it can provide an important basis. The control information of a choreography is shared publicly. Its changes and progress can be documented and proven with timestamp hashes, placed in the Blockchain. [42] describe such a feature in their work. Since not all Blockchains are designed to freely allow the storing of arbitrary data the core invention presented by the paper is how the data stored in the Bitcoin Blockchain.

Over the course of 24 hours, various submitted timestamp hashes are collected and hashed again, resulting in one aggregated hash. This aggregated hash is then used to create a Bitcoin private key. By using the corresponding Bitcoin address of this private key, a new transaction is published. In this transaction the smallest possible Bitcoin amount (0.00000001 BTC) is transferred from the given generated Bitcoin address to another arbitrary address. This way the Bitcoin address is stored in the Blockchain and becomes publicly available. By publishing the submitted hashes, together with their aggregated hash in a separate medium, like Twitter, the correctness of the resulting Bitcoin address can be validated. [42] offer this simple and, in terms of Bitcoin transactions, cheap feature as a public service via their website [2].

An advanced and in-depth proposal is made in the work of [91]. By utilizing so called *smart*

[2] `http://www.originstamp.org/`

contracts [73], enabled in the Ethereum Blockchain, they provide a runtime verification solution for choreographies.

Unlike the Blockchain of Bitcoin which mostly focuses on financial transactions, Ethereum encourages developers to utilize their Blockchain for any number of use cases. In order to facilitate this, the Ethereum scripting language is very flexible and Turing complete. This enables the management of complex and advanced smart contracts in Ethereum's Blockchain. In addition, each contract has access to a private key-value based storage space. Further details about this Blockchain are provided in Section 5.1.

The first contribution of [91] is a *translator* component which is able to convert BPMN models into smart contracts, described in Ethereum's scripting language. This initial *factory contracts* provide an abstract definition of the BPMN model's corresponding workflow. When a workflow needs to be enacted, these factory contracts are able to create dedicated smart contracts for each instance.

The instance contracts become the controlling entities of their corresponding workflow instances. During the creation of an instance contract, the public keys of the choreography's participants together with their corresponding roles must be provided. Enriched with this information, the instance contract is able to control and document all execution steps (i.e. the overall execution state), taken by the participants.

The participants do not communicate directly with each other. Instead they interact through transactions which are submitted against an instance contract and its contract storage. These transactions alter the state of the given contract and at the same time advance the execution state of the given workflow instance. Since all transactions are validated against the contract definition, it can be ensured that only authorized participants can alter the workflow state at a given execution point. Furthermore, the execution sequence can be enforced to match the workflow definition. At last, all this information is publicly documented in the Ethereum Blockchain.

The smart contracts for workflows are proposed in a passive and an active variant. The passive variant, called *choreography monitor*, simply provides the described controlling mechanisms. The other variant, called *mediator*, is further enhanced with active features like data transformation, message sending or other simple computations. These features are embedded in the contract's definition and executed if required by all mining Ethereum nodes.

Ethereum's scripting language is still executed under a closed-world assumption. The language itself is Turing complete, but can not access remote APIs or services. Furthermore, data-intensive transactions and computations should not be placed directly on the Blockchain. Therefore, [91] propose another component, called *trigger*, which provides a bridge for smart contracts to the outside world. Triggers are essentially clients which run full Blockchain nodes. They run local programs which constantly monitor the newly created Blocks on the Blockchain. Therefore, they are also enabled to react on updates to certain smart contracts. Given a specific execution state, they may call external APIs or receive information from remote sources. These triggers then automatically update the contract based on the external information. Among other things, this component can be used to handle data intensive storages off-chain, e.g. external databases.

Since the workflow management communication is performed entirely through smart contracts, the proposed feature of [91] can be categorized as a message controlling-based approach. The instance contracts together with the Blockchain P2P network becomes sort of a communication bus for workflow controlling.

The main downside of this described approach is the fact that all participants of the choreography must be known in advance. This is required in order to include the relevant public keys and roles into an instance contract during its creation. This greatly reduces the flexibility of the overall choreography and makes the workflow execution less robust. If one of the participants is unreliable or becomes unreachable, the whole workflow instance may be stuck. On the other hand, without the public key and role information access security and execution sequence enforcement are not possible.

At last, the approach of [91] is utilizing the novel Blockchain implementation Ethereum. While these second generation Blockchains provide great programmatic freedom, they have to operate their own Blockchain environment and often also employ new synchronization and conflict resolution mechanisms in order to improve their performance. The software foundation of every Blockchain together with the number, independence and geographic distribution of its miners have great impact on the security level of the projects using it. These questions about a Blockchain's quality are further discussed in the following section.

5.3 Blockchain-based Runtime Verification Proposal

Based on the described motivational scenario from Section 4 a novel runtime verification approach is proposed by this thesis. It aims to retain the flexibility of workflow choreographies, while at the same time providing a maximum of security and verifiability.

Instead of utilizing smart contracts of second generation Blockchains like Ethereum, only transaction techniques of existing first generation Blockchains are used. This limits the possible features but enables the usage of existing well supported and high security Blockchains, like Bitcoin. As outlined in the previous section, Bitcoin currently has the largest and most distributed miner base of all operated Blockchains. On the downside, the runtime verification proposal has to address limited adaptability and scalability when using Bitcoin.

Counterparty already offers a programming framework operating on top of Bitcoin. But at the same time it requires the conversion of BTC into the specialized currency XCP. Furthermore, the advertised feature of smart contracts is currently only enabled for the Bitcoin testnet. By directly using custom Bitcoin transactions, the usage of this meta-framework can be avoided. In order to ensure the correct execution of Counterparty's specialized syntax, another pool of specialized Counterparty nodes is operated. Similarly to Litecoin, there exist to the best of our knowledge no scientific evaluation about this node base.

At last, the usage of Counterparty prohibits the usage of alternative features like the *simple payment verification* (SPV) or the *greedy publishing mode* which are described in Section 5.4.3. Therefore, the Bitcoin Blockchain is directly used. Similar to the *choreography monitor* component proposed by [91], the solution approach describes a passive component which enables access restricted documentation of the progress of a workflow.

Initially, a free Bitcoin output is selected by the process owner at the start of a new workflow instance to serve as the control token for the choreography. At the same time, the Blockchain becomes the distributed storage for the token. Whoever is currently in possession of the token is responsible for the execution of a part of the choreography. In order to enable parallelism, the token can be split and joined. Participants can document progress of the workflow and most importantly the handover to other participants by submitting new transactions which propagate the token.

Each transaction is enriched with additional metadata about the current state of the workflow. Since Bitcoin transactions are push based, a token sender gives its approval of a handover from one participant to anther by simply publishing the transaction. But also the approval of the token receiver must be documented in the transaction. Therefore a signature of the token receiver is embedded in the workflow metadata stored in the transaction.

As described in Section 2.2.4, the basis for a choreography is always a contract [71]. The transaction chain related to the token of a workflow instance provides undeniable proof about the workflow's progress. If this progress somehow violates the agreements of the contract, penalties can be claimed by the process owner from the involved participants. At the same time, it is possible for participants to prove their successful involvement in a choreography to claim their rewards.

In order to preserve the flexibility of the choreography, the participants are not predetermined but can be chosen dynamically on demand. On the downside this prevents the enforcement of a correct workflow sequence. Still, it is not possible for a single participant to forge critical documentation entries. Therefore, a process owner can always monitor the progress of a workflow instance by observing the Blockchain. If the execution of a certain workflow instance deviates from the given process model, a process owner and all other choreography participants of this instance will be able to detect and react on it.

To change a Bitcoin transaction into a documentation entry which proofs that a workflow instance has been handed over from one participant (i.e. company) to another, it must provide the following characteristics and contain the following information.

First, token handovers must be access-protected. Only the current owner of a workflow token must be able to decide where to pass on the token. Since the token is essentially an amount of Bitcoins, this kind of access protection is already a built-in feature of Bitcoin.

As described in Section 2.3, each output of a Bitcoin transaction is protected by a script which commonly requires a Bitcoin signature of the owner as parameters.

At the same time the receiver of the token must confirm that a handover, together with the included metadata, is accepted. Bitcoin transactions are by design only push-based. This means, there is no built-in requirement for a payee to agree to a transaction. If a payer decides to forward a certain amount of Bitcoins to a payee and knows about a Bitcoin address of the payee, money can be forwarded without questions. Therefore a Bitcoin signature of the token receiver, which signs the handover transaction data must also be included into the transaction.

When the handover transaction is completed, signatures of both sender and receiver must be contained. Next, the following information has to be included in the transaction to document the state of the current workflow execution path.

Workflow instance id To emphasize which instance is addressed by the transaction, the identification number of the workflow must be included.

Task id A company works on a specific tasks and then hands over the control of the workflow to another company to perform the next task. The identification number of the task which should be performed by the receiving company must be included in the transaction.

timestamp This timestamp documents the moment the current task, processed by the sending company, ends and the following task, processed by the receiving company, starts.

Workflow data hash Most workflow instances require data to operate on. This data is continuously altered by the fulfilled steps of the workflow. In order to document the current state of the workflow data before the execution of the next task, a hash must be placed in the transaction.

Receiver signature Not only the sender must confirm the handover of a workflow, also the agreement of the receiver must be documented. Therefore, the receiver must also sign the transaction template before publishing. In this transaction template, all the data described above must already be included. This way the receiver documents approval to receive control over the workflow instance under the documented conditions.

At last, identification data of sender and receiver must be exchanged. By design Bitcoin transactions are sent between Bitcoin addresses. In this proposed approach new addresses are generated for each handover. These addresses are anonymous and protect the privacy of the involved participants. Still sender and receiver must be able to mutually prove with whom they performed the handover.

Therefore, it is assumed that beside the Bitcoin infrastructure, a RSA-based *public key infrastructure* (PKI) is in place. By utilizing RSA-based signatures and certificates, an actor can prove its identity to others. When sender and receiver want to perform a handover, they first have to share the respective Bitcoin addresses they want to use. This exchange is enriched with RSA-based signatures and certificates. This way, each handover partner confirms the ownership to a given Bitcoin address before the handover takes place. By storing this received signature, a choreography participant can also prove the identity of the corresponding handover partner to the process owner, if required.

The handover process for the runtime verification approach, proposed in this thesis, consists of the following steps. Furthermore, Figure 5.1 illustrates this handover process in a sequence diagram.

#1 The first tasks are common to all choreographies. After the sending company has selected a potential receiving company they mutually identify each other and negotiate the metadata of the handover.

#2 When a consensus is reached, the sender transfers the symmetrically encrypted workflow data to the receiver. This way, the time consuming data transfer is completed before the handover but the receiver can not yet start working on the following task. On the sender side, the workflow data is hashed to prove its state during handover.

#3 Bitcoin addresses are exchanged through PKI signatures to provide a legal confirmation that the address is indeed owned by the respective partner.

#4 A transaction template is created by the sender which holds the negotiated handover terms (i.e. the required metadata to completely document the state of a workflow).

#5 The sender transmits the transaction template to the receiver together with the symmetric key to unlock the workflow data. The template is sent as an RSA-based signature. This way, the receiver already has proof that the sender intends to perform the given handover. If the transaction template contains the negotiated handover terms from step #1, the receiver approves the template by creating and returning a Bitcoin-based signature of the template. For this signature, the private key of the receiver's Bitcoin address is used. Since the receiver can now decrypt the workflow data, the execution of the next workflow task can be started.

#6 The sender validates the receiver's signature. If the signature is correct, the transaction is finalized by adding the Bitcoin-based signature of the sender. At last, the Bitcoin transaction is published by the sender. Since all Bitcoin transactions are broadcast and shared publicly, the receiver can monitor if the sender actually takes care of publishing the transaction. If the sender does not take care of publishing, the receiver needs to contact the respective mediator of the choreography (i.e. probably the process owner). The transaction template, signed by the sender, serves as proof that a workflow handover was intended by the two partners.

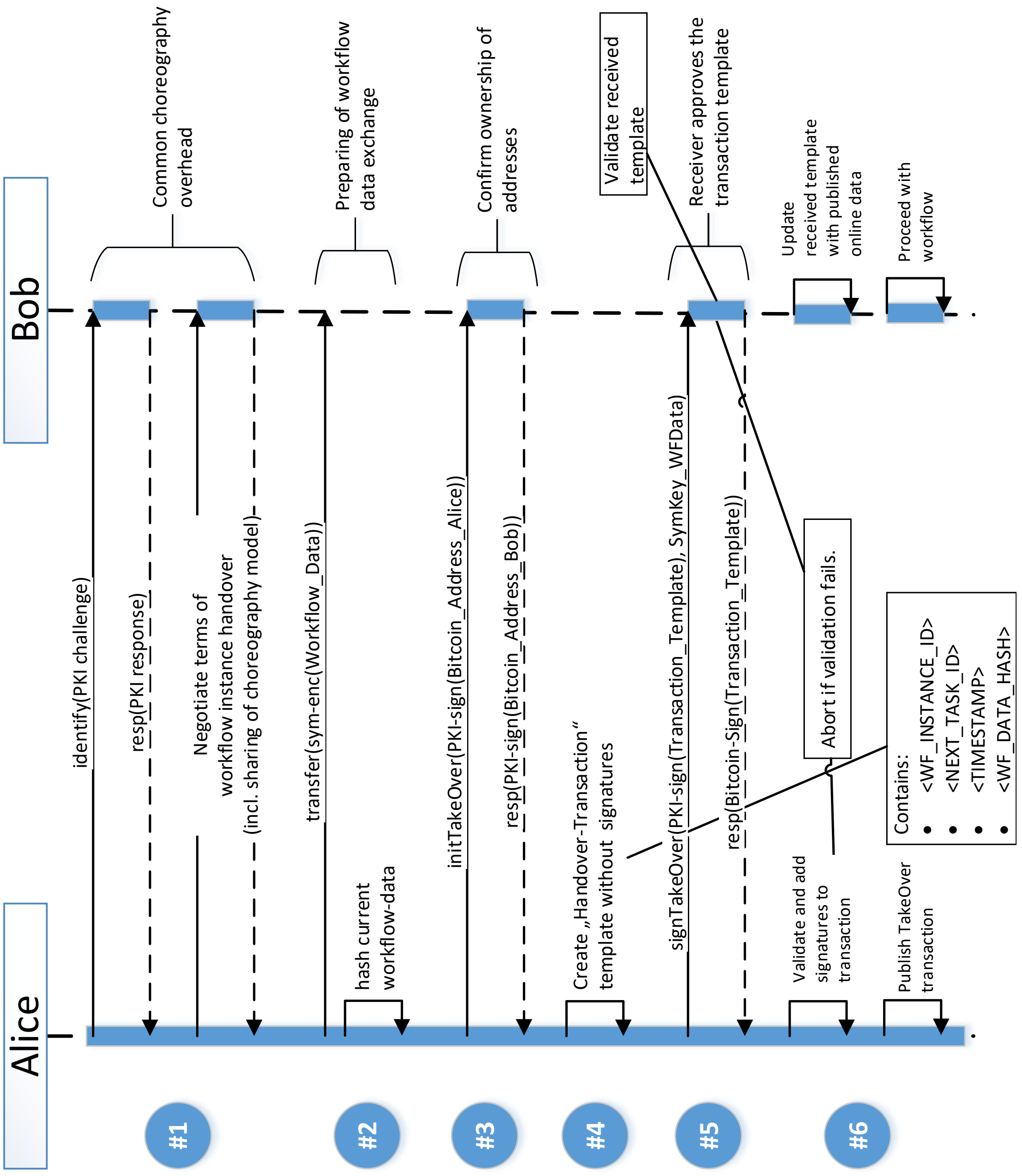

Alice
Bob
#1
#2
#3
#4
#5
#6
identify(PKI challenge)
resp(PKI response)
Negotiate terms of workflow instance handover (incl. sharing of choreography model)
transfer(sym-enc(Workflow_Data))
hash current workflow-data
initTakeOver(PKI-sign(Bitcoin_Address_Alice))
resp(PKI-sign(Bitcoin_Address_Bob))
Create „Handover-Transaction“ template without signatures
signTakeOver(PKI-sign(Transaction_Template), SymKey_WFData)
resp(Bitcoin-Sign(Transaction_Template))
Validate and add signatures to transaction
Publish TakeOver transaction
Abort if validation fails.
Contains:
<WF_INSTANCE_ID>
<NEXT_TASK_ID>
<TIMESTAMP>
<WF_DATA_HASH>
Common choreography overhead
Preparing of workflow data exchange
Confirm ownership of addresses
Validate received template
Receiver approves the transaction template
Update received template with published online data
Proceed with workflow

Figure 5.1: Intermediate Handover of a Workflow Instance between Companies

By observing the Blockchain, the process owner can monitor the progress of the workflow instance. Optionally, it is possible for the process owner to immediately collect more detailed information about the latest progress of a workflow instance. For example initially, the process owner hands over the workflow instance to the first choreography participant Alice. After completing the designated task, the Alice performs a handover with the next participant Bob. Right after the new handover transaction is published, the process owner observes the changes and can contact Alice in order to collect the identity information about Bob. In comparison to other approaches, the information collection is pull-based rather than push-based. This process is further illustrated in Figure 5.2.

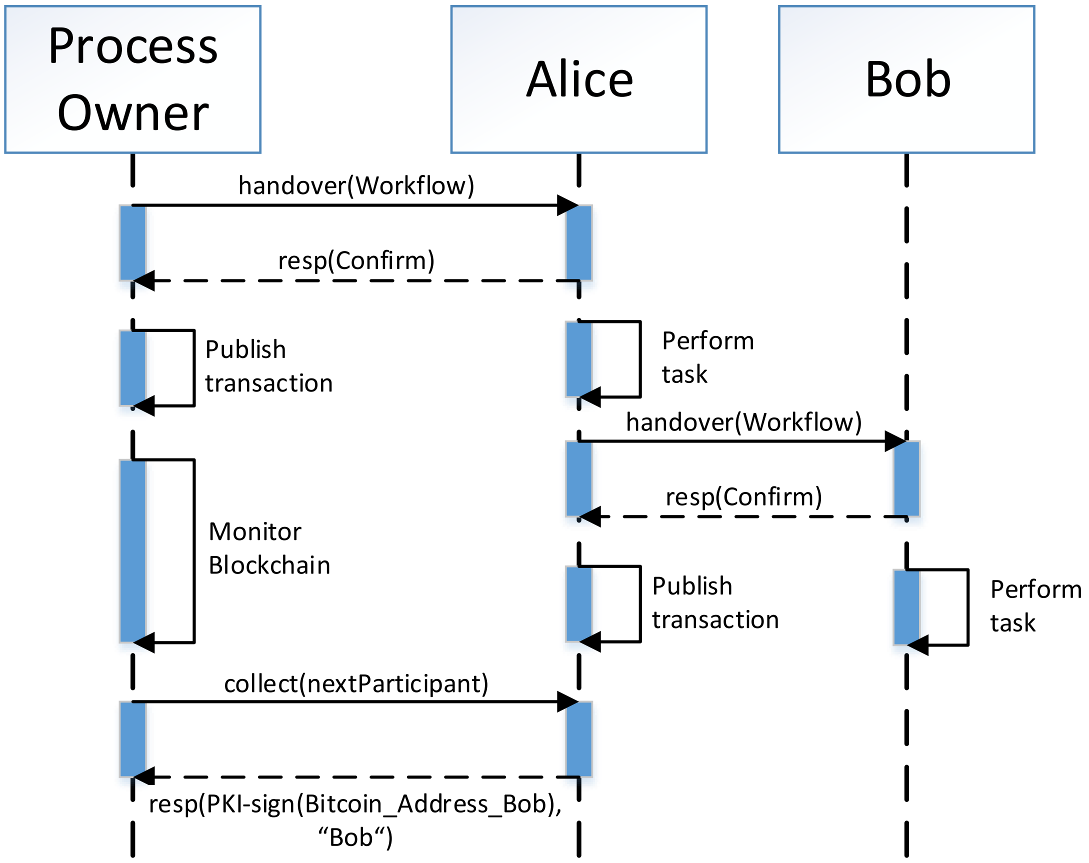


Figure 5.2: Pull based monitoring of process owner

The general structure of a workflow-handover transaction is illustrated in Figure 5.3.

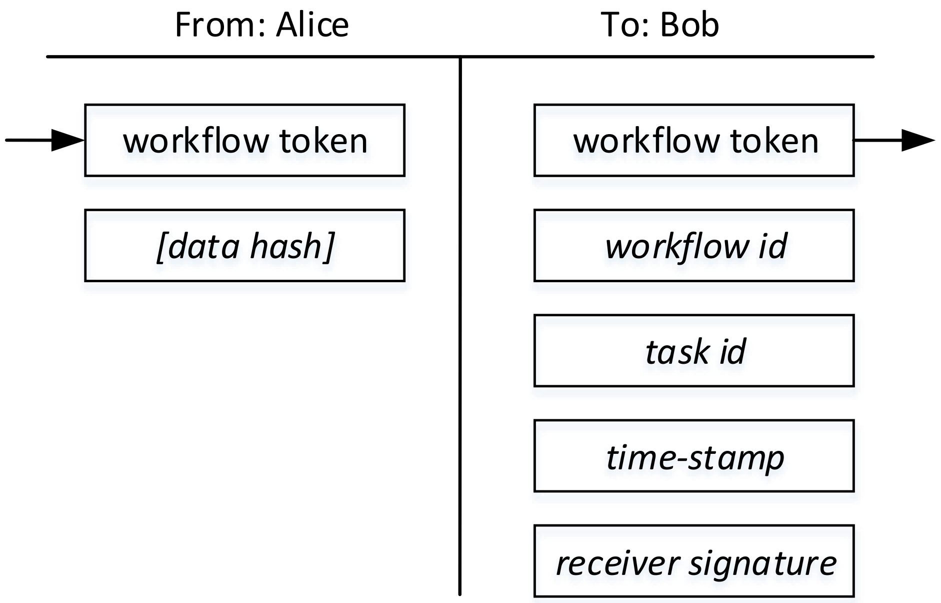


Figure 5.3: General Structure of a Workflow-Handover Transaction

To completely describe the execution of a workflow instance that includes activities, exclusive-or (XOR) path decisions and parallel execution paths, additional types of transactions are needed. The controlled handover between companies, as described above, documents the execution of activities. Each handover can mark the end of a previous activity and the start of a new one. XOR path decisions do not require a dedicated documentation marker since they always resolve into one single execution path. By analyzing the sequence of activities, a participant can determine how the XOR path decision was resolved. Besides this, the following documentation elements (i.e. transactions) are required.

Start of a workflow In order to mark the execution start of a workflow instance, this documentation element has to be published.

End of a workflow By publishing the distinct end of a workflow instance, all participants get notified that this instance has ended. Furthermore the instance's duration can be calculated.

Split into parallel execution To enable parallel execution paths the documentation of a workflow instance must be split and follow different paths. This split must be explicitly recorded to mark the start of the different sub-paths that diverge from it.

Join from parallel execution When parallel executing paths of a workflow instance are joined, the different documentation paths of the sub-paths must also be joined. To explicitly record this, a designated transaction is required.

The transaction to *start a workflow* is submitted by the process owner. It takes an arbitrary number of common Bitcoin inputs and outputs the token to be used for the workflow instance. It further documents the workflow id, a timestamp and a specific start-of-workflow marker. The timestamp of this transaction defines the starting time of its given workflow instance. At last, a change output can return the surplus Bitcoins. Thus, this transaction prepares the workflow instance token. The token itself still remains under the control of the process owner, the output which holds the token still belongs to the process owner.

The general structure of a workflow-start transaction is illustrated in Figure 5.4.

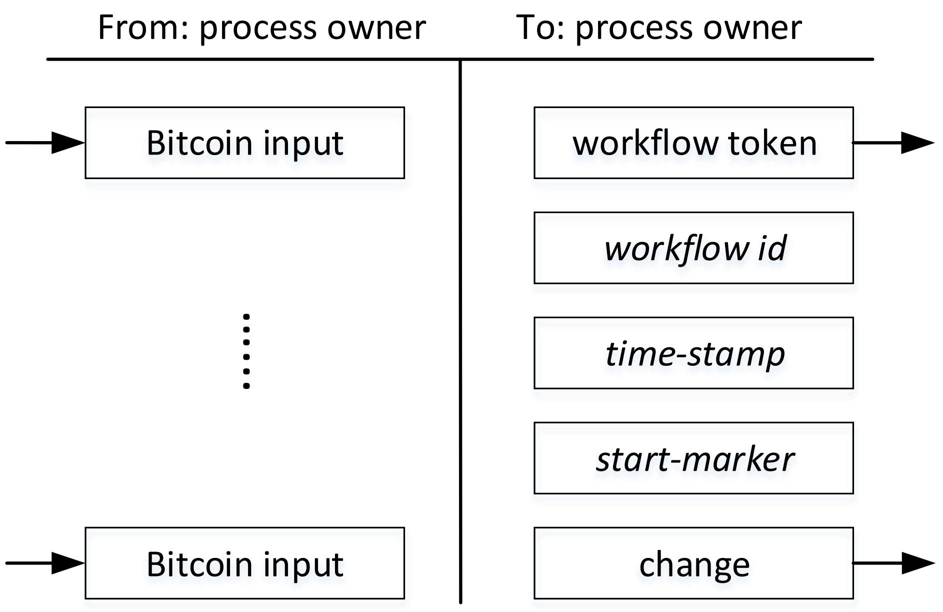


Figure 5.4: General Structure of a Workflow-Start Transaction

In this thesis it is assumed that a process always starts and ends at the process owner. Since the process owner usually expects some kind of result to be returned by the participants of the choreography, the last workflow handovers should point back to the process owner. Therefore, the transaction to *end a workflow* is also submitted by the process owner. As input it expects the token of the workflow instance and optional a data hash to document. The token is then no longer required for the runtime verification framework and the output can be used for arbitrary purposes. The transaction further documents the workflow id, a timestamp and a specific end-of-workflow marker. The timestamp of this transaction defines the ending time of its given workflow instance. Input as well as output remain under the control of the process owner. The Bitcoin money received through the workflow token can then be used for new workflow instances. The general structure of a workflow-end transaction is illustrated in Figure 5.5.

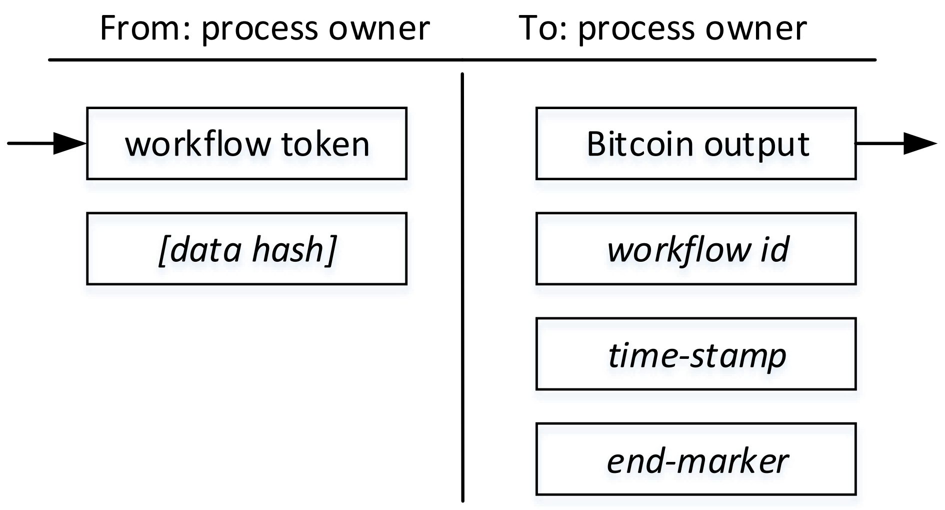


Figure 5.5: General Structure of a Workflow-End Transaction

In order to enable parallelism in workflows a specific transaction to *split a workflow* is required. This transaction is not meant to be used to transfer workflow tokens between different participants. Instead, only one participants who decides to split a workflow creates the transaction. The single token input and at least two token outputs are all under the control of the current workflow token owner. This predefines the number of paths, the workflow execution is split into. For each of the created paths, the workflow token owner can then add individual workflow-handover transactions to other participants. Again also the workflow id, a timestamp and a specific split-of-workflow marker are documented. The timestamp of this transaction defines when the given workflow instance was split into parallel paths. If a data hash was documented from the previous task execution, this data is documented along with the token. The general structure of a workflow-split transaction is illustrated in Figure 5.6.

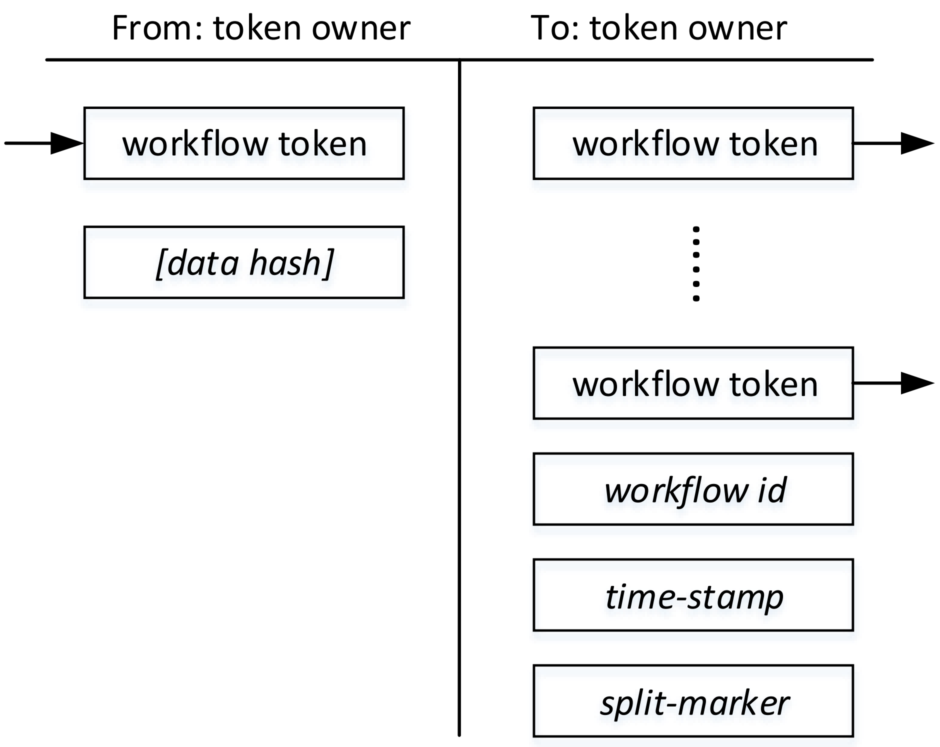


Figure 5.6: General Structure of a Workflow-Split Transaction

To join parallel execution paths, a transaction to *join a workflow* is required. Similar to the start-, end- and split-transactions it is not used to transfer the ownership of a workflow token. When a choreography participant accepts a workflow task which requires inputs from multiple execution paths (i.e. it requires a workflow join), the execution can not be proceeded until all other execution paths also have been handed over to this specific participant. When executing different workflow paths in parallel, one path will always be the first to finish. The workflow token of this subpath is then handed over to a participant which agrees to proceed the execution. All other execution subpaths then must also handover their execution tokens to this specific participant. The different execution tokens are then merged into a single token by a dedicated workflow-join transaction. Naturally this transaction expects at least two token inputs with optional data hashes and provides a singe token output. As usual also the workflow id, a timestamp and a specific join-of-workflow marker are documented. The timestamp of this transaction defines when the given workflow instance was joined from parallel paths The general structure of a workflow-join transaction is illustrated in Figure 5.7.

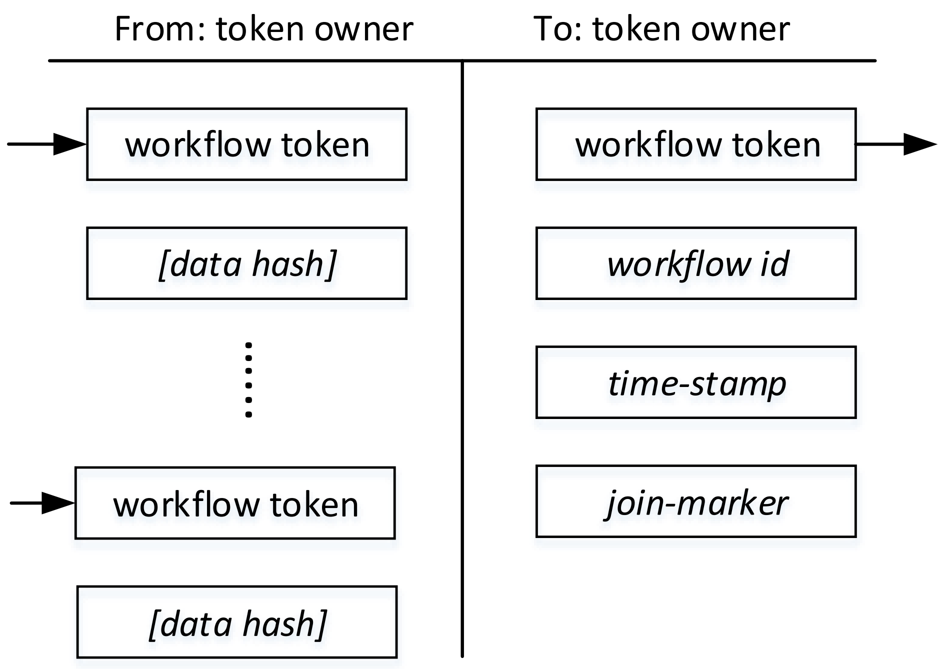


Figure 5.7: General Structure of a Workflow-Join Transaction

5.4 Prototype Description

The previous section provided a general description of the runtime verification proposal of this thesis. In this section implementation specific details about the Bitcoin transactions, the handover between participants and the implemented prototype are given. Section 5.4.1 explains the concept of standard transactions in Bitcoin, describes how they are structured and demonstrates how they are validated. The detailed implementation of Bitcoin transactions enriched with workflow information is outlined in Section 5.4.2. At last, the framework implemented around the proposed runtime verification concept is described in Section 5.4.3.

5.4.1 Bitcoin Standard Transactions

In comparison to second generation Blockchains like Ethereum, the Bitcoin Blockchain was designed to serve a specific purpose, the transfer of digital money. Since Bitcoin established the first Blockchain, its original design provided various possibilities to create experimental transactions for other use cases. As described in Section 2.3, a Bitcoin transaction basically consists of a set of inputs and outputs. Each input points to the output of a previous transaction. The funds associated with each output are locked by a script. An output can only be accessed if an input can provide input parameters that resolve the script guarding the output to true. The structure of a Bitcoin transaction is welldefined. Only two elements do not have a defined length and can be filled with arbitrary values, the script locking an output, historically called *scriptPubKey*, and the unlocking script provided by an input, historically called *scriptSig*. In this thesis these two scripts will be referred to as *locking script* and *unlocking script* as done by [10].

Bitcoin defines a custom scripting language which is expected to be used for the locking and unlocking scripts. While it is on purpose not Turing Complete, it still enables extensive variants of programs. Unfortunately, different bugs where encountered in the implementation of the script's interpreter in earlier versions of Bitcoin. As a solution a standardization test for transactions was introduced [10]. The developers of the main Bitcoin software[3] added an *IsStandard()* method which checks the contained scripts. Only if all locking and unlocking scripts of a transaction exhibit one of five different predefined structures, the received transaction is regarded as "standard". Currently the main Bitcoin software discards all incoming transactions that are sent across the mainnet which are not standard. This means that all miner nodes which run an instance of the main Bitcoin software will not accept non-standard transactions. In order to include a non-standard transaction into the Bitcoin Blockchain one first must find a miner that is accepting non-standard transactions. In addition, this specific miner then must win the race of creating a new block, which can take a very long time. It is not explicitly prohibited to create and publish non-standard transactions but it has become de facto very difficult to integrate them into the Blockchain.

[3]`https://github.com/bitcoin/bitcoin/`

Only two of the five variants still provide the possibility to insert arbitrary data and still be considered as standard [10]. The first three transaction types are called *Pay-to-Public-Key-Hash* (P2PKH), *Pay-to-Public-Key* and *Multi-Signature*. These three transaction types define very specific script structures and therefore only serve very specific use cases, namely payment. The most commonly used script type is the P2PKH scripts. It is used for every simple payment transaction which is conducted between two actors. The proposed runtime verification approach also partly relies on this transaction type. That is why it is described in greater detail. The locking script has to be of the following structure.

Listing 5.1: P2PKH locking script

```
OP_DUP OP_HASH160 <public-key-hash> OP_EQUAL OP_CHECKSIG
```

The unlocking script has to be of the following structure.

Listing 5.2: P2PKH unlocking script

```
<signature> <public-key>
```

In order to determine if the locking script results to true, both scripts are concatenated and executed together. All elements which are not well-known commands are considered as data chunks. Since the Bitcoin script language is stack-based, all encountered data chunks are pushed to the stack. The combination of a P2PKH unlocking and locking script results in the following script.

Listing 5.3: P2PKH scripts combined for validation

```
<signature> <public-key> OP_DUP OP_HASH160
<public-key-hash> OP_EQUAL OP_CHECKSIG
```

The execution of this P2PKH is very straightforward. The provided signature and corresponding public key are pushed to the stack. The public key on the stack is duplicated and the topmost public key entry on the stack is hashed. These steps are illustrated in Figure 5.8.

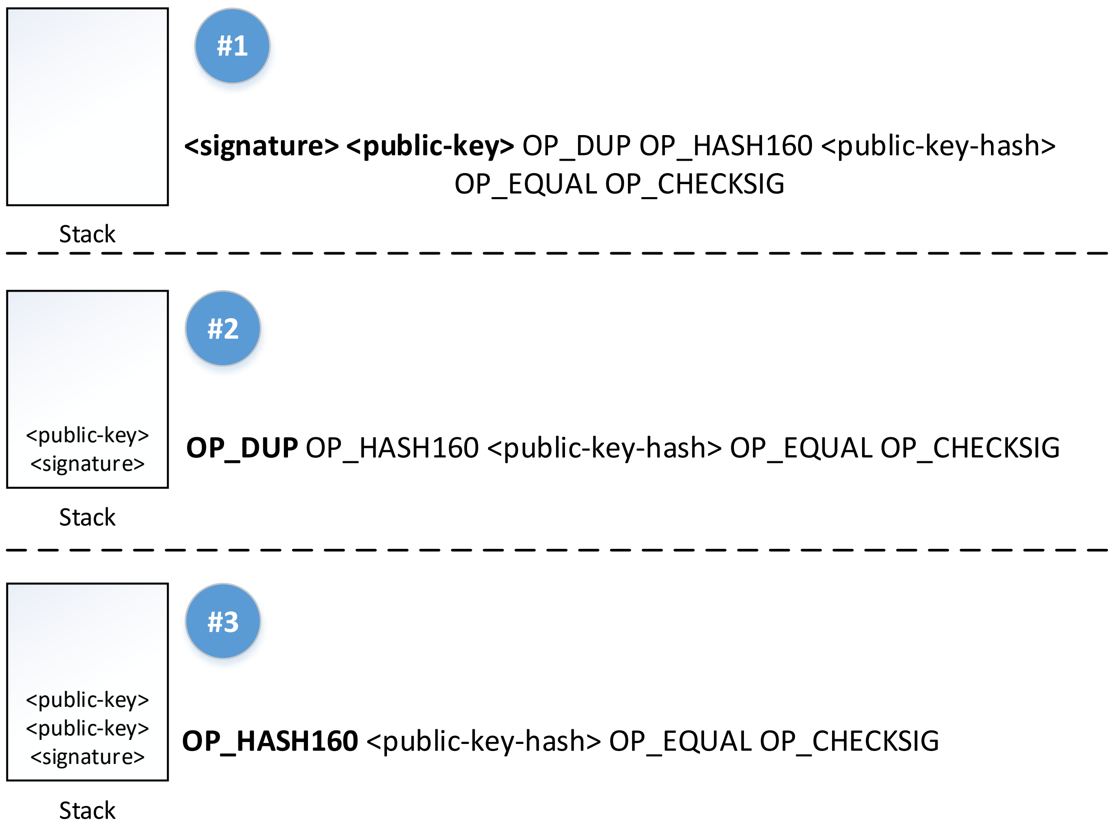

#1
<signature> <public-key> OP_DUP OP_HASH160 <public-key-hash> OP_EQUAL OP_CHECKSIG
Stack
#2
<public-key>
<signature>
OP_DUP OP_HASH160 <public-key-hash> OP_EQUAL OP_CHECKSIG
Stack
#3
<public-key>
<public-key>
<signature>
OP_HASH160 <public-key-hash> OP_EQUAL OP_CHECKSIG
Stack

Figure 5.8: First steps of validating a P2PKH transaction (adapted from [10])

Next the public key hash of the locking script is pushed to the stack and compared to the just created hash. This is a cheap way of determining if the public key, provided by the unlocking script, is the expected one. At last the remaining public key on the stack is used to check the signature which also resides on the stack. These steps are illustrated in Figure 5.9.

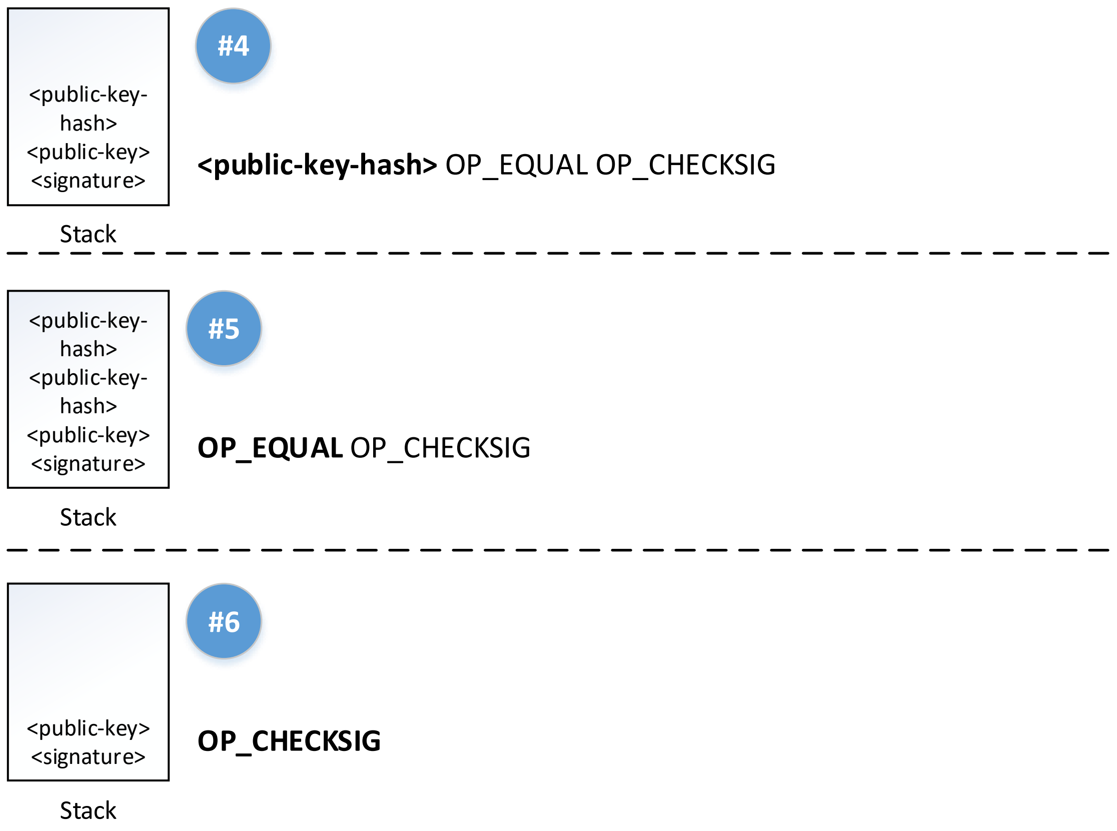


Figure 5.9: Next steps of validating a P2PKH transaction (adapted from [10])

The signature is expected to contain a hash of the currently validated transaction. Since the signature itself can not be part of this hash, the unlocking scripts must be removed before hashing the transaction. The unlocking script of the referenced input is replaced with the locking script of the currently validated output. The unlocking scripts of all other inputs are set to blank. The process of validating the signature to an input of a transaction that uses the P2PKH method is further illustrated in Figure 5.10.

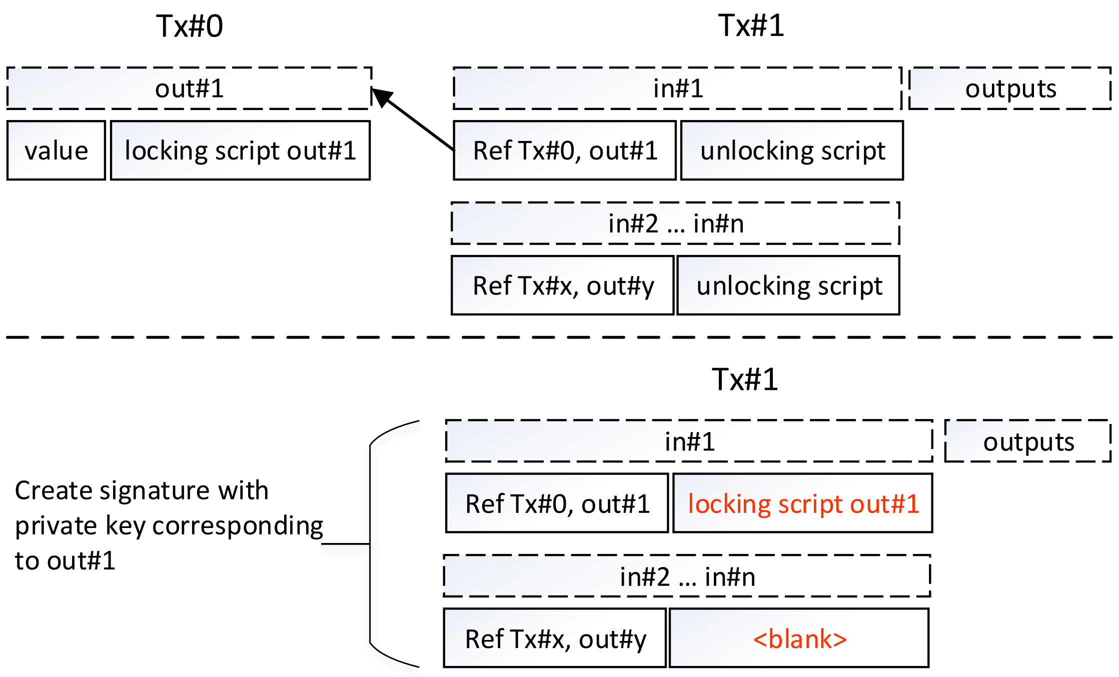


Figure 5.10: Creation process of Bitcoin validation signatures

The *Pay-to-Public-Key* script type is an older and simpler variant of the P2PKH type. The locking script directly contains the public key and only the signature is expected as input. This omits the hash check before the signature check. While still being used by older Bitcoin clients, this transaction type has two disadvantages in comparison to the P2PKH type. Instead of the cheap hashing check, the expensive signature check is always executed in order to check the validity of the unlocking script. Furthermore, the placement of the used public key in the locking script increases the byte size of this script significantly. To perform quick and efficient validation of new broadcast transactions, mining nodes try to keep all unspent transaction outputs directly in memory. The usage of P2PKH transactions instead of Pay-to-Public-Key transactions greatly reduces the memory required for the unspent outputs.

In order to enable payments which have to be approved by multiple parties, the *Multi-Signature* script type is defined. It enables payers to lock a transaction output with a script that requires not one but multiple signatures from different Bitcoin key pairs to unlock. This transaction type can be used to implement fault management strategies in the proposed runtime verification prototype. Since fault management strategies are out of scope for this thesis, the Multi-Signature script type is not discussed in detail.

The two standard transaction types which still allow a certain degree of freedom are *Data output* and *Pay-to-Script-Hash* (P2SH). *Data output* is the only standard type without a direct purpose for payment. It is designed to serve as simple and limited data field. It is the last remaining concession towards alternative and experimental use cases in the Bitcoin Blockchain. To directly place data in a transaction, a specialized output with 0 BTC must be created. Since its only purpose is to contain data, it should never be referenced by another input or carry value.

In order to ensure that this output is never successfully consumed by a new input, its locking script must never evaluate to true. This is achieved by simply placing the *OP_RETURN* operator at the top of the script. When this operator is executed, it immediately stops the execution. At that point, the value *true* is not on top of the stack, therefore the validation fails. Miners therefore can safely archive this kind of unspent output and do not have to keep it in memory. Originally only *40 bytes* of data were allowed to be stored in such an output. As of *version 0.11.* of the main Bitcoin software it is allowed to store up to *80 bytes* of information in such data output output. Only one such data output is allowed per transaction. The data output transaction type is illustrated in Figure 5.11.

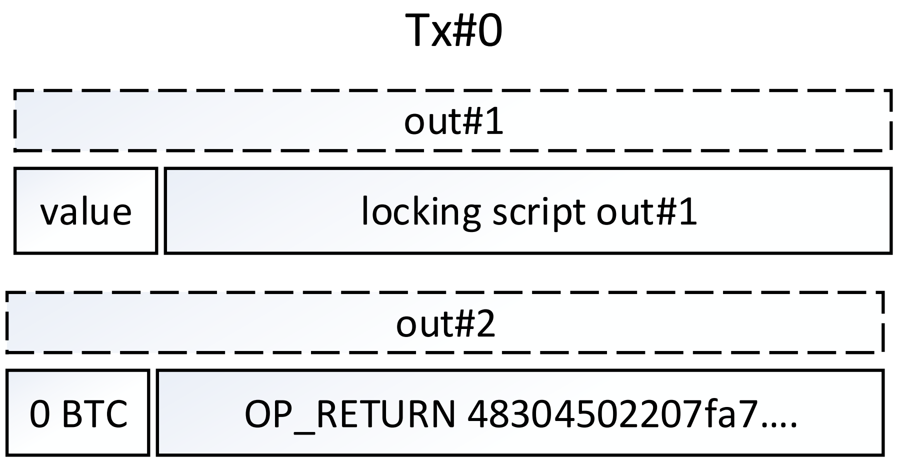


Figure 5.11: Structure of standard transaction type “data output”

The P2SH type is a quite new and powerful transaction type to conduct payments. Amongst others, it allows the controlled usage of non-standard scripts. In contrast to the strictly defined P2PKH transaction type structure, the P2SH transaction type enables the usage of various payment scripts. In order to allow miners to save memory, the locking script of P2SH transactions remains short and strictly defined. It is the unlocking script which can contain arbitrary data. P2PKH locking scripts require a signature and a public key as parameters. P2PKH locking scripts require as parameter an entire redeem script followed by the redeem script’s own parameters. The locking script of a P2SH transaction is a simple hash value comparison. Upon creating a P2SH output, the transaction publisher must decide which script should be provided for unlocking the output and hash it. This hash is then placed in the locking script in the following way.

Listing 5.4: P2SH locking script

```
OP_HASH160 <script-hash> OP_EQUAL
```

In order to unlock an output with such a defined script, a redeem script which matches the placed hash must be provided. In addition, this provided redeem script itself is evaluated and must resolve to true. This feature is normally used for Multi-Signature scripts which tend to be quite long. When used in the P2SH variant, they save memory for the miners. The unlocking process of a P2SH is illustrated in the following listings. As redeem script, a simple P2PKH is used. Listing 5.5 shows the redeem script, defined by the publisher of *Tx#0*.

Listing 5.5: P2SH redeem script example

```
<redeem-script> = OP_DUP OP_HASH160 <public-key-hash> OP_EQUAL OP_CHECKSIG
```

In order to unlock a P2SH output, the redeem script and its required parameters must be placed in the unlocking script.

Listing 5.6: P2SH unlocking script example

```
<signature> <public-key> <redeem-script>
```

The P2SH validation is then performed in two stages, first the provided redeem script is compared against the defined hash. Second, the redeem script itself is evaluated with its parameters.

Listing 5.7: P2SH validation example

```
 Stage #1: <redeem-script> OP_HASH160 <script-hash> OP_EQUAL

 Stage #2: <signature> <public-key> OP_DUP OP_HASH160 <public-key-hash>
OP_EQUAL OP_CHECKSIG
```

5.4.2 Workflow Handover Transactions

To store the workflow information required for a handover between two participants in the Blockchain, as defined in Section 5.3, the two transaction types data output and P2SH are used. The elements workflow instance id, task id, timestamp and receiver signature are included in a transaction by using a data output element. By using a simple adapted P2SH output, the workflow data hash can also be included. The only downside of using P2SH elements is the fact that the stored data can only be placed in the redeem script. This redeem script is part of the unlocking script which becomes only visible on the Blockchain after the output has been spent, i.e. another transaction consumed the output by placing the redeem script on the Blockchain.

The 80 bytes storage of the data output element are divided in the following way to store the described elements. The first byte is used to store the length of the stored data block, which may vary depending on the workflow transaction type. Next, two bytes are reserved to store the workflow instance id. This enables the definition of 65, 535 unique workflow instances in this kind of runtime verification proposal. To store the task id, which follows after the workflow instance id, one byte is reserved. Therefore, 255 different tasks can be defined inside a single workflow model. The task id is followed by a Unix timestamp with a length of 4 byte. At last, the Bitcoin signature of the receiver is placed. Then length of this signature is not fixed but commonly ranges between 71 to 72 bytes. This results in a data block with a total length of 79 to 80. The structure of the data output workflow element is illustrated in Figure 5.12.

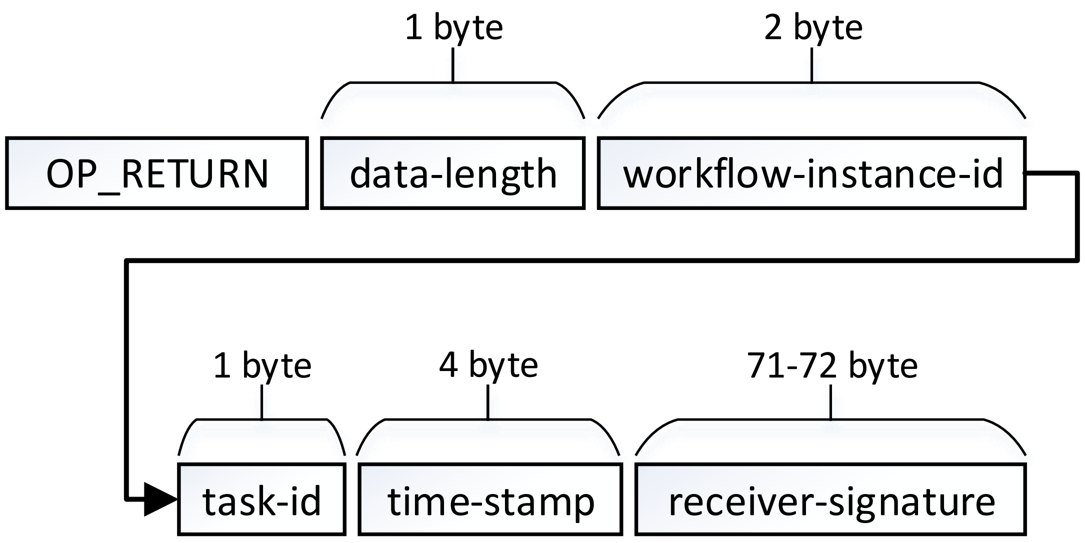


Figure 5.12: Structure of workflow data in a Bitcoin data output

The workflow data hash does not fit into the data output element, therefore a P2SH transaction has to be used to store it. At the same time, such P2SH outputs are used to transfer workflow instance tokens, therefore also the access restriction features of a P2PKH transaction are required. In order to achieve this, a P2PKH script with an optional data hash appended is used as redeem script. The data hash itself does not add any specific functionality to the script, it is just there to be placed on the Blockchain as plain text. A simple *OP_DROP* command ensures that the hash is removed from the stack before the actual P2PKH script is executed. The inclusion of the data hash is completely optional. The structure of the P2SH redeem script with included workflow data hash is illustrated in the following listing.

Listing 5.8: P2SH workflow handover redeem script

```
<workflow-data-hash> OP_DROP OP_DUP OP_HASH160 <public-key-hash>
OP_EQUAL OP_CHECKSIG
```

The complete unlocking script, including the required parameters of the P2PKH script, is illustrated in Listing 5.9.

Listing 5.9: P2SH workflow handover unlocking script

```
<signature> <public-key>  <workflow-data-hash> OP_DROP OP_DUP OP_HASH160
<public-key-hash> OP_EQUAL OP_CHECKSIG
```

As mentioned before, the workflow data hash is placed onto the Blockchain only after the given output has been consumed, i.e. the given token has been passed on. But the redeem script hash placed in the P2SH locking script ensures that only the correct workflow data hash can be placed in the redeem script. Furthermore, before a handover transaction can be published both workflow participants are in possession of the workflow data in its current state. The receiver of a workflow handover is able to verify that the given P2SH output incorporates the correct workflow data hash. In addition, the sender of a workflow handover can provide the data hash when the process owner demands it. This way, a process owner can also verify that a data hash has been documented, even if the corresponding token output has not yet been passed on.

The general steps to perform a handover between two participants were illustrated in Figure 5.1. In step 4 a *handover-transaction template* is created by the sender of the handover which already includes almost all required workflow data. This template is only missing two signatures, one from the receiver and one from the sender. This initial template is described in detail in Figure 5.13.

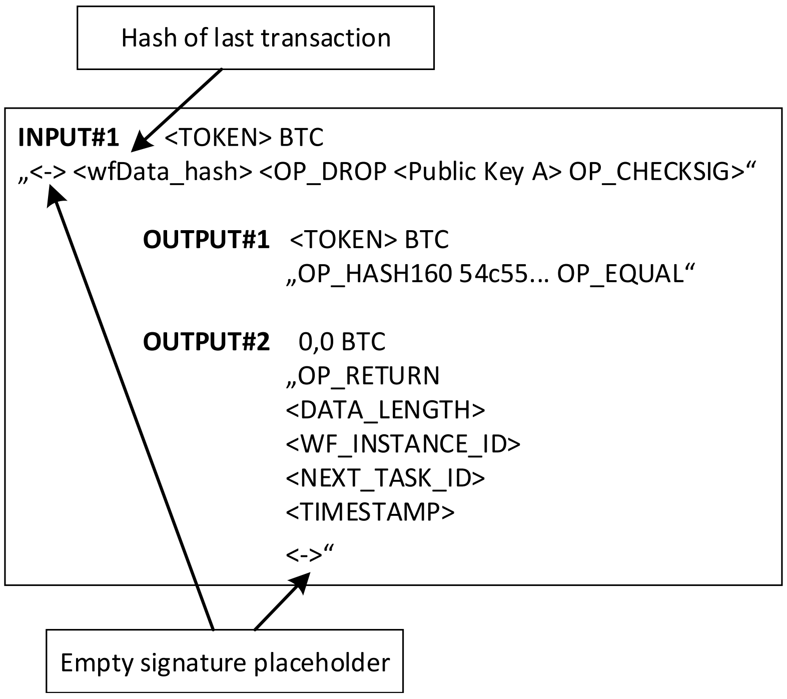


Figure 5.13: Handover transaction template without signatures

Upon receiving the template, the handover receiver is able to validate the correctness of the following critical elements.

INPUT#1 contains redeem script INPUT#1 references the output of a previous transaction. This output must contain a P2SH locking script. The unlocking script, currently included in the template, is still missing parameters (i.e. the sender signature and public key) but the redeem script itself is already complete. Therefore the handover receiver is able to validate the correctness of the redeem script, thus also the correctness of the included wfData_hash of the last transaction.

OUTPUT#1 can be retrieved Though the data of OUTPUT#1 is abstracted by a P2SH script, the handover receiver knows the defined structure of the script. In addition, all required data to check the correctness of the included script hash is already known. The redeem script should be constructed from the hash of the receiver's public key and the hash of the already transferred workflow data. By recreating the redeem script and comparing it to the hash placed in OUTPUT#1, the handover receiver ensures that the token is indeed correctly passed on and that the hash of the just received workflow data is correct.

OUTPUT#2 contains the negotiated terms Except for the workflow data hash, all negotiated workflow data is included in OUTPUT#2, as defined above. The correctness of the included data, therefore can directly be verified.

Previous workflow execution is valid During the negotiation the handover receiver also receives the workflow model. Since INPUT#1 is referencing a previous workflow transaction, the receiver is able to trace the execution history of the workflow instance. Besides other meta-information about the workflow, it can be determined if the workflow execution still conforms with the defined workflow model.

If the received handover template is successfully validated, the handover receiver simply hashes it and signs it. For the signature the same Bitcoin key-pair is used that has been utilized to receive the token in OUTPUT#1.

After defining which data elements are stored by what transaction elements, the technical structure of a workflow-handover transaction is illustrated in Figure 5.14.

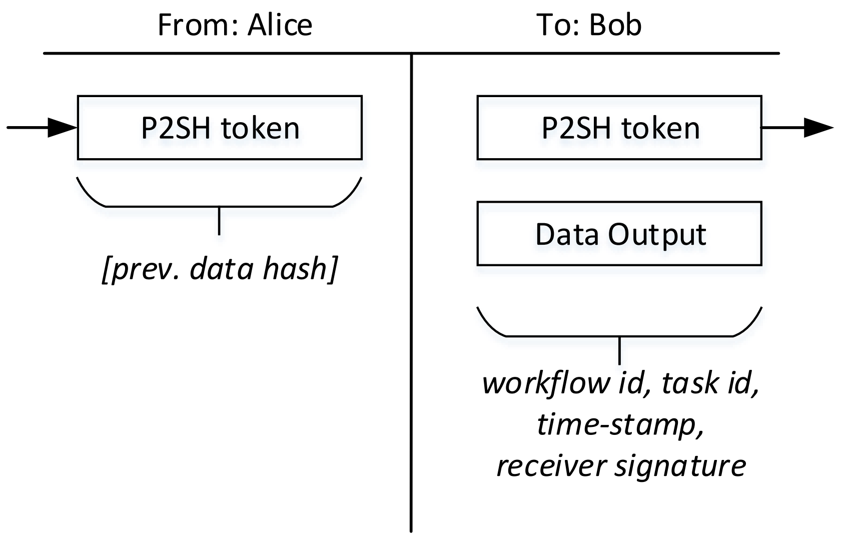


Figure 5.14: Detailed structure of a workflow-handover transaction

It is also possible to incorporate other types of redeem scripts inside the P2SH script. Fault management strategies can be employed through the usage of a Multi-Signature type script instead of P2PKH type script. While the P2PKH script only can be unlocked by the new token owner, a Multi-Signature script can be unlocked by multiple different parties. In case a fault occurs during a workflow, an escalation strategy has to be employed by the process owner. To enable the process owner to intercept a workflow token in the case of incorrect enactment, a Multi-Signature script can be placed as redeem script in the P2SH locking script.

The Multi-Signature script is then configured to grant access for two people instead of one, namely the next token owner and the process owner. For instance, if the new token owner stops responding and does not perform the negotiated task, the process owner could decide to collect the token instead. Fault management is out of scope for this thesis, therefore in the proposed prototype only simple P2PKH scripts are employed as redeem scripts.

The technical details of the other required workflow transaction types workflow-start and workflow-end are illustrated in the following figures. As explained in the general concept in Section 5.3, for these transactions the token itself remains under the control of the same participant.

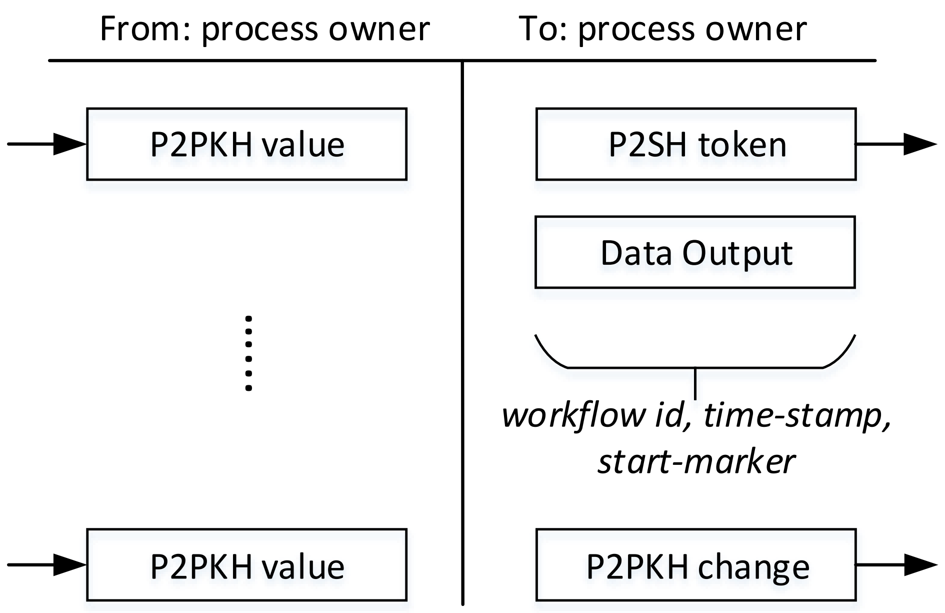


Figure 5.15: Detailed structure of a workflow-start transaction

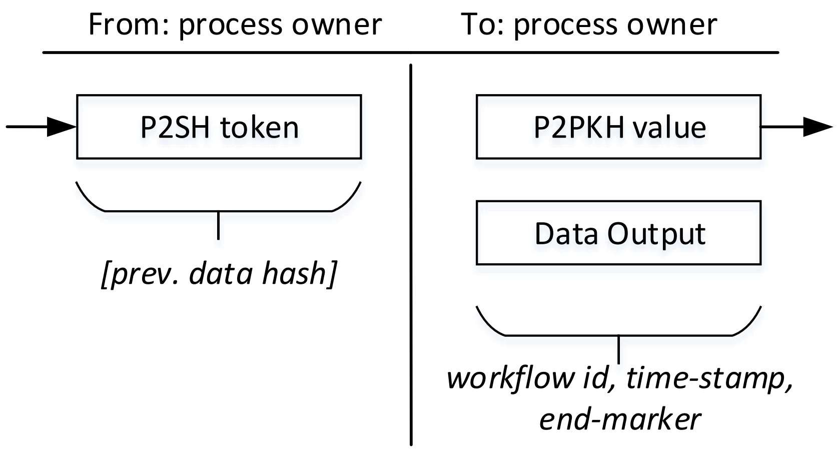


Figure 5.16: Detailed structure of a workflow-end transaction

The technical details of the required workflow transaction types workflow-split and workflow-join are illustrated in the following figures.

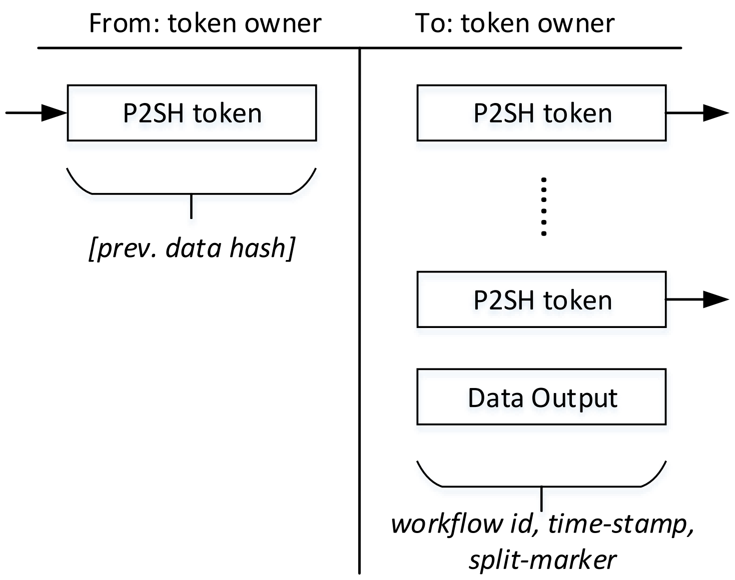


Figure 5.17: Detailed structure of a workflow-split transaction

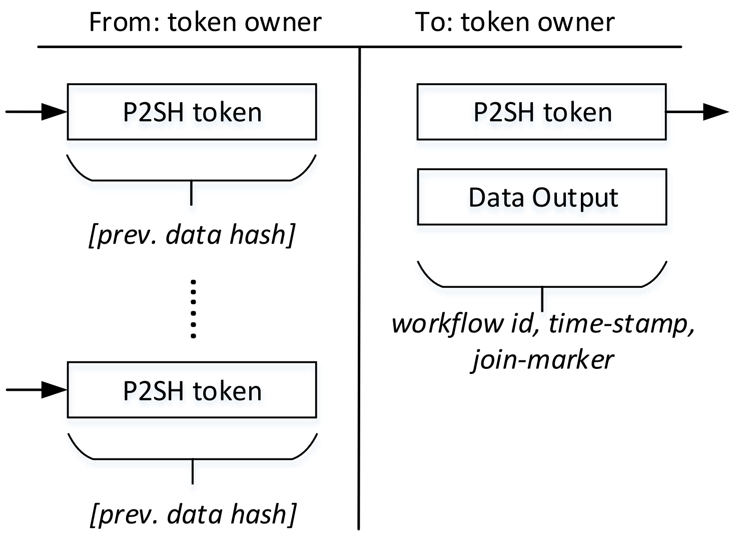


Figure 5.18: Detailed structure of a workflow-join transaction

5.4.3 Workflow Handover Framework

A Java-based software framework was developed to implement the runtime verification approach as described above. This framework is designed to efficiently abstract away all implementation details in order to simplify its integration into choreography-oriented WfMSs. The framework is further enhanced by using a *simple payment verification* (SPV) [23] Bitcoin client as slim foundation, a *remote REST API* for data collection and a *greedy publishing* mode. All these features increase the usability and testability of the prototype. On the other hand, these features are not essential for the implementation of the proposed approach.

Traditional Bitcoin clients interact with the Bitcoin P2P network and manage an internal keystore for the received and sent funds. These keystores are referred to as *wallets*, since the contained keys define which unspent transaction outputs can be accessed. In order to listen for changes concerning a wallet, Bitcoin clients are connected to the Bitcoin P2P network. They monitor blocks and transactions that are broadcast across the network. Based on the received information, they maintain their own copy of the Bitcoin Blockchain. Bitcoin clients always want to ensure that their copy of the Blockchain is up-to-date. This way clients are able to verify received transactions and can ensure that only valid payment transactions are created by them.

This precondition leads to two challenges. The Bitcoin Blockchain is continuously growing and currently requires over 73 GB of disc space [6]. In addition, the synchronization of a local Blockchain copy with the P2P network is not very fast. The missing information is collected from other nodes in the P2P network. If a client is running all the time, updates can be received continuously. But if a client was offline for a longer timespan, larger chunks of information have to be obtained. Such synchronization may take minutes, making clients very impractical for infrequent usage.

Both of these challenges hinder the creation of slim Bitcoin clients. Especially mobile applications can not operate under these conditions. They do not have the required disc space available and the significant increase of the starting time of these applications are unacceptable for users. To address this problems, the concept of *simple payment verification* (SPV) was proposed. This concept, which is also used by the prototype of this thesis, does not rely on downloading the entire contents of the Blockchain. This benefit is gained by reducing the level of security during transaction verification in comparison to a full node.

A full node maintains a full copy of the Blockchain, including all blocks with all their transactions. Therefore, a full node is able to verify on its own, if a received block or transaction are valid or if an output is still unspent. Upon starting, a SPV client does not synchronize the whole content of each block in the Blockchain. Instead, only the header of each block is stored. In addition, only information relevant to transactions and addresses contained in the client's wallet are requested from other clients. This reduced synchronization is way faster than the synchronization of full nodes and therefore more feasible for slim Bitcoin clients with long offline periods.

SPV clients are not able to verify the contents of blocks or transactions. Instead the used wallet framework BitcoinJ connects to a random number of Bitcoin peers (>=10). If all peers relay the same blocks or transactions, they are considered valid. The greatest threat to this kind of SPV clients therefore are Sybil attacks [67], where an attacker is in full control of a clients internet connection. Another, but less realistic threat is the Finney attack [22]. During this attack, a malicious node prepares a valid block which includes a transaction that simply shifts funds of the attacker from one address to another. Before publishing, the attacker spends the same funds at a merchant that accepts unconfirmed transactions. After the transaction, the prepared block is published. This scenario is very unlikely in this certain use case, since the attacker must be able to create a valid block faster than the entire mining network. Furthermore, the attacker must be able to purchase and receive goods from a merchant before a block is created by another mining node.

After the initial synchronization, the SPV client monitors the Bitcoin network just like a full node, but only collects information about addresses and transactions that are related to its wallet. Each newly created block that is relayed by all connected nodes and fits on top of the stored and synchronized block header chain increases the trust into this synchronized data. Among other information, the header of a block stores the hash of the previous block and its Merkle root [55]. By using this Merkle root, a SPV client is able to verify if a certain transaction is indeed included in a certain block. Through the usage of Bloom filters [26], a SPV client can request information about a transaction from other Bitcoin nodes. These nodes then return the requested transactions together with their respective Merkle paths. As long as the SPV client trusts the collected header of a block, it can prove that a transaction was included into the given block. This is another way of meta-verifying the validity of a given transaction for SPV clients.

During the handover of a workflow choreography the receiving participants want to validate the execution path of the instance prior to the handover. Therefore, they need to be able to obtain information about old transactions that have been submitted to the network or are currently pending. Since the implementation proposed by this thesis relies on a SPV foundation, only block hashes and transactions directly related to the client's wallet are monitored. In order to obtain information about transactions unrelated to the wallet, a separate data collection framework has to be used. Many different companies[4,5,6] provide live access to Bitcoin information through *remote REST APIs*. By calling different REST interfaces, slim Bitcoin clients can obtain information about any confirmed or still pending transaction in the Bitcoin network. This way clients are able to reconstruct the execution of a workflow instance without running a full Bitcoin node. For the prototypical implementation in this thesis the REST API of `blockcypher.com` is used.

As described in Section 2.3, the required proof of work for the creation of new blocks in

[4]`https://blockchain.info/de/api`
[5]`https://api.blockcypher.com`
[6]`https://www.blocktrail.com/api`

Bitcoin is configured to result in a median block creation time of ten minutes. Unfortunately there is a lot of variation in the time between block creations. The time between the arrival of two blocks roughtly follows an exponential distribution. There may be mere seconds between the creation of two blocks or even an hour [40]. If too many transactions are published at the same time or if blocks are created too slow, published transaction must be buffered by the miners of the P2P network. On average only 1, 500 transactions are currently included in a new block [25]. If there are more than 1, 500 transactions queued to be included in a new block, some transactions might need to wait two or even three block creation intervals until they are confirmed. Also the fees offered by each transaction affect their confirmation time. Naturally, transactions with higher fees are treated with higher priority. This results in an even higher variation for the confirmation time of transactions [40].

Therefore, the transaction confirmation time of Bitcoin is expected to be a major challenge for a runtime verification approach based on the Bitcoin Blockchain. The conservative approach to runtime verification would be to wait for the confirmation of each published transaction before appending new transactions to it. For long running use cases with long intervals between handovers, like transportation, this would be sufficient. But it would represent a bottleneck for fast running workflows with short intervals between handovers, like software computations.

Because of this, the workflow handover framework, which implements the approach proposed by this thesis also incorporates a *greedy publishing* mode. A full Bitcoin node receives notifications about pending transactions on its own. Alternatively, the Blockchain information retrieval REST API, described above, is able to return information about pending transactions as well. This way, also SPV-based clients are able to retrieve information about pending transactions.

Through this greedy approach, it is possible to append new transactions to a workflow, even though the latest ones have not yet been confirmed. Since information about these transactions has not yet been placed in the Blockchain but is available in the Bitcoin network, choreography participants are still able to verify the validity of a transaction to a given workflow instance. Also Bitcoin miners accept new transactions which reference unconfirmed transactions as input. It is possible to create whole chains of unconfirmed transactions, enabling the participants of a shared workflow instance to continue with its execution even though not all workflow transactions have been included into the Blockchain yet.

By using this approach, the execution of a fast pacing workflow is not delayed. Furthermore, multiple chained transactions can be confirmed at once in a single block. Through this, less blocks have to be created to confirm all published transactions of a workflow instance. In addition, each participant is in possession of the complete workflow execution chain and can prove that handovers of the workflow instance took place that were confirmed by both handover partners. The risk of this approach is that a whole chain of pending transaction may be dropped if something goes wrong.

If one intermediate transaction of the pending chain is somehow lost, the whole remaining chain is also dropped since it became invalid. Each published transaction is *flooded* through the P2P network and stored in the buffers of various miners. Even during a conflict in the Blockchain, where it may occur that single blocks are dropped, the transactions should still remain in the pending buffer. The highest threat for this greedy approach is malicious behaviour of one of the participants. Each participant is theoretically able to publish an alternative version of an intermediate transaction in the pending chain. In this case, two alternative transactions become available for a single output. It is undefined which transaction will be included in the Blockchain. If the alternative transaction of the malicious participant is chosen, the remaining pending transaction chain is dropped.

On the contrary, it is very unlikely that the alternative transaction, published by the malicious participant, is a valid handover transaction. The malicious participant still requires another participant to confirm a handover. At the same time, this participant can easily verify that another workflow transaction is already pending for the given token output. At last, the alternative transaction also documents which participant caused the disruption, resulting in penalties and loss of reputation.

The features SPV and greedy publishing both simplify the usage of the runtime verification framework. By using a SPV foundation, the framework becomes much easier to test and integrate into a choreography-oriented WfMS. The proposed usage of a remote REST API as an additional information source reduces the risk introduced by the SPV approach. The runtime verification framework can operate in the same way, if a full node is used instead. While reducing the usability of the framework, its level of security would be increased.

Greedy publishing aims to enable fast paced workflows in the slow running environment of the Bitcoin Blockchain. To enable this feature in conjunction with an SPV client also requires the usage of a remote REST API to fetch additional information. If a full node would be used, the feature of greedy publishing would still be possible but the usage of the remote REST API can greatly be reduced. Since the full node receives and buffers most pending transactions itself, the REST API would only be required during rare exceptions.

The prototype itself was developed by using the following technologies and frameworks.

Java Development Kit (JDK) 1.8 Serves as the technological foundation.

Apache Maven 3.3.9 Provides flexible dependency management. Required software libraries for the prototype can easily be defined and are loaded automatically. Furthermore, Maven defines clear build processes which are supported by many integrated development environments (IDEs).

Spring Beans 4.2.6 Supports the modular composition of the different components of the framework through dependency injection. Single components of the prototype are defined by using either annotation-based or XML-based configuration. These elements can then dynamically be injected into higher level components where needed.

Apache HttpClient 4.5.2 & Google Gson 2.7 The framework HttpClient is able to request and load any type of web resource, including a REST interface. The data returned by the mentioned Bitcoin Blockchain information REST APIs is commonly structured in JSON[7]. In order to extract the required information from the returned JSON objects, the framework GSON is used.

BitcoinJ 0.14.2 This framework provides basic management functions to operate a Bitcoin wallet. It can operate as a SPV Bitcoin client or as a full Bitcoin node. When running as SPV client, the reduced copy of the Bitcoin Blockchain can easily be created and maintained. The original purpose of the framework is to enable common payment transactions. Therefore, many of the original payment-oriented functions were adapted or re-purposed in order to enable the creation and publishing of transactions with data outputs and P2SH outputs as defined in the previous subsection.

JUnit 4.12 In order to assert the correct behavior of single components or of the whole framework itself, numerous test where defined by using the JUnit testing framework.

The software framework is divided into three main modules. The module *bitcoin-core* provides the basic Bitcoin functionality required to create the raw Bitcoin transactions which are enriched with runtime verification information. The information retrieval functionality from remote REST APIs is enabled by the module *bitcoin-crawler*. At last, these two models are both utilized by the module *handoverFramework* which provides a high level interface to perform the identity-aware workflow runtime verification tasks that have been described in the previous two subsections.

The module *bitcoin-core* is responsible for maintaining a Bitcoin wallet. In addition, the module provides the functionality to create and broadcast the workflow transactions as defined in Subsection 5.4.2. As the handover process describes, incoming workflow transactions are first received as templates from other participants. Therefore, this module also contains functionality to validate the structural correctness of both sent and received workflow transactions. The following figure illustrates the class diagram of this module.

[7]`http://json.org`

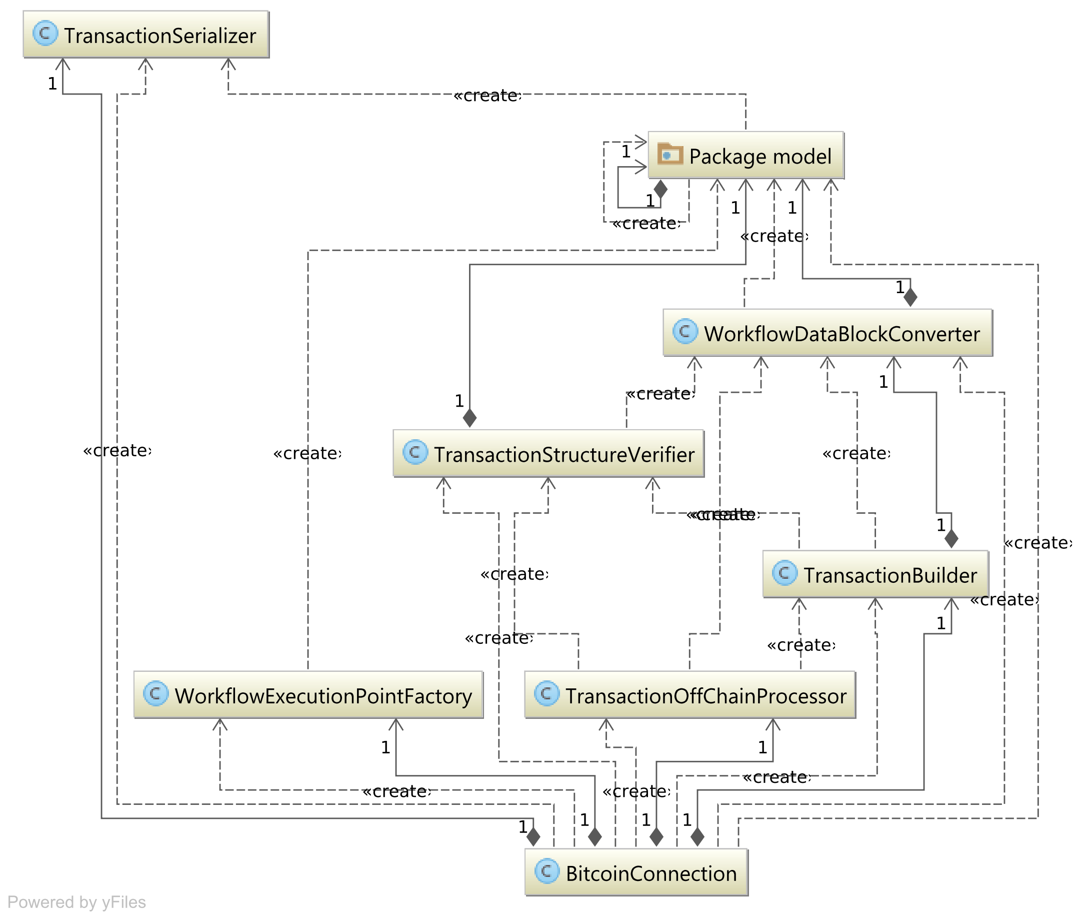

TransactionSerializer
Package model
WorkflowDataBlockConverter
TransactionStructureVerifier
TransactionBuilder
WorkflowExecutionPointFactory
TransactionOffChainProcessor
BitcoinConnection
«create»
1
Powered by yFiles

Figure 5.19: Class Diagram of Module *bitcoin-core*

The class *WorkflowDataBlockConverter* is able to create the byte blocks for the *data output* outputs of the different workflow transaction types as outlined in Figure 5.12. The means to create the structurally defined workflow transactions are provided by the class *TransactionBuilder*. The exchange and off-chain signing of handover templates between handover partners is supported by the classes *TransactionOffChainProcessor* and *TransactionSerializer*. To validate the structural correctness of basic workflow handover transactions the class *TransactionStructureVerifier* can be used. The main features of this module are exported by the class *BitcoinConnection*. It initializes and maintains a BitcoinJ wallet. All exposed functions operate on top of this wallet.

The retrieval of arbitrary Bitcoin transaction data is enabled by the module *bitcoin-crawler*. It can be used to retrieve existing information about a Bitcoin transaction. The following figure illustrates the class diagram of this module.

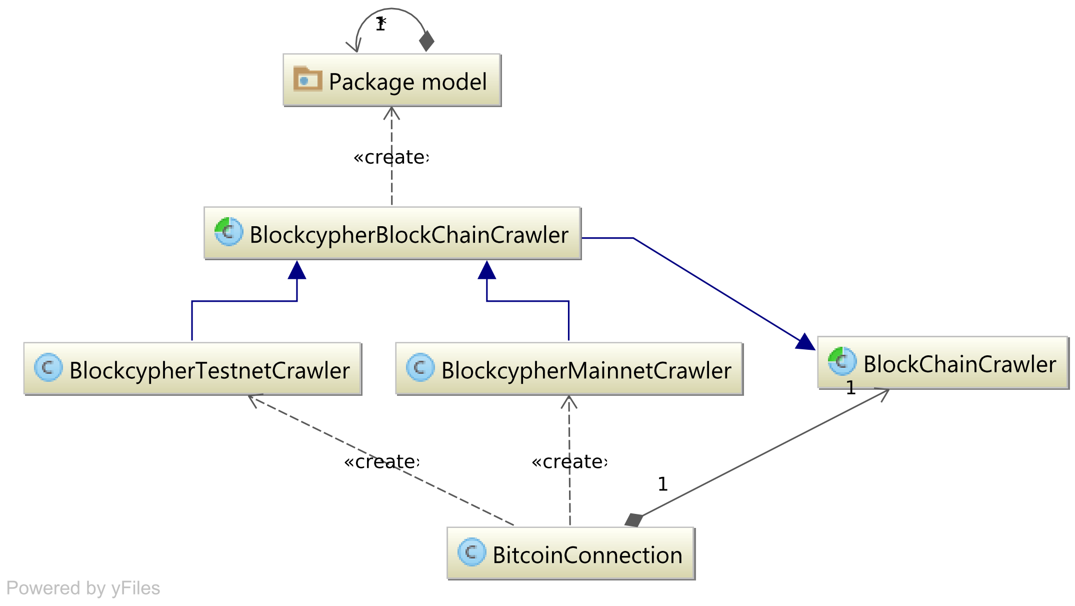


Figure 5.20: Class Diagram of Module *bitcoin-crawler*

The basic retrieval functionality is abstractly defined in the class *BlockChainCrawler*. As already mentioned, the prototype of this thesis relies on the REST API of `blockcypher.com`. The retrieval mechanisms required for the specific structure of this API are implemented in the class *BlockcypherBlockChainCrawler*. Since there is a distinction between the APIs operating on top of the *Bitcoin mainnet* and the *Bitcoin testnet*, there are two distinct configurations of this crawler. Some of the provided broadcasting functions from module *bitcoin-core* can be optionally used in conjunction with this information retrieval features. A transaction does not have to be re-published if it can already be found as a pending transaction by the information crawling module. Therefore, the class *BitcoinConnection* already relies on the functionality of this module.

The functions of the two previously described modules are utilized to enable the management of identity-aware workflow handovers. This runtime verification framework is finalized in the module *handoverFramework*. In the following Section 6 it is demonstrated how the described framework can be used during the execution of a workflow in a choreography-oriented WfMS environment. The structure of this module is illustrated through the class diagram in Figure 5.21.

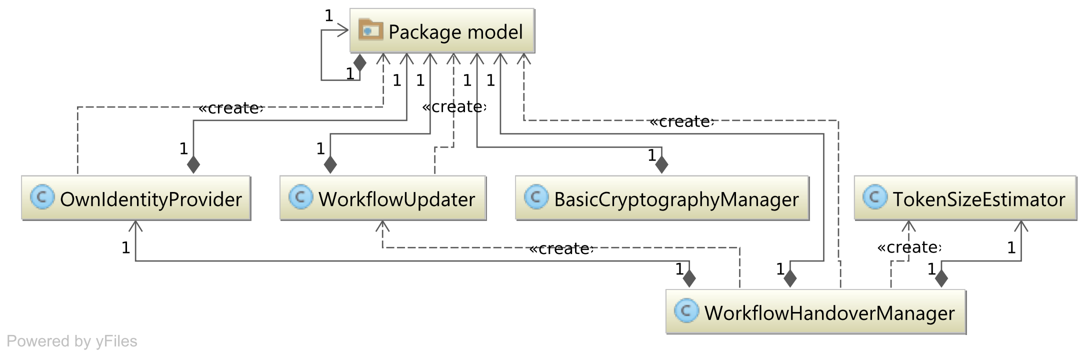


Figure 5.21: Class Diagram of Module *handoverFramework*

The class which combines all functionality exported by this framework is named *WorkflowHandoverManager*. It uses an instance of *BitcoinConnection* to operate a Bitcoin wallet and exchange Bitcoin transactions. Furthermore, it maintains a persistent *workflow graph storage*. This storage contains the workflow metadata known by this runtime verification instance. The stored metadata is used in conjunction with the Bitcoin transactions contained in the wallet.

The workflow graph stores and updates information about all started or received workflow instances together with available identity information about the involved participants. Following the proposed concept described in Section 5.3, a RSA-based PKI infrastructure is used to identify different actors in the choreography network. The class *OwnIdentityProvider* is used to provide the runtime verification framework with the RSA identity information of the operator of the currently running program instance. All cryptographic

PKI operations which are required during a workflow handover are provided by the class *BasicCryptographyManager*.

If a new workflow instance is started by the WorkflowHandoverManager, the class *TokenSizeEstimator* is used to estimate the required token size of the workflow instance. Since each transaction requires a fee which is deducted from the workflow token itself, the token must be sizable enough to suffice for the whole duration of a workflow execution. Based on the estimated number of tasks and splits the required token size is approximated. This estimation is performed very conservatively to ensure that the token will always be sizable enough.

The functionality, offered by the WorkflowHandoverManager, can optionally be performed in the described greedy publishing mode. This mode operates in contrast to the classic conservative mode, were each workflow transaction has to be confirmed before the workflow execution can be proceeded.

Before receiving the control over a workflow instance from another choreography participant, the class *WorkflowUpdater* uses a BlockChainCrawler to retrieve the published execution information of the workflow instance so far. Based on this information, a choreography-oriented WfMS is able to validate if this execution information conforms with the defined workflow model and if the handover can be accepted.

CHAPTER 6

Prototype Evaluation

Research challenge *RC-1* was addressed in the previous section. For this, different enactment verification approaches were discussed. We demonstrated that it is indeed possible to utilize a first generation Blockchain for runtime verification in choreography-oriented WfMSs.

In order to address *RC-2*, different functional and non-functional properties of the proposed prototype are evaluated in this section. Different characteristics related to the dynamic nature of distributed workflow choreographies are discussed in Section 6.1. The discovered results are then compared to the proposed prototype and other already existing approaches for runtime verification.

To assess the performance overhead created by the proposed framework, a performance analysis is conducted in Section 6.2. Similar to the performance evaluation conducted by [70], different workflow choreographies are simulated with and without runtime verification. This way an estimate about the framework's impact on the execution performance of workflow instances in a choreography-oriented WfMS can be calculated.

6.1 Functional Comparison

A comprehensive qualitative comparison of existing runtime verification approaches for WfMSs is a challenging task. As described in Section 5.2, runtime verification approaches are limited in number and of heterogeneous nature. Additionally as explained in Section 2.1, in the scientific field of choreography-oriented WfMSs there exist no established standards or architectures. Suitability of a given runtime verification approach for choreography-oriented WfMSs becomes difficult to assess. A functional comparison therefore has to focus on the few commonly agreed features of workflow choreographies.

Workflow choreographies operate as distributed systems. The more heterogeneous, geographically distributed and organisationally independent such systems become, the more dynamic and diverse they become. Many established solutions from orchestration-oriented WfMSs can not be applied in this context. Therefore, in choreography-oriented WfMSs many unique situations have to be addressed by fault management strategies [39, 58, 68]. For instance workflow participants might become unavailable, tasks might be processed incorrectly or network participants might compete against each other. In order not to limit these fault management strategies, a runtime verification system must remain as flexible as possible. In this thesis the term *flexibility* is therefore used to describe the capability of a runtime verification system to deal with the dynamic nature of a distributed choreography and its participants.

Choreography-oriented WfMSs and runtime verification approaches have been extensively discussed in the previous sections. Based on the findings, different relevant criteria have been extracted and are described in Section 6.1.1. Furthermore,the runtime verification approaches described in Section 5.2 are discussed in respect to these criteria. A similar kind of comparison approach has also been used by other scientific publications in the field of distributed software systems [8, 45, 88].

6.1.1 Extracted Flexibility Criteria

The following criteria which influence choreography-oriented fault management, choreography - oriented WfMSs and their participants in general have been extracted from the findings of this thesis. Depending on their structure and implementation approach, the different described approaches for runtime verification influence those criteria.

Participant Selection

A major influence on the stability of a distributed workflow instance is the selection of its involved participants [39, 68]. While often not directly discussed, predefining the participants greatly reduces the organizational effort during the enactment. At the same time, the workflow also becomes less robust. Depending on the nature of a distributed system, it is possible that participants might become unavailable. In a predefined setting, a workflow instance may halt in this case.

The alternative variant would be to select the required participants on-demand during the enactment. Especially for long running workflows where participants may have to wait a long time before they are involved, this can increase overall the robustness. Of course also a mixture out of the two approaches is possible. The different variants in participant selection are further illustrated in Figure 6.1.

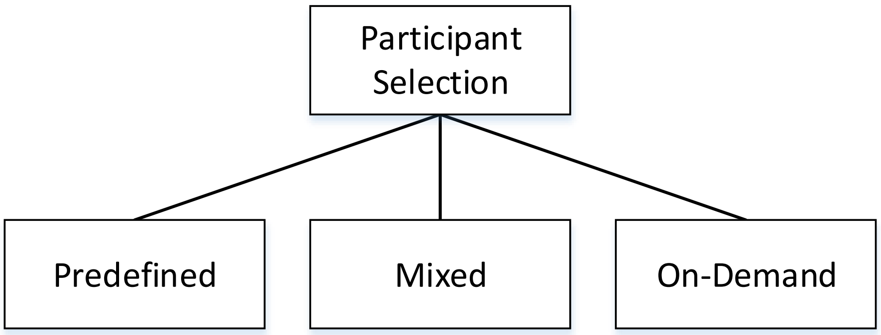


Figure 6.1: Variants of participant selection

Information Sharing

The categories outlined in this section influence the acceptance of the overall distributed system by the participants.

When choreography-oriented workflow enactment takes place as cooperation between independent companies, information sharing becomes an issue [13, 89]. Participants of B2B choreographies might also be potential competitors. It becomes a requirement to share as little information as possible with other participants. Therefore, it is important to evaluate if runtime verification approaches require companies to share internal information more than absolutely necessary. The less information is shared the more the system becomes acceptable for the independent participants.

The factor *information sharing* can be divided into three sub-factors as illustrated in Figure 6.2.

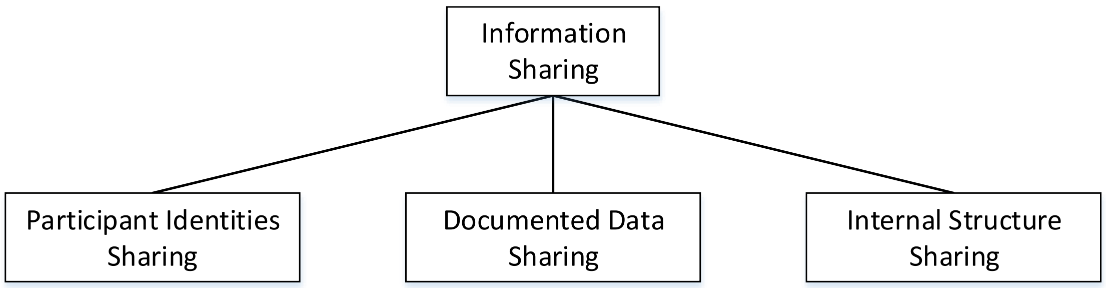


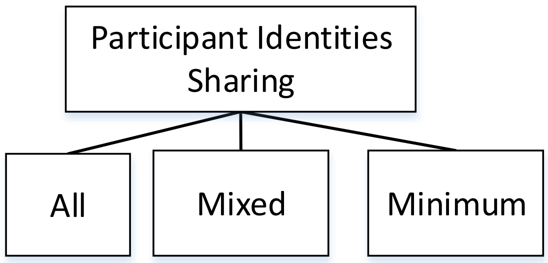
Figure 6.2: Sub-factors of information sharing

Participant Identities Sharing Cooperation requires companies to disclose identity information about themselves. But this information is only really required by the owner of a workflow instance and the direct interaction partners in a choreography. There is no actual necessity for choreography participants to know the identity of all other choreography participants.

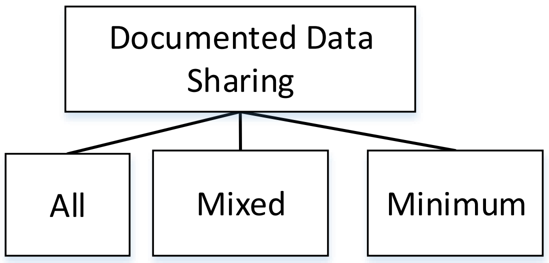
Documented Data Sharing Analogous to the previous sub-variant, it is not required that choreography participants have knowledge about all the data shared during a workflow enactment. Participants just need to know about the data shared with their direct neighbors.

Internal Structure Sharing While not desired, some runtime verification approaches may require participants to disclose parts of their internal structure. Therefore this sub-variant has to be highlighted.

The different variants of the sub-factors *participant identity sharing* and *documented data sharing* are illustrated in Figure 6.3.



(a) Variants of participant identity sharing



(b) Variants of documented data sharing

Figure 6.3: Participant Identity sharing and Documented Data sharing

As stated before, *internal structure sharing* is not desirable in a B2B setting but some approaches to runtime verification might require it. The different variants of this influence factor on flexibility are illustrated in Figure 6.4.

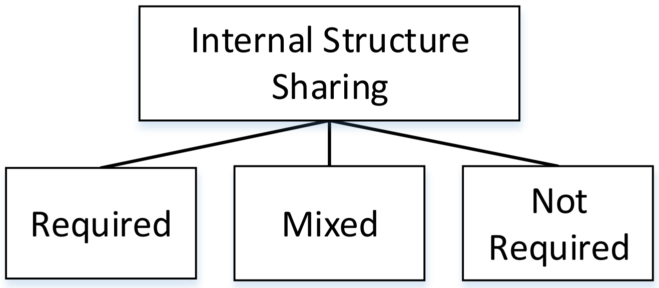


Figure 6.4: Variants of internal structure sharing

6.1.2 Exhibited Security Features

The previous section described criteria through which runtime verification proposals can influence the flexibility of choreography oriented WfMSs. Additionally, these runtime verification proposals are also expected to provide certain security-related features.

First and foremost a runtime verification framework should enable the discovery of incorrect enactment. It must not be possible for a participant to conduct incorrect behavior during the enactment of a choreography-oriented workflow instance without being noticed by the process owner or other process participants. In addition to this feature, runtime verification frameworks may be able to apply even stricter constraints. The enactment sequence of a workflow instance can be completely enforced. The different security features are illustrated in Figure 6.5.

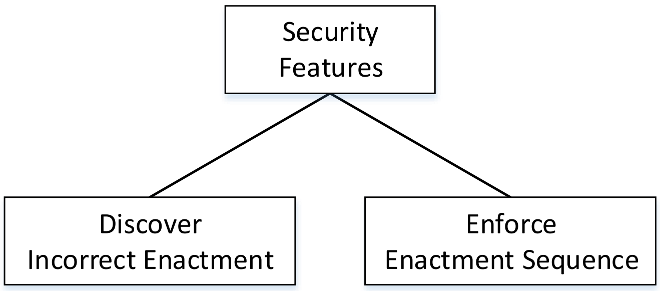


Figure 6.5: Possible security features exhibited by runtime verification frameworks

The different variants of the sub-factors *discover incorrect enactment* and *enforce enactment sequence* are illustrated in Figure 6.6.

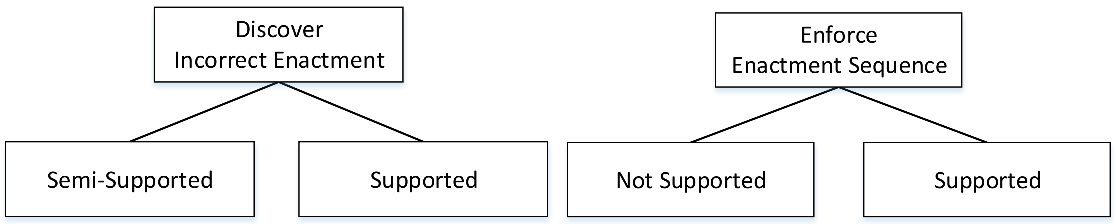


Figure 6.6: Variants of the sub-factors of the exhibited security features

The discovery of incorrect enactment can either be completely *supported* or *semi-supported*. Semi-supported means that there might occur cases where runtime verification can not be conducted. For each workflow instance enactment the process owner must receive an undeniable proof. Furthermore, the process owner must be able to monitor the current state of the enactment at any given time.

6.1.3 Application of Flexibility Criteria

The results of the functional comparison are listed in Tables 6.1, 6.2 and 6.3. Each approach provides different means to document the execution of a workflow execution. Still, none of the listed approaches limit the dynamic execution of workflow instances. Also, fault management strategies can still be applied.

In addition to the previously described criteria, the verification trust basis of each approach is listed. The verification trust basis is the component of the systems the participants of a choreography have to trust when performing verification. Afterwards, the individual result of each listed runtime verification proposal is briefly discussed.

Table 6.1: Criteria application to runtime verification approaches (part I)

Proposal	Participant Selection	Participant Identities Sharing
Bengtsson and Westerdahl [19]	On-Demand	All
Kuntze et al. [54]	Predefined	Minimum
Montagut and Molva [68]	Predefined	Minimum
Von Riegen and Ritter [89]	On-Demand	Minimum
Baouab et al. [13]	Predefined	Minimum
Lim et al. [60]	On-Demand	All
Hwang et al. [46]	On-Demand	Minimum
Weber et al. [91]	Predefined	Minimum
Thesis proposal	On-Demand	Minimum

Table 6.2: Criteria application to runtime verification approaches (part II)

Proposal	Data Sharing	Internal Structure Sharing	Fault Discovering
Bengtsson and Westerdahl [19]	All	Not required	Semi-Supported
Kuntze et al. [54]	Minimum	Required	Semi-Supported
Montagut and Molva [68]	Minimum	Not required	Semi-Supported
Von Riegen and Ritter [89]	Minimum	Mixed	Supported
Baouab et al. [13]	Minimum	Mixed	Semi-Supported
Lim et al. [60]	All	Not required	Semi-Supported
Hwang et al. [46]	Minimum	Not required	Supported
Weber et al. [91]	Mixed	Not required	Supported
Thesis proposal	All	Not required	Supported

Table 6.3: Criteria application to runtime verification approaches (part III)

Proposal	Sequence Enforcement	Verification Trust Basis
Bengtsson and Westerdahl [19]	Not Supported	Signature enhanced Token
Kuntze et al. [54]	Supported	Trusted Platform Module & Process Slip
Montagut and Molva [68]	Supported	Cryptographic Onion
Von Riegen and Ritter [89]	Supported	Enterprise Service Bus
Baouab et al. [13]	Supported	Message Interception Component
Lim et al. [60]	Not supported	Hierarchical Signatures
Hwang et al. [46]	Supported	DRA4WfMS Cloud Storage
Weber et al. [91]	Supported	Blockchain
Thesis proposal	Not Supported	Blockchain

6.1.4 Result Discussion

Discussion of Bengtsson and Westerdahl [19]

The approach of Bengtsson and Westerdahl [19] is a very simplistic one. In their scenario, document-based web service calls are used to enact a choreography. The XML encoded web service request is passed on between the participants of the choreography. Each participant appends its cryptographic signature to the document as proof for participation. Furthermore, each participant sends its intermediate result to the process owner through asynchronous requests.

While this proposal is very flexible, the runtime verification capabilities are very limited. The participant selection can be carried out on-demand, since nothing is predefined. Likewise the passed token is not encrypted, all previous participants are disclosed to whoever holds the token. Same holds true for the shared data. The element to proof the enactment and to perform verification on is the signature-enhanced XML document. The process owner receives this token at the end of the enactment and as intermediate results of the choreography. These results are sent asynchronously to the process owner. It is not guaranteed that these messages arrive at the process owner. A monitoring through the process owner is therefore also not guaranteed. The handover between participants is not explicitly documented. If the XML document is intercepted, a malicious participant can hijack the execution.

Discussion of Kuntze et al. [54]

The shared trust basis in the work of Kuntze et al. [54] are physical devices called *Trusted Platform Modules* (TPMs). In their scenario, each service must be situated on a hardware platform where such a device is installed. TPMs are able to carry out cryptographic operations. They are used to verify the operating service and incoming workflow requests.

The controlling element of a workflow instance is called a *Trusted Process Slip* (TPS). It is a cryptographic token that is passed along with the workflow instance.

Since the TPS must ensure that only authorized companies can participate, the participant selection must be done prior to the execution. The data included in the TPS is encrypted. Through the TPM it is ensured that only authorized participants can access instructions relevant to them. The identity of the participants and the processed data therefore does not have to be shared with all participants. Two major drawbacks of this approach are its invasiveness and its lack of monitoring. The TPMs have to be physically installed at each participant by a third party, requiring them to reveal information about their internal structure.

Furthermore, TPMs are only able to collect audit data locally. The process owner is not notified about the progress of its enacted workflow instances. The process model is included in the TPS. Only when the process slip is returned to the process owner, the path of the enactment can be determined. Upon receiving a workflow instance, the trusted process slip is verified and updated by the TPM. This way, a correct enactment of a workflow instance can be enforced while at the same time fallback strategies can be implemented.

Discussion of Montagut and Molva [68]

Montagut and Molva [68] propose a token passing concept, similar to [19] but in advanced form. Instead of a unprotected XML document, a token that is encrypted in multiple layers like an onion is passed along. The onion structure also is supposed to ensure the confidentiality and correct sequence of the workflow instance enactment. After each task execution, a choreography participant is supposed to report back to its preceding participant which in succession reports back to its preceding participant. This way, the process owner is notified about the progress of the workflow instance.

The process owner must create the token prior to the enactment. To create this cryptographic structure, keys must be distributed to all participants. Therefore the participants must be known in advance. The process owner must incorporate all possible paths, including fallback strategies into the onion. This way, the enactment course can remain dynamic within the boundaries of the process model. Depending on the complexity of the process model, this can significantly increase the size of the onion. Each layer of the onion can be accessed by the appropriate choreography participant. It contains the minimal information the participant requires to execute its work.

Since no additional framework is required beside the passed on token, no internal structure has to be shared by the participants. Likewise to the approach of [19], the concept of voluntary and asynchronous reporting back to the process owner is not a robust way of providing monitoring. When messages get lost or are not sent on purpose, the process owner is not able to discover faulty behavior. But all other participants, involved in the enactment are able to discover faulty behavior. The onion structure completely enforces the correct enactment of the workflow instance.

Discussion of Von Riegen and Ritter [89]

Instead of utilizing a token as the controlling instance, this approach suggests the mandatory usage of a shared communication medium. All independent organisations that want to participate in choreographies must connect to an enterprise service bus (ESB). Instead of directly connecting their offered web services with each other, these services are registered in the ESB. When this is done, all service communication is routed through the ESB. [89] further enhance the ESB with a atomic logging mechanism. All service calls are persisted to a shared logfile. This file can then be utilized to monitor and verify the behaviour of a workflow instance. Similar to the proposal of [60], other aspects of runtime verification are not discussed and must therefore be deduced.

While not explicitly stated, the usage of an ESB allows the dynamic allocation of services and resources. Therefore, it should be possible to select choreography participants on demand. Similar to the enhancement of atomic logging, an ESB can be enhanced to ensure confidentiality. Therefore participant identity sharing and data sharing is assumed to be minimal. The same holds true for the enforcement of the correct sequence for workflow instances. The main benefit of the approach is the atomic logging mechanism which produces a shared logfile. This logfile serves as proof of the workflow execution. By monitoring the logfile a process owner is notified about the progress of enacted workflow instances and can discover faults immediately.

Even though [89] claim that the ESB approach is very lightweight, it represents a somewhat invasive approach for the participating companies. The ESB connection software has to be integrated into the local infrastructure by each company in order to participate in a choreography. Furthermore, the ESB becomes the shared trust basis. While not explicitly stated, the ESB itself must also be maintained and operated. Effectively, the third party that provides the ESB becomes the shared trust basis.

Discussion of Baouab et al. [13]

Similar to the previous approach, Baouab et al. [13] propose a message controlling facility. The published services of each participant are masked with a message filtering component. Based on different message and security policies the component decides whenever messages should be filtered, reordered or routed. If required, notifications to other participants are sent through a event publishing facility not unlike CEP.

The proposal strongly emphasizes the loose coupling of the companies. Therefore, instances of the message controlling facility are only deployed on the infrastructure of each participant. No shared infrastructure is utilized for the message control facilities. Since no global state is exchanged, the process owner has to define and distribute the requirements of a workflow choreography prior to the enactment. These rules and requirements are referred to as *policies*. Therefore, the participants must be selected beforehand. Since the policy generation and distribution is entirely controlled by the process owner, the participant identity sharing and data sharing can be kept to a minimum.

Similar to the approach of [89], the message interception component must be integrated into the facilities of each participant. It is not specified if the message interception component is integrated by the participants themselves or a third party. If the participants are able to integrate the message interceptor themselves, no internal structure must be shared. If not, they must disclose parts of their internal structure to a third party.

The communication to other participants (i.e. also the process owner) is handled through event based messaging. This is an asymmetric communication variant. It is possible that published events are intercepted or lost, therefore the fault discovery is only semi-supported. Through the defined policies the process owner is able to define and enforce the correct execution sequence of the workflow instances. The trust basis in general is the message interception component. Since this component must be utilized by all choreography participants, it is likely to be provided by a third party.

Discussion of Lim et al. [60]

While not covering the entire verification process, this proposal aims to use special hierarchical signatures to document the progress of a workflow instance. These signatures allow the usage of multiple keys to create a signature, thus enabling process participants to document the progress of a workflow accordingly. When the execution of a workflow is joined from multiple parallel paths, the keys of all directly preceding companies can be used to document their approval.

The concept of Lim et al. [60] strongly focuses on the mechanics of those hierarchical signatures and does not discuss other aspects of runtime verification in detail. Similar to the concept of [19], the usage of signatures does not require the selection of the choreography participants prior to the enactment. As stated, the issues of confidentiality are not addressed. Therefore, all previous participants are known to anyone who holds the token and all shared data is visible.

The lightweight approach that utilizes only signatures does not require the participants to share information about their internal structure. Like all other previously described token-based approaches, fault discovery is only semi-supported. That means the process owner or anyone else can discover faulty behavior, if the token element is returned. In case the token gets lost or intercepted, no runtime verification can be performed.

Discussion of Hwang et al. [46]

In an attempt to make the choreography-oriented execution of a workflow instance independent of the participating companies and their local WfMSs, Hwang et al. [46] propose a cloud-based approach. The entire execution state of a workflow instance is encapsulated in one single XML file. This file is passed along the participants like a token.

At the same time, a cloud-based control instance, called *DRA4WfMS Cloud*, is used to repeatedly store and monitor the passed tokens. Through various cryptographic access restrictions it is ensured that all participants can access and alter only data relevant to them. Furthermore, this way the document is secured in the untrusted cloud environment.

Each time a participant is finished with executing a task, the token is returned to the cloud system. The system performs verification and stores the document. Furthermore, subsequent participants are notified.

At first glance, the approach of [46] combines all the benefits. Through the regular returning of the control token to the cloud system, the participants can be selected on demand. The identities and data of the involved participants must only be shared minimally. The internal structure of each participant must not be disclosed, since the entire workflow is controlled through one encryption protected XML file.

Faults can quickly be discovered though the central monitoring of the cloud system. At last, since routing is also performed by the cloud system the correct enactment sequence of the workflow instance can be enforced.

The flaw in the concept is the shared trust basis. While being referred to as a scalable and trustworthy approach, the controlling cloud system DRA4WfMS effectively becomes the new centralized coordinator. This does not fit to the definition of decentralized control in workflow choreographies. While being operated on a scalable cloud platform the routing system DRA4WfMS becomes the new bottleneck and single point of failure. Furthermore, the entire encryption is managed by the system. Therefore, all participants have to trust this third party to correctly manage their initiated workflow instances.

Discussion of Weber et al. [91]

A combination of the benefits of both token-based and messaging control-based approaches is attempted by a new kind of strategy, the employment of Blockchains. The approach of [91] is described in greater detail in Section 5.2.4. A token-like object is used to protocol the state changes of a workflow instance. But instead of passing this token along with the participants, the token is stored in the Ethereum Blockchain as a smart contract. As the state of the contract is altered by the participants, the contract changes its access restrictions, thus controlling the control flow of the workflow instance.

In order to include access keys during the creation of smart contracts for workflow instances, all choreography participants must be known prior the workflow enactment. On the other hand, this way the identities of the participants can be protected from each other. The participants do not directly interact with each other, instead the input and output data is shared through the state management transactions of the smart contract. Furthermore, this payload is encrypted to protect it, thus keeping on-chain data sharing to a minimum.

Data-intensive tasks are supposed to be handled by trigger components. These components are under the control of single choreography participants and must be trusted. Therefore, the required data sharing is classified as *mixed*.

Since the entire management of the workflow takes place in the Blockchain, nothing about the internal structure of the participants must be shared. The progress of a workflow instance is publicly documented in the Blockchain as its corresponding smart contract is altered. Through this, a process owner can monitor the progress of a workflow instance the entire time. The correct sequence of a workflow is enforced by the control logic of a smart contract.

Discussion of own proposal

The runtime verification prototype of this thesis is discussed in detail in Section 5.3.

Due to its open design, the participant selection of this thesis's proposal can be done on-demand. Since each next participant is selected by the current token owner, the identities of all participants do not need to be shared. Only the process owner, who is able to collect the workflow information pull-based is able to know the identities of all participants.

Data sharing is not handled in the current design of the proposal. The passed along data is visible to all participants of the choreography. Similar to the approach of [91], the utilization of a Blockchain externalizes the used trust basis and does not require the participants to share information about their internal resource structure.

To perform correct handovers for workflows, both the sending and the receiving participant must provide their signature. Each new task that is performed is documented in these handover transactions. The process owner and all other participants can monitor the Blockchain in order to analyze the execution path of a workflow instance. Incorrect behavior immediately becomes visible to all choreography participants, including the process owner. Due to the fact that the process model of a workflow is not incorporated in the logic of a workflow transaction, the sequence of performed tasks can not be enforced.

6.1.5 Summary

The perfect approach for runtime verification in choreography-oriented WfMSs has not yet been found. Many approaches utilize tokens that are passed along with the participants during the enacment in order to document and/or controll the progess of a workflow instance [19, 46, 54, 60, 68]. In token-only approaches, participants have to be predefined to subsequently protect the privacy of identity and data. Furthermore, the enactment sequence can only be enforced this way. All token-only approaches suffer from limited fault discovery. Since the token is the only element of proof that work was done, the communication of this token is critical.

But there is no mechanism in place that guarantees that a token is returned to the process owner. It may get intercepted or lost. Furthermore, a process owner is only able to monitor the progress of a workflow instance, if the intermediary token is returned periodically.

Other approaches aim to control the messages that are passed between the participants [13, 46, 89]. To enable this, different communication and control facilities have to be established. By utilizing this approach the proposals of [89] and [46] are able to exhibit very flexible features. On the downside the facilities introduced by those approaches become the shared trust basis. This new shared trust basis is under the control of a third party. The same partially holds true for the approach of [13] but they explicitly address this tight coupling issue. In their approach they yet again trade flexibility in participant selection and fault discovery in order to provide a somewhat more decentralized and trustworthy message controlling facility.

Recent approaches try to overcome the issue of the shared trust basis by incorporating a Blockchain. [91] utilizes the Ethereum Blockchain and the proposal of this thesis utilizes the Bitcoin Blockchain. Through this, both approaches are able to provide complete fault discovery and to fully protect the internal technical structure of the participants. At the same time the respective Blockchain serves as secure, independent and decentralized basis of trust.

The selected Blockchain has great impact on the exhibited features of a Blockchain-based approach. While both Blockchain-based approaches protect the identities of their participating companies, they differ in their participant selection, data sharing and sequence enforcement.

[91] requires the participants to be selected in advance. This way, [91] are able to partially protect the confidentiality of the protected data and to enforce the execution sequence of the enacted workflows. The approach of this thesis exhibits greater flexibility by enabling an on-demand participant selection. On the other hand, our proposed approach is not yet able to provide confidentiality for the shared data and also the correct execution sequence can not be enforced. Nevertheless, the work of Weber et al. [91] comes closest to this thesis.

6.2 Performance Analysis

[70] conducted a performance analysis for software applications by comparing the execution time of an application with and without a specific feature included. They executed workflow applications within computational Grids. In order to approximate the imposed time overhead of the Grid, they compared the execution time of the workflow applications within the Grid against an idealized model for the execution time.

Similar to the described approach of [70], this thesis assesses the impact of the runtime verification feature on the execution time of enacted choreography-based workflow instances. The workflow instances are executed with and without the proposed runtime verification prototype included. As discussed in Section 2.1, the research about choreography-oriented workflow systems is still in its infancy. That is why there is a lack of mature choreography-oriented workflow systems that can be used as foundation for the performance analysis. As a compromise, the routing mechanics of a choreography-oriented WfMS are custom built instead. The remainder of this section is organized as follows. Section 6.2.1 describes the approach, design and implementation of this choreography enactment simulation as well as the simulated workflow instances. The results are outlined in Section 6.2.2.

6.2.1 Simulation Setup

A choreography is conducted by various individual participants. According to the motivational scenario defined in Section 4, that is addressed in this thesis, the various participants all represent independent companies. Each company operates an internal WfMS, which can be either orchestration-oriented or choreography-oriented. The cooperation of the WfMSs of these companies represents the actual choreography.

Choreography Participants

As a simplification, the different available companies are represented as a collection of semantically isolated routing endpoints implemented in Java. Each company operates its own choreography logic and runtime verification instance (i.e. instance of WorkflowHandoverManager). The runtime verification instance itself operates on top of a designated SPV Bitcoin wallet. As explained in Section 5.4, the runtime verification prototype is designed to be integrated in an existing choreography-oriented WfMS. Instead of a full fledged WfMS, the prototype now is integrated into the custom build choreography logic.

Since a mature choreography-oriented WfMS implementation is unavailable, a choreography logic was custom built for the simulation. It is capable of handling basic business process models as described in the following section. Using this choreography logic, companies are able to start the enactment of their corresponding workflow instances. Furthermore, each company listens to a specified port, waiting for incoming handover requests of other companies. When a correct workflow handover is received, the execution of the next task of the workflow is simulated. Since an actual execution of a task is not required in the course of this analysis, each task execution is only represented by a waiting period of 5 seconds. After the simulated execution, the next handover partner is selected and the workflow instance is passed on. This way, a workflow instance is passed along the choreography participants through TCP/IP socket communication.

Each company is strictly isolated. It maintains its own RSA key-pair to support the cryptographic steps of the workflow handovers. Furthermore, only specific information is publicly available for every company. Namely the contact ports of companies to send workflow handover requests to, the public RSA key to exchange signed and encrypted information and the used business process models together with the execution paths defined by the simulation.

The described components of the simulated companies are further illustrated as FMC Blockdiagram [47] in the following figure.

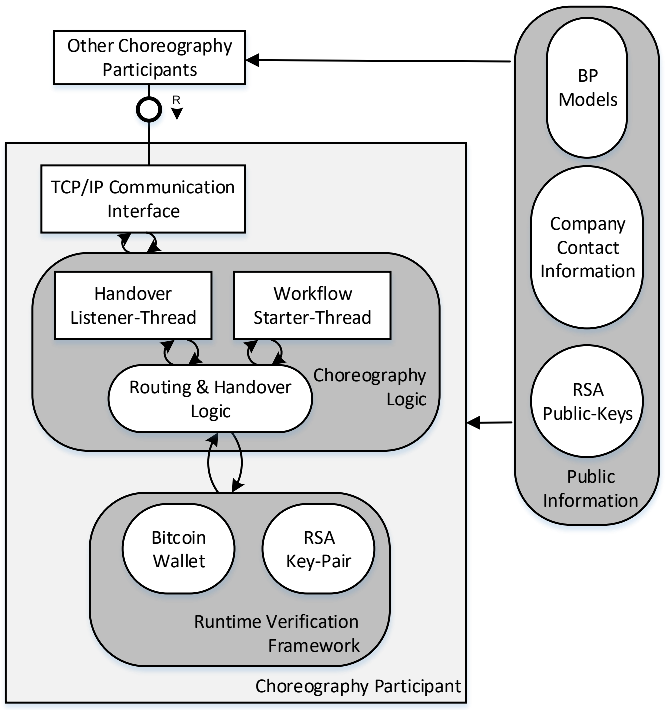


Figure 6.7: FMC Blockdiagram of the basic components of a choreography participant in the simulation

In order to document the course of the choreography enactment, all companies log their progress to a shared logfile.

Simulated Business Processes

In addition to a custom routing logic, a custom interpretable representation of business processes must be defined. This custom choreography logic controls the routing of rudimentary business processes. The following well-known BPMN-like elements can be used to define basic processes.

- Process Start
- Process End
- Activity
- XOR Split- and Join-Gateway
- AND Split- and Join-Gateway

The elements *Start* and *End* mark the beginning and end of a business process. In our simplified setting only one Start- and End-element is allowed per process model. The *Activities* depict the actually execution of tasks. Each activity is associated with a task-name and task-id. Activities also may require input data, produce new output data or change the given input data. *XOR gateways* mark necessary decisions that have to be made during the execution. The outcome of these decisions defines on which process branch a workflow instance proceeds. Parallel execution is enabled through *AND gateways.* These gateways simply split or join the execution of a workflow into branches that can be executed in parallel. By utilizing the BPMN notation [72], these elements can be illustrated in the following way.

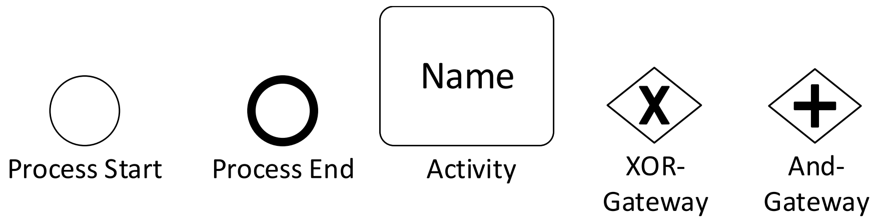


Figure 6.8: Business process elements used in the simulation, in BPMN notation

Using these elements, basic business processes can be defined. During choreography-based workflow execution, a process owner initially hands over the enactment of a workflow instance to a partner company. Without a runtime verification framework, as soon as this partner company passes the workflow instance on to another choreography participant, the process owner has no more knowledge on who is currently enacting the initiated workflow instance. Therefore, any business process which contains at least two activities is a suitable candidate for the runtime verification system. To maximize the verification effort of the prototype, in the simulation each activity is enacted by another company. Therefore for each activity a handover is required. The following four business process models, illustrated in BPMN notation, are used as basis for the simulation.

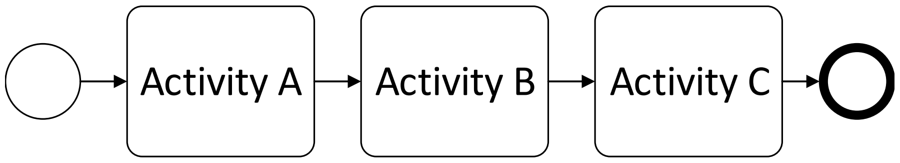


Figure 6.9: Business process model for simulation, named *Simple BP*

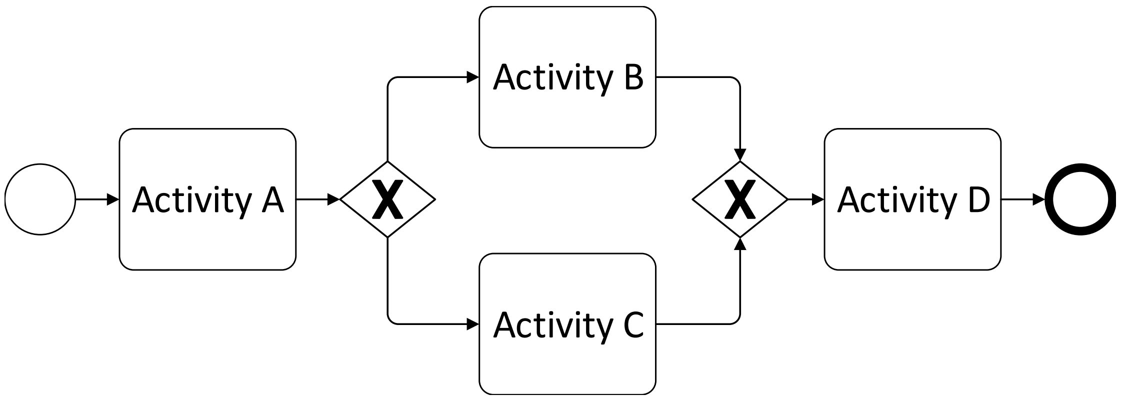


Figure 6.10: Business process model for simulation, named *XORSplit BP*

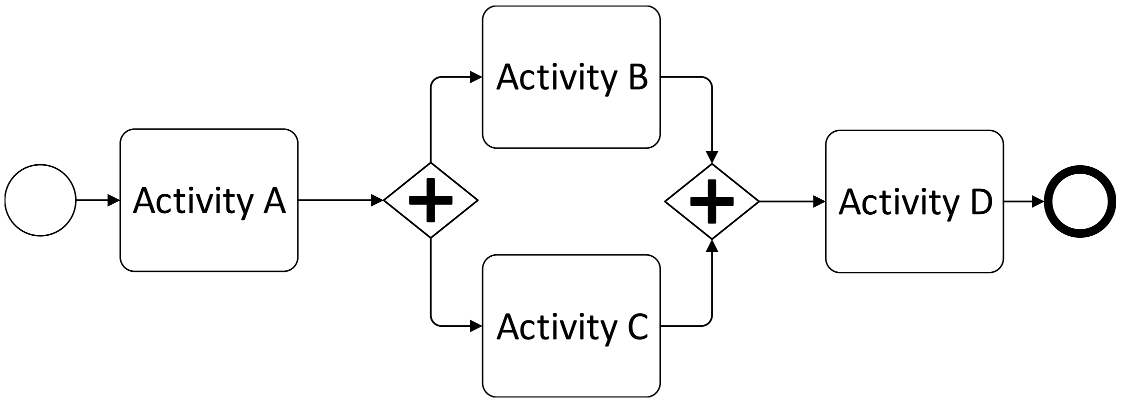


Figure 6.11: Business process model for simulation, named *ANDSplit BP*

The three previous business process models combine the described elements only in a sequential way. But as the following model illustrates, the elements can also be combined in multiple layers.

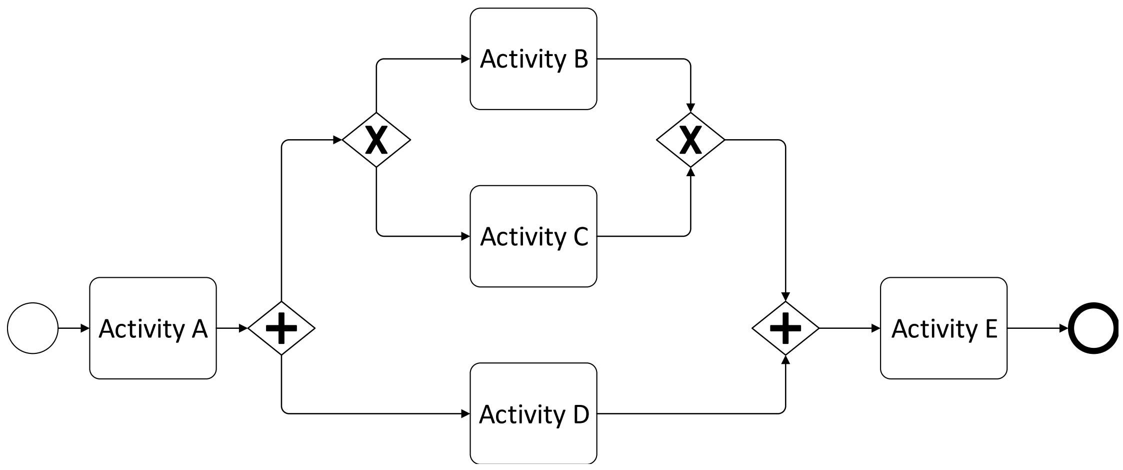


Figure 6.12: Business process model for simulation, named *Layered BP*

Even though the business process models are illustrated by using the BPMN notation, inside the simulation program they are directly defined in the code. Since there exits no mature choreography-oriented WfMS that can be utilized for the simulation, the hard-coded definition of the business process models is the simplest solution.

Simulation Moderation

In order to start, moderate and end a single simulation run, a Java program operates as the main controlling entity. It expects a number of input parameters to configure the simulation as well as a valid *Spring Application Context* that defines the different available companies as described above.

While there is a lot of simulation data which is generated during the start of the simulation by utilizing a random number generator, the simulation is kept deterministic by its input parameters. Amongst other things, a seed is used to initialize the random number generator. These conditions are required to keep the simulation reproducible. When a simulation is executed twice using the same input parameters, the deviation of the results will only be determined by the given behavior of the Blockchain. The following input parameters are expected by the simulation.

Testnumber Number to uniquely distinguish the current test configuration from other configurations. Also used to uniquely name the produced logfile.

Business process model name Business process model to enact, as described above.

Execution variant number Whenever a XOR gateway is used in a process model, different execution variants are created, depending on which branch gets selected for the further enactment. By using this variant number as input, the selected paths of the XOR gateways can be predetermined.

Include fault in process flag The most basic purpose of the proposed runtime verification framework is to prevent or highlight incorrect workflow executions during a choreography. By following the chain of workflow transactions in the Blockchain, a process owner can determine if the execution paths still conform to the defined business process model. In addition, each potential choreography participant can also collect and verify the execution trace of a workflow instance before accepting a handover. The given input parameter flag defines if one of the participants in the simulation should attempt to perform an incorrect handover.

Random number generator seed As already mentioned, the results of the used random number generators are kept deterministic by employing a seed for the initialization.

Use runtime verification flag In order to establish an optimal baseline a simulation can also be run without the runtime verification framework. This baseline can then be compared to the test runs that utilized the framework. Through this comparison the impact of the framework on the execution time can be assessed. The given input parameter flag determines if the framework should be used in the given simulation run.

Run in greedy publishing mode flag As explained in Section 5.4.3, the implemented prototype can be configured to greedily publish chains of workflow transactions to the Bitcoin P2P network without waiting for confirmation first. Whether this publishing mode should be used or not is controlled by this input flag.

Company set to use In order to enable parallel simulation runs, different sets of independent company configurations are defined. As long as there are enough funds available, different simulation runs can be executed in parallel. Which configuration set should be used for the current simulation run is defined by this input parameter.

Index of company with enough money One company in a configuration set must have enough funds in its wallet to start the Blockchain documentation of a workflow instance. This input parameter defines which company should be chosen to start a given workflow instance in the current simulation run.

Bitcoin network to use Bitcoin operates a *testnet* and *mainnet* Blockchain, as described in Section 2.3. Which net should be used in the simulation run is defined by this input parameter.

The companies along with their included runtime verification frameworks are defined as beans in Spring Application Contexts. As described, there exist different independent sets of companies. Each set is configured as one consistent application context. Determined by the input parameter *Company set*, the simulation loads a specific Spring Application Context (i.e. company set). This step also immediately loads the wallets which are included in the runtime verification framework components of the companies.

After the companies have been loaded through the application context, the business process model to be simulated is generated according to the parameter *business process model name*. To enforce deterministic behavior, this business process model is further enriched with execution metadata according to the input parameters *execution variant number*, *fault in process number*, *random number generator seed* and *index of company with enough money*. Amongst other things it is predefined which path should be taken through the business process model, which companies are responsible to execute specific tasks or what workflow data is produced by certain tasks. The general business process model as well as the specific execution instructions are transferred to the companies.

After this initial generation of testdata, the configured companies are instructed to spawn the required execution threads. While the companies enact the given workflow instance, the main Java program keeps monitoring their progress. As soon as all companies have finished the enactment of the process instance, a final verification of the documented workflow traces against the expected execution paths is performed.

All relevant data of a given simulation run is logged to a file. The documentation files of all recorded simulation runs provide the basic data which is further analyzed in Section 6.2.2.

The structure of the simulation program, described in this section, is further illustrated in Figure 6.13 as FMC Blockdiagram.

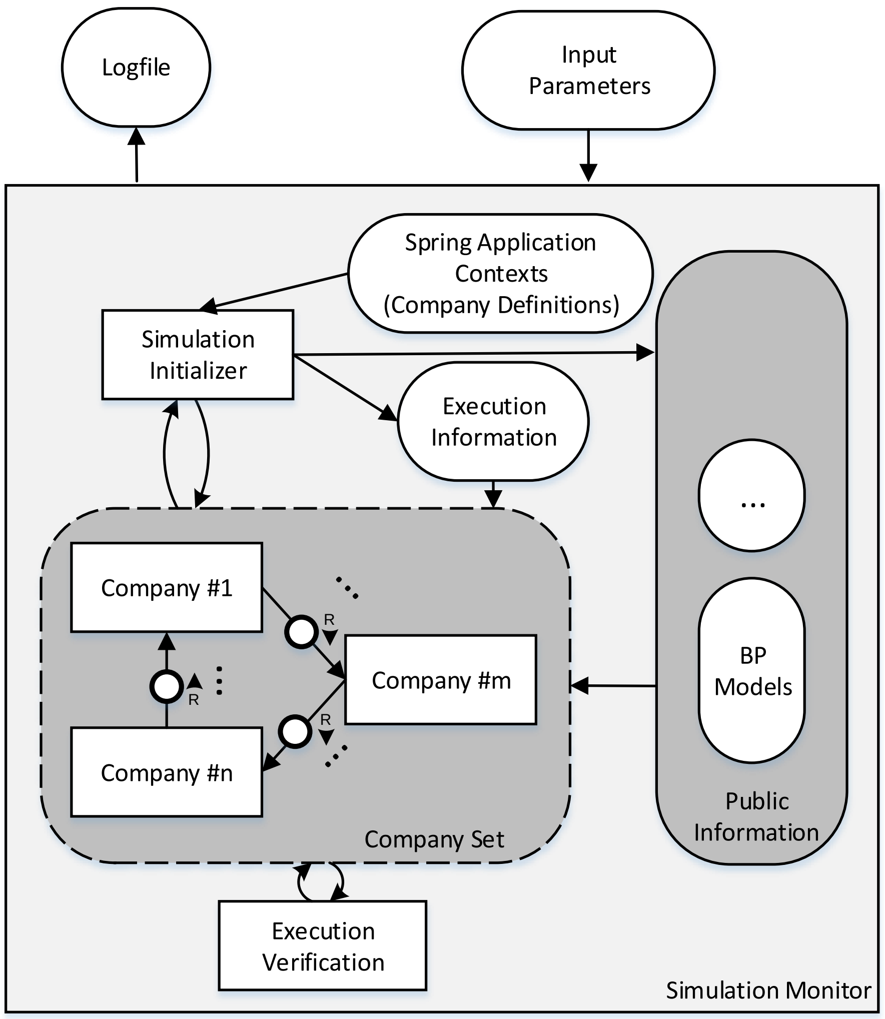


Figure 6.13: Blockdiagram of the simulation monitor component

Prototypical Limitations

As mentioned before, due to the prototype state of the proposed runtime verification framework a number of technical limitations remain in the implementation. As described in Section 5.3, only during the publishing of a workflow-handover transaction the control over the workflow token is passed on. During the publishing of the other workflow transactions, the token remains under the control of the same workflow participant. This implementation approach was not a conceptual necessity but a result of simplification reasons. Unfortunately, simplification leads to some difficulties during the enactment.

For instance, a workflow must only have one start- and end-node. Furthermore a workflow must be ended by the same company (i.e. process owner) it was started by. Since the end-workflow transaction can not be used to transfer token ownership, the token must be under the control of the process owner prior to publishing. When the last activity of a business process is not enacted by the process owner, the token must be transferred to the process owner in an extraordinary workflow-handover transaction. In the simulation these extraordinary transactions are called *filler tasks*. They do not simulate any task and serve only as a mechanism to transfer token ownership. Due to this compromise it is possible that a simulation may incorporate more workflow-handover steps as defined in the business process model.

Technical Structure

As already described, the Simulation is implemented as a Java program. It includes rudimentary choreography-oriented routing logic and utilizes the runtime verification prototype through the class *WorkflowHandoverManager* of module *handoverFramework*.

To enable the definition of basic business processes which also can be used as routing instructions of workflow instances, the simulation defines a custom model. This model incorporates different variants of the class *BusinessProcessElement*. These variants correspond to the business process elements described above. Compositions of different business process elements are stored as instances of the class *BusinessProcessDescription*. To further enhance this business process descriptions with execution metadata they are stored as instances of the class *ExecutionPath*. The following figure illustrates the class diagram of the model.

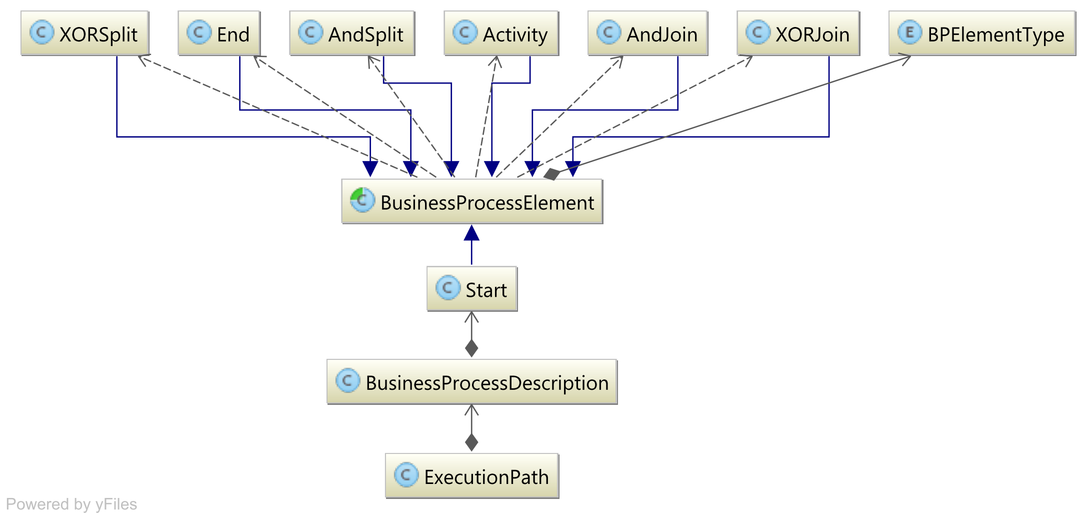


Figure 6.14: Class Diagram of the custom business process model

The simulation software is structured accordingly to the simulation descriptions above. The main simulation moderation is provided by the class *Simulator*. This main class utilizes simple helper classes from the package *dataGeneration* to generate the required simulation data. The previously described business process definitions reside in the package *model*. Companies together with their multi-threaded logic are represented by the class *SimulationAgent*.

The logic to start workflow instances resides in the class *StartClientThread*. Each simulation agent (i.e. company) listens for incoming handover requests by utilizing the class *BitcoinRuntimeVerifierServer*. The class *BitcoinRuntimeVerifierHandoverClient* contains the logic to hand over a workflow instance to another simulation agent. The class *ServerConnectionThread* contains the logic to receive a workflow instance from another simulation agent. The public information which is available for all simulation agents is provided by a number of shared data storages which are defined in the package *sharedStorages*.

At last during the handover of a workflow instance and at the end of each simulation, the recorded workflow execution traces are verified against the defined business process models. This logic is provided by the class *SimulationExecutionVerification*. The following figure illustrates the class diagram of the simulation.

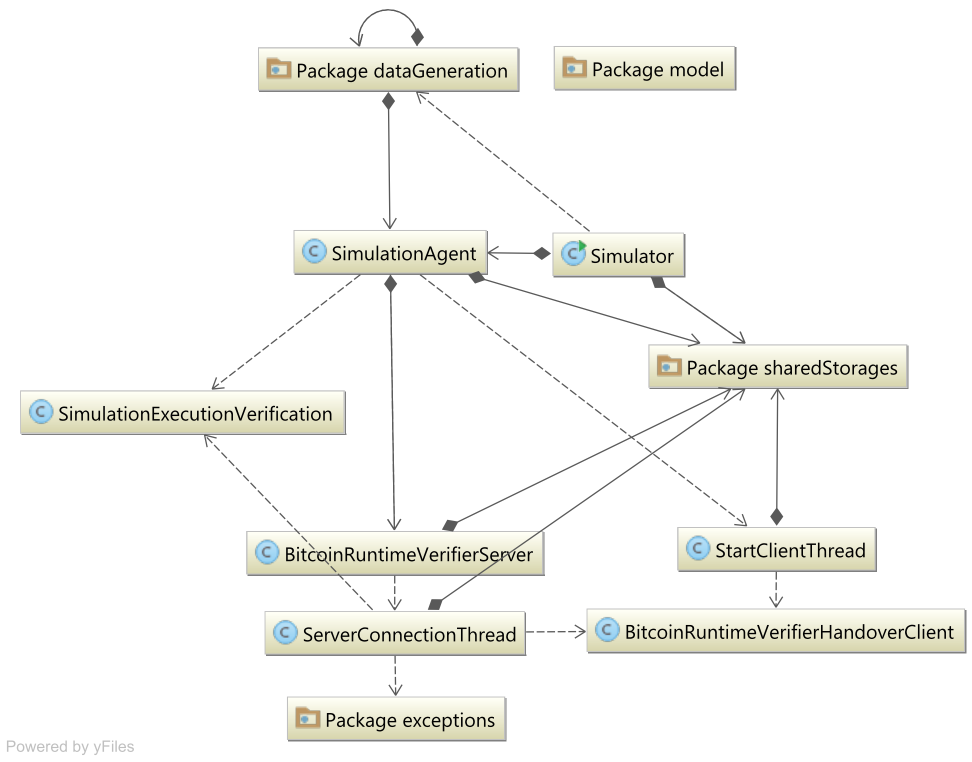


Figure 6.15: Class Diagram of the Simulation

6.2.2 Simulation Result

In order to conduct the simulation, a pool of test data is defined which contains different variations of the test parameters described in the previous section. This test data can be viewed in Section 6.2.3. The course of these simulation runs is documented in logfiles. The analysis of these logfiles enable the comparison of test runs which use the runtime verification framework and test runs which do not. In addition to that, it can be determined if incorrect handover are indeed detectable through the runtime verification framework. At last, the simulation highlights the difference in transaction throughput when the proposed greedy publishing mode is used.

In total 122 workflow instances were enacted through the custom choreography-oriented WfMS described in Section 6.2.1. A list with the transaction hashes of the workflow instances starting transactions can be found in the appendix. The workflow instances which used the runtime verification prototype published a total of 450 transactions enriched with workflow metadata to the Bitcoin Blockchain. The total execution time of all workflow instances amounts to 50.551 hours. The execution time of workflow instances include waiting periods for transaction confirmations. Each published transaction must reach at least a confirmation depth of 1 before a workflow enactment is considered finished.

Approximately 0.085417 BTC were spent to fuel the transactions of the simulation. Given the currency exchange rate of Bitcoin to Euro from 08.31.2016, one Bitcoin amounts to € 512.8969 [1]. The cost to fuel the transactions in fiat currency therefore amounts to approximately € 43.81. Considering that 450 transactions were published during the simulation, the average fee of a Bitcoin transaction enriched with workflow metadata results to 0.000189816 BTC or € 0.09735581.

In order to create a baseline for the impact of our proposed runtime verification prototype, the business process models described in Section 6.2.1 were enacted as choreographies without using the runtime verification framework. Table 6.4 illustrates the resulting median execution times of the different test runs.

[1] `http://api.coindesk.com/v1/bpi/historical/close.json?currency=EUR&start=2016-08-31&end=2016-08-31`

Table 6.4: Verification-less workflow test runs

Test Number	Business Process	Activities Covered	Test Runs	Average Duration [s]	StdDev (σ) Duration [s]
1	Simple BP	3	3	15.544	0.074298198
4	Simple BP	2	3	10.468	0.012283684
11	XORSplit BP	3	3	15.510	0.047590849
12	XORSplit BP	3	3	15.553	0.130349104
14	XORSplit BP	1	3	5.016	0.001247219
19	ANDSplit BP	4	3	18.016	0
20	ANDSplit BP	4	3	18.016	0
21	ANDSplit BP	3	3	13.021	0.007318166
27	Layered BP	4	3	15.592	0.048689492
28	Layered BP	4	3	15.548	0.015369523
29	Layered BP	4	3	16.570	1.399602166
30	Layered BP	4	3	15.539	0.023338095

These results of verification-less test runs serve as baselines. Without the runtime verification framework included the duration of the test runs is very consistent. In comparison to the average duration, the standard deviation is very small. These baselines are compared to test runs of similar configuration but with the runtime verification framework included. Test runs which utilize the runtime verification prototype are repeated more often. They are run with and without the proposed greedy mode enabled. It is expected that these test runs exhibit a higher standard deviation (σ) due to their dependency on the Bitcoin Blockchain. The resulting average execution times of the runtime verification-based test runs are illustrated in Tables 6.5 and 6.6.

Table 6.5: Verification framework workflow test runs in non-greedy mode

Test Number	Business Process	WF Steps	Test Runs	Total # of Tx	Average Duration [s]	StdDev (σ) Duration [s]
3	Simple BP	6	2	12	5,501.125	936.20350
8	XORSplit BP	6	4	24	3,489.367	1,110.33008
15	ANDSplit BP	10	4	40	7,835.141	2,309.38215
25	Layered BP	10	4	40	9,973.858	6,135.97428
26	Layered BP	9 - 10	4	37	7,537.909	3,443.67078
32	Layered BP	9	4	36	9,591.835	3,106.87162
			Total	189		

Table 6.6: Verification framework workflow test runs in greedy mode

Test Number	Business Process	WF Steps	Test Runs	Total # of Tx	Average Duration [s]	StdDev (σ) Duration [s]
2	Simple BP	6	4	24	706.864	429.716
5	Simple BP	3	6	18	650.132	944.669
7	XORSplit BP	5 - 6	4	21	710.946	306.177
10	XORSplit BP	4	5	20	4,050.133	7,250.916
16	ANDSplit BP	9 - 10	4	39	541.850	221.299
23	Layered BP	10	3	30	2,829.985	2,039.841
24	Layered BP	11	5	55	1,154.829	645.618
			Total	207		

In order to demonstrate that the proposed runtime verification framework is capable of detecting errors, a number of workflows include corrupted handovers. Upon noticing such incorrect behavior the enactment of the workflow is stopped prematurely. That is why not all test runs are suited for the overall time impact comparison of the runtime verification prototype. Therefore not all 32 test run configurations are listed in Tables 6.4, 6.5 and 6.6. Therefore, also the number of transactions in Tables 6.5 and 6.6 do not amount to the total of 450 submitted transactions.

A quick analysis of the tables makes it obvious that the usage of the runtime verification framework significantly increases the duration of the enacted workflows. But also workflows that do not run in the proposed greedy mode exhibit a significant longer duration in comparison to the ones that do.

The following section analyses the results of the runtime verification framework without the greedy mode in greater detail. After this, the impact of the runtime verification framework based in conjunction with the greedy mode is also further analyzed.

Non-Greedy Mode Results

Workflows that were enacted using the runtime verification framework in the non-greedy mode exhibited the highest increase in execution duration. The duration difference per business process of the baseline results and the non-greedy runtime verification results are illustrated in the following bar diagram. Note that the y-axis is plotted in $\log_{10}$.

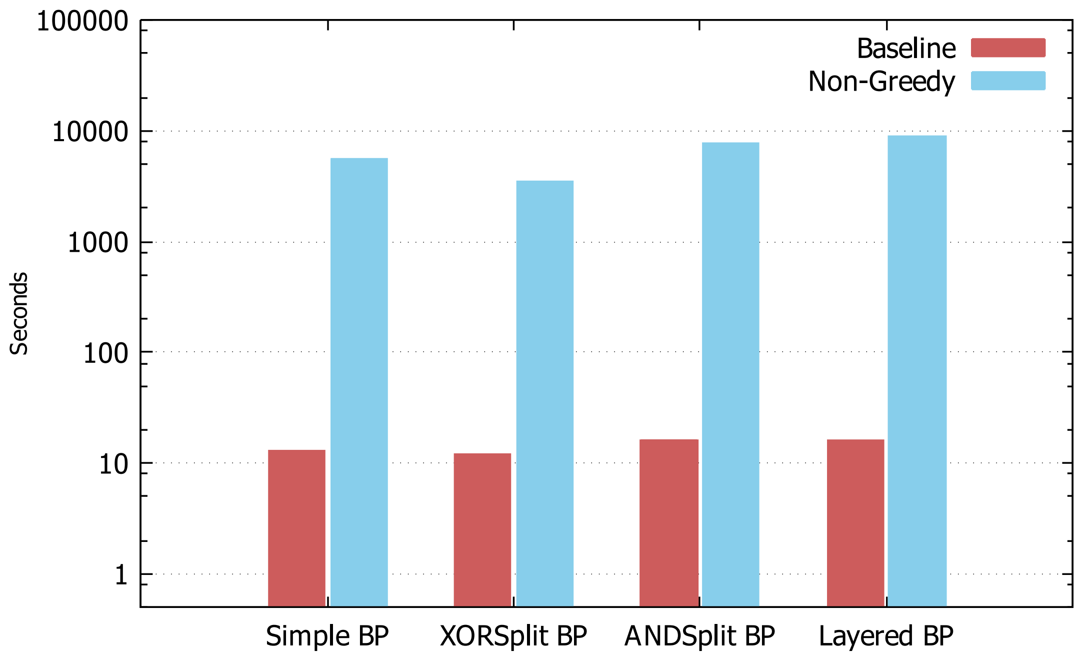


Figure 6.16: Bar Diagram comparing durations of the baseline and the non-greedy test runs

The average increases in duration per business process are listed in Table 6.7.

Table 6.7: Average duration increase in non-greedy runtime verification test runs

Business Process	Average Duration [%] Increase	Average Duration [s] Increase
Simple BP	42,297	5,488
XORSplit BP	29,014	3,477
ANDSplit BP	47,918	7,819
Layered BP	57,136	9,019

The only difference between a baseline enactment and a non-greedy runtime verification enactment is the usage of the runtime verification framework, described in Section 5.3. To properly document and control a workflow instance, the runtime verificatoin framework creates different workflow documentation elements. For instance, the start, the end or a handover of a workflow have to be recorded in the framework and the Blockchain. These documentation tasks, managed by the framework, must be responsible for the significant increase in execution duration.

Each time the runtime verification framework records a workflow step, similar sub-routines are executed. Each of these sub-routines has an impact on the time required by the recording effort of the framework. Each recording task of the runtime verification is made up of the following sub-routines.

REST API Requests Each runtime verification framework instance operates on top of a Bitcoin wallet. When this underlying Bitcoin wallet is operated in SPV mode, only transactions that directly concern the wallet are recorded and received. Therefore, a number of requests to the REST API may be required during a documentation task in order to collect information about workflow transactions that are not provided by the wallet.

Bitcoin Transaction Broadcasts & Confirmations When finally prepared, workflow transactions have to be broadcast to the Bitcoin P2P overlay. Furthermore, in the non-greedy mode of the simulation, each choreography participant waits for a confirmation depth of 1 for every broadcast transaction before continuing the enactment of the choreography.

Framework Recording Logic Each documentation task requires its own dedicated logic. Amongst other things, this includes preparing Bitcoin transactions, verifying the correctness of the instance's execution path and persisting the given documentation step to the permanent framework storage.

The average duration of a recording task performed in non-greedy mode is 879.677 seconds. In order to assess how the different sub-routines influence the runtime verification recording tasks of the framework, the duration of each recording task together with the duration of its sub-routines were analyzed. This analysis was conducted across all business process types, since the executed recording steps are always the same. The following diagram illustrates how the average execution time of a recording step is composed of the described sub-routines.

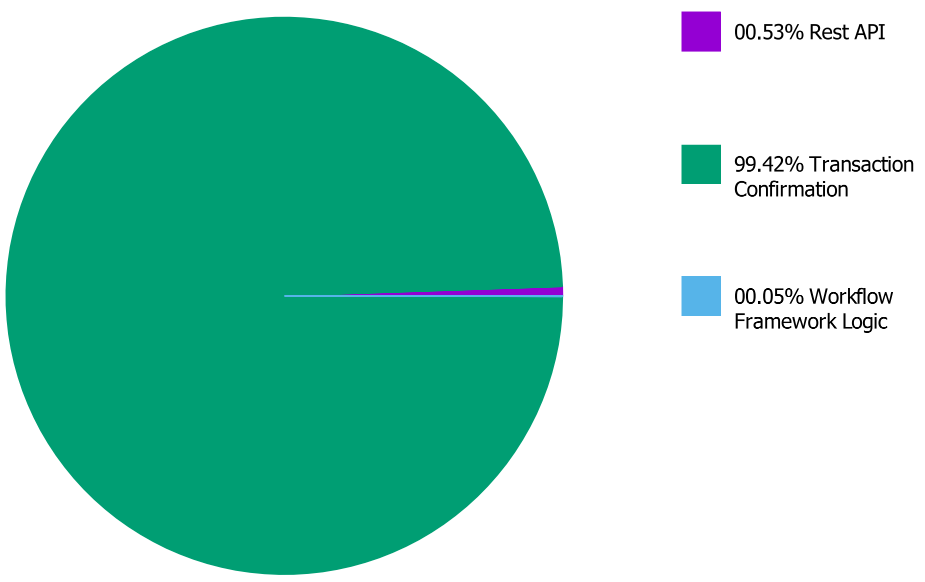


Figure 6.17: Pie diagram illustrating the composition of the average duration of a non-greedy runtime verification recording step

These results demonstrate that waiting for a transaction to confirm takes up the most time in the documentation steps of the runtime verification framework. Still, knowing the average recording duration for a workflow step does not enable estimations of the expected increase in overall execution duration. This becomes obvious when the average test run executions of the Tables 6.5 and 6.6 are compared to their respective standard deviations (σ). The execution durations of both modes have a very high standard deviation in comparison to their average. In some cases the standard deviation is even higher than the average. Therefore, even though the number of required recording steps of the test runs is known in advance, the exhibited execution duration varies significantly.

In Table 6.5, test run number 3 exhibited a much higher average execution duration than test run number 8 even though they perform the same number of recording steps. The same holds true for other test runs like test run number 15 and 25. Table 6.6 illustrates similar results. For example, test run number 2 has a higher average execution duration than test run number 16, even though test run number 16 performs more recording steps.

Therefore, the workflow transaction confirmation time has to be analyzed in greater detail. The distribution of all non-greedy transaction confirmation waiting durations is illustrated as a Box Plot in Figure 6.18.

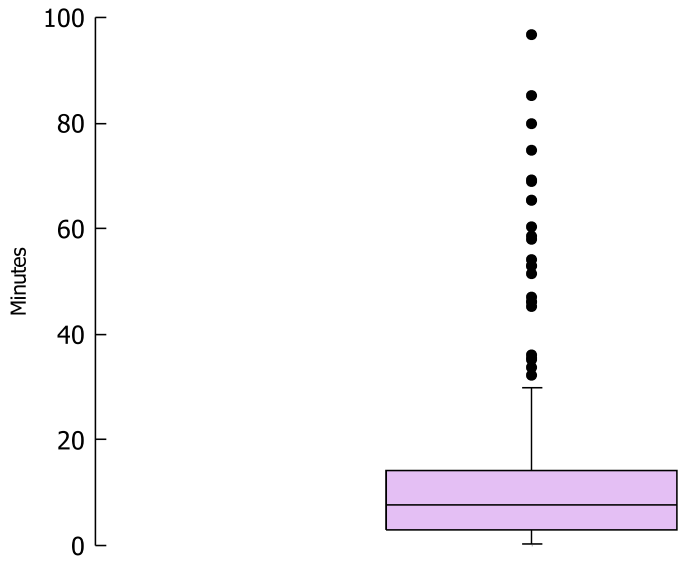


Figure 6.18: Box Plot illustrating the distribution of confirmation duration [min] of non-greedy transactions

A median transaction confirmation time of 7.741 minutes was recorded for all workflow transactions in the non-greedy mode. This is even slightly faster than the median confirmation time of 10 minutes the Bitcoin network is configured to exhibit. Still, a lot of outliners were recorded. One transaction took even 172.779 minutes to confirm.

This result is not surprising, given the fact that the block creation duration is approximately distributed exponentially [40]. But there also seem be other factors which influence the confirmation time of a transaction. During the simulation the following potential factors on the transaction confirmation were uncovered.

As described in Section 2.3, the Bitcoin network struggles with scalability. The overall transaction throughput is sometimes not high enough for the increasing demand. Transactions which are queued to be included into a block must be buffered by the miners. During a time of peak load, the number of waiting transactions can increase significantly, effectively jamming all other transactions that arrive at a later point in time. The measurements for this thesis were recorded from 10.29.2016 to 11.15.2016. During this time period the load in the Blockchain network remained relatively stable [2]. Shortly before and shortly after the recordings, the Bitcoin network exhibited rapid load spikes, which both times resulted in a immense increase of waiting transactions[3]. It is very likely

[2]https://blockchain.info/de/charts/mempool-count?start=2016-10-29&timespan=3weeks&sampled=true

[3]https://blockchain.info/de/charts/mempool-count?start=2016-10-15&timespan=2months&sampled=true

that a submitted transaction will take longer to confirm when there are already a lot of other waiting transactions buffered by the network.

Another potential influencing factor on the transaction confirmation time is the transaction fee. Each transaction must provide a fee in order to give miners an incentive to store it into a block. The higher this fee is, the higher the priority of the transaction becomes. Transaction fees are measured in BTC per kilobyte (kB). The number of inputs and outputs as well as the utilized scripts in these inputs and outputs determine the size of a transaction. Estimating the optimal fee for a transaction can be challenging. This value may be influenced by various factors, like operational cost for miners (e.g. power cost or hardware cost) or the current load in the Bitcoin P2P network. In version 0.12.0 of the official Bitcoin software the minimal transaction fee is configured as at least 0.00001 BTC/kB [4] but this is not a representative value.

The website `blocktrail.com` offers a metric for the current optimal fee per kilobyte for transactions [5]. Based on the observations collected during the last 30 blocks, transactions which provide at least this optimal fee have a change of a least 75% to be included into the next block. On the date of 11.24.2016, `blocktrail.com` suggested an amount of 0.00069654 BTC/kB as the current optimal fee. This metric can certainly serve as a guideline. On the other hand, the runtime verification framework proposed by this thesis creates workflow transactions with a fee of about 0.0005 BTC/kB. Even though this fee is lower than the optimal fee suggested by `blocktrail.com`, the measured median confirmation time of the workflow transactions was still lower than the networks specified median confirmation time of 10 minutes.

At last, the amount of Bitcoins sent by a transaction may also be a potential influence factor. As mentioned, the size of a transaction is mostly determined by its number of inputs and outputs. When only standard P2PKH payment scripts are utilized, the size of the used scripts always stays the same. For example a payment transaction that moves four BTC can have a similar size than a transaction that moves only 0.01 BTC. There are a number of reasons why transactions that move a higher amount of Bitcoins might be treated with a higher priority than transactions that move a lower amount.

Miners might prioritise high value transactions because this way they increase the overall monetary throughput of the network. But most importantly, small value transactions are frowned upon because they potentially can be used to spam the network. One person may purchase 1 BTC and publish $1,000,000$ transactions of 0.000001 BTC for free in order to spam the network. This is why the official Bitcoin software incorporates the *dust* rule [20]. The output of a transaction is referred to as *dust* when the fees for spending exceeds 1/3 of its value. This rule establishes a lower bound to the amount one transaction can possibly move but the problem of small value transactions as a thread for polluting the network remains.

[4]`https://github.com/bitcoin/bitcoin/blob/master/doc/release-notes/release-notes-0.12.0.md`

[5]`https://api.blocktrail.com/v1/BTC/fee-per-kb?api_key=18d4abb73b9375470ee74d7d0260d079bb813e63`

The simulation uncovered this potential influence factors on the confirmation time of a transaction. A thorough evaluation of these influence factors remains a part of future work.

As a summary, it can be concluded that the confirmation duration for workflow transactions is by far the greatest impact factor for the performance of the runtime verification framework in non-greedy mode. Also the requests of the REST API play a somewhat significant role. The REST component requires ten times more execution time than the framework's actual verification logic. But these requests can be avoided when the wallet, the runtime verification framework operates on, is not run in SPV mode.

Due to the high standard deviation of the transaction confirmation duration, a reliable prediction model for the increase in execution duration can not be created based on the test data collected in this thesis. Therefore, estimates of the expected duration increase imposed through the runtime verification framework can not be given.

Greedy Mode Results

When operating in greedy mode, the runtime verification framework still exhibits a significant increase in execution duration of the enacted workflows in comparison the verification-less baseline. Still, the impact of the verification framework on the execution duration is somewhat reduced in comparison the non-greedy test runs. The duration difference per business process of the baseline results and the greedy runtime verification results are illustrated in the following bar diagram. Note that the y-axis is plotted in $\log_{10}$.

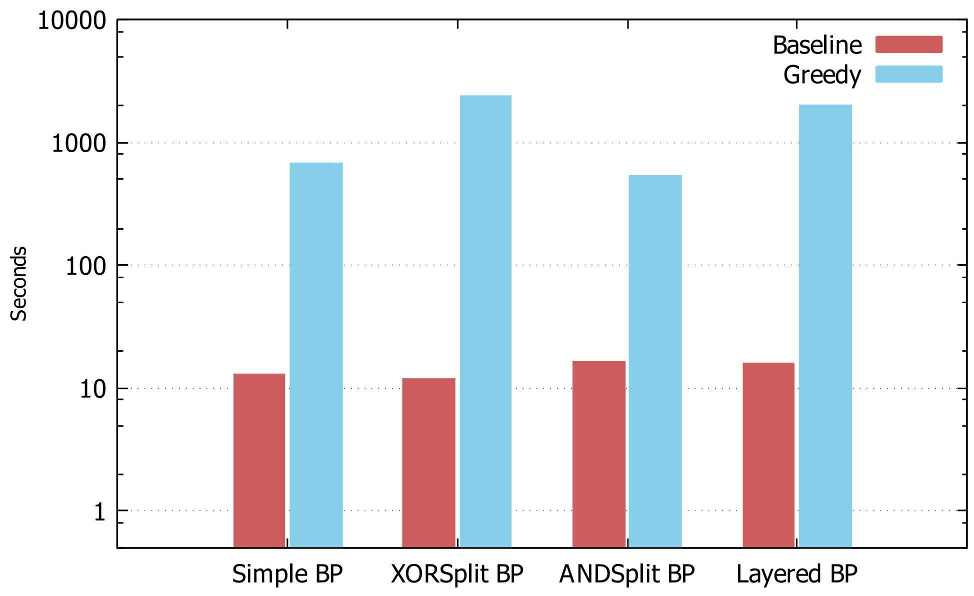


Figure 6.19: Bar diagram comparing durations of the baseline and the greedy test runs

The average increases in duration per business process are listed in Table 6.8.

Table 6.8: Average duration increase in greedy runtime verification test runs

Business Process	Average Duration [%] Increase	Average Duration [s] Increase
Simple BP	5,217	665
XORSplit BP	19,794	2,369
ANDSplit BP	3,314	525
Layered BP	12,600	1,977

Again the only difference between the baseline enactment and the greedy runtime verification enactment is the conducted recording of the single workflow steps. During each workflow step recording the sub-routines *REST API requests*, *Bitcoin transaction broadcast* and *framework recording logic*, described in the last section, are executed. Note that the sub-routine *Bitcoin transaction confirmation* is not executed for each recording. Instead, in the greedy mode there is only one collective waiting period for the transactions of a workflow instance.

First the composition of a single recording task is highlighted. The average duration of such a recording task is 12.296 seconds. The influence of the different sub-routines on the recording tasks of the runtime verification framework are illustrated in Figure 6.20.

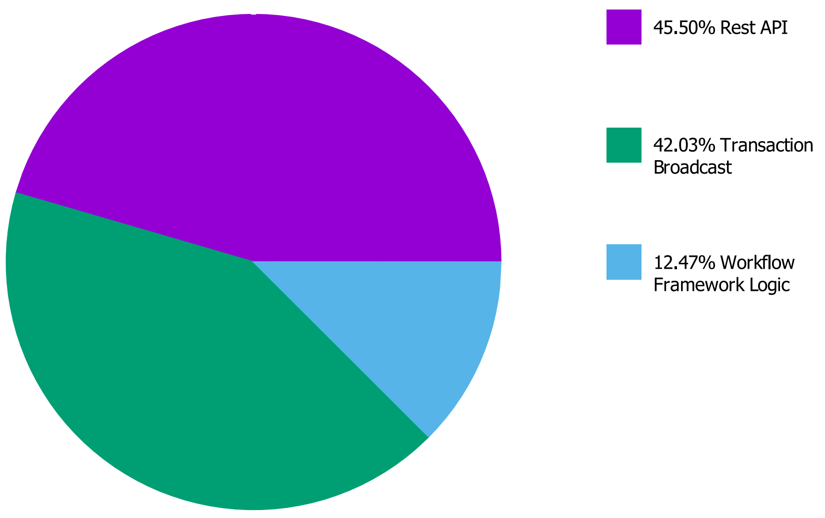


Figure 6.20: Pie diagram illustrating the composition of the average duration of a greedy runtime verification recording step

This is a much more diverse result. The REST API requests and the Bitcoin transaction broadcasts both impact the duration of a single workflow step recording about the same. Also the average recording step duration of 12.296 seconds is much lower than the

879.677 seconds of the non-greedy mode. The framework can potentially be even faster when operated on top of a full Bitcoin node instead of SPV wallet. This would remove the need for the REST API requests. Most importantly the workflow instances and their included tasks can be enacted faster.

But the collective waiting time for the transaction confirmations also have to be taken into account. A workflow instance is only considered finished when its submitted transactions have reached at least a confirmation depth of 1. The average execution duration of the greedy test runs is $1,520.677$ seconds. This duration is composed of the recording steps and the single transaction confirmation waiting period in the following way.

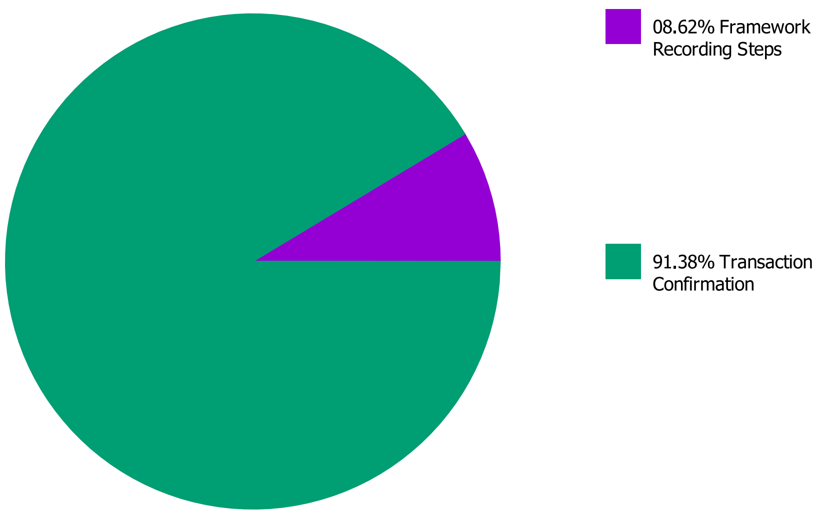


Figure 6.21: Pie diagram illustrating the composition of the average duration of a greedy test run

Even though the greedy mode was able to reduce the execution duration of the enacted workflow instances, the transaction confirmation duration remains the greatest impact factor. The distribution of the transaction confirmation waiting time of greedy published transactions is illustrated as a Box Plot in Figure 6.22.

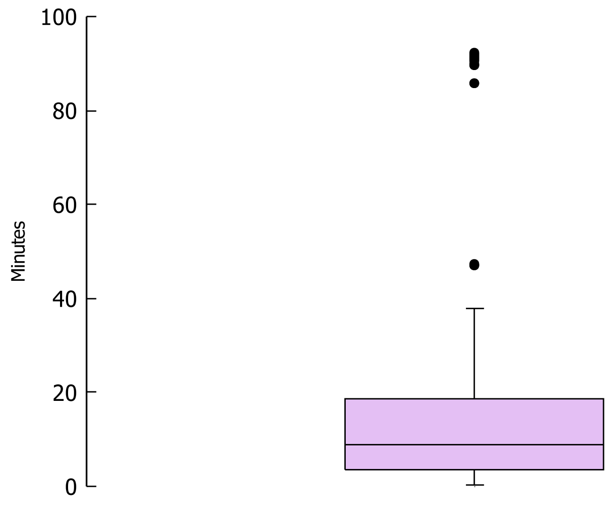


Figure 6.22: Box Plot illustrating the distribution of confirmation duration [min] of greedy transactions

The greedily published workflow transactions exhibit a median transaction confirmation time of 8.755 minutes. Similar to the results of the non-greedily published workflow transactions a lot of outliners occurred. The recorded maximum confirmation duration is 309.180 minutes.

No statement can be given if chaining unconfirmed transactions has any effect on the transaction confirmation time. To the best of our knowledge, this method is not discussed in any other Bitcoin related publication or software. The median transaction confirmation time of the greedy mode results is slightly higher than the median transaction confirmation time of the non-greedy mode results. It can not be determined if this deviation is related to the chaining of unconfirmed transactions. The deviation between the two median confirmation durations can also be explained through the exponential distribution of Bitcoin's block creation time.

Due to the high standard deviation of the recorded transaction confirmation durations, a reliable prediction metric could not be created for the duration of greedily enacted workflow instances when the number of recording steps is known.

Summarized Result

In both the non-greedy and greedy choreography enactments result, waiting for the published transactions to be confirmed has by far the highest impact on the performance of the proposed runtime verification framework. This is a very unpredictable factor since the transaction confirmation duration is distributed exponentially [40]. Besides the block creation duration, other potential influence factors on the transaction confirmation time were discussed. An detailed evaluation of these factors remains part of future work.

The greedy enactment mode of the prototype was able to reduce the overall test run duration by about a factor of 56 in comparison to the non-greedy enactment mode. This significant improvement was achieved even though the median transaction confirmation duration of the non-greedily published transaction was lower than the median of the greedily published transactions. While the framework has to wait for the confirmation of each non-greedy transaction independently, the chained greedy transactions only have one overlapping waiting time. This increase in performance is traded against a reduced level of security, as explained in Section 5.4.3.

Due to the limited resources of this thesis, the simulated choreography participants operated on SPV wallets instead of full node wallets. The REST API requests that had to be sent to counteract the shortcomings of the SPV wallets were explicitly separated from the actual verification logic of the framework. In comparison to the confirmation waiting time, the REST API did not impact the execution time significantly. By using full node wallets, they can be avoided entirely.

Due to the high standard deviation of the transaction confirmation duration, the collected results were not suited to construct a practical prediction metric. Therefore, the expected increase of the execution duration of workflow instances can currently not be estimated.

Enactment Correctness

The capability of the runtime verification framework to actually detect incorrect enactments was also tested. A number of test runs were configured to purposefully exhibit incorrect behavior at a random handover between choreography participants. To perform incorrect behavior, a company tries to handover a workflow instance to another company with the instructions to enact a task which does not fit at this certain stage of the enactment.

This is a valid attack angle, since the task which is supposed to be executed by the receiving company is documented by the handover transaction that is signed by both the sending and the receiving participant. When a receiving company detects such incorrect behavior during a handover it aborts the handover process. Therefore, this company also does not sign the workflow handover template proposed by the sending company.

The sending company is therefore not able to publish a correct workflow handover transaction. When the sending company is notified that the receiver recognized the incorrect handover, it ends the enactment of the workflow instance by publishing an extraordinary workflow-end transaction.

The fault detection success rate of the runtime verification prototype in the various test runs is listed in Table 6.9.

Table 6.9: Fault detection success rate of executed test runs

Test Number	Business Process	Test Runs	Successful Detections
5	Simple BP	6	100%
6	Simple BP	7	100%
9	XORSplit BP	7	100%
10	XORSplit BP	5	100%
17	ANDSplit BP	7	100%
18	ANDSplit BP	4	100%
31	Layered BP	2	100%
32	Layered BP	4	100%

The prototype was able to successfully detect all attempts of incorrect behavior.

6.2.3 Simulation Data

The simulation was conducted, using the random number generator seed 1714322490. Test data definitions used for the test runs are listed in table 6.10.

Table 6.10: Test run defintions

Test Number	Business process model	Execution variant	Included fault in process	Use runtime verification	Run in greedy mode
1	Simple BP	1	false	false	true
2	Simple BP	1	false	true	true
3	Simple BP	1	false	true	false
4	Simple BP	1	true	false	true
5	Simple BP	1	true	true	true
6	Simple BP	1	true	true	false
7	XORSplit BP	1	false	true	true
8	XORSplit BP	2	false	true	false
9	XORSplit BP	1	true	true	false
10	XORSplit BP	2	true	true	true
11	XORSplit BP	1	false	false	true
12	XORSplit BP	2	false	false	true
13	XORSplit BP	1	true	false	true
14	XORSplit BP	2	true	false	true
15	ANDSplit BP	1	false	true	false
16	ANDSplit BP	1	false	true	true
17	ANDSplit BP	1	true	true	false
18	ANDSplit BP	1	true	true	true
19	ANDSplit BP	1	false	false	true
20	ANDSplit BP	1	false	false	true
21	ANDSplit BP	1	true	false	true
22	ANDSplit BP	1	true	false	true
23	Layered BP	1	false	true	true
24	Layered BP	2	false	true	true
25	Layered BP	1	false	true	false
26	Layered BP	2	false	true	false
27	Layered BP	1	false	false	true
28	Layered BP	2	false	false	true
29	Layered BP	1	false	false	true
30	Layered BP	2	false	false	true
31	Layered BP	2	true	true	true
32	Layered BP	2	true	true	false

6.3 Enabled Use Cases

The runtime verification approach of this thesis enables on-demand selection of choreography participants. Therefore, any company that is currently in control of a workflow token can freely decide where to forward this token to.

In addition, the performance evaluation demonstrated that in the most secure non-greedy operation mode the proposed prototype significantly increased the execution duration of a workflow. The main influencing factor of the duration increase was the transaction confirmation time. During the conducted evaluation a median transaction confirmation time of 7.741 minutes were recorded.

Both these factors implicate that the runtime verification prototype is best suited for use cases with long running task. In B2B workflows with tasks that take themselves a very long time the duration increase generated by Blockchain-based runtime verification is of little significance. Exemplary real world use cases that fit this description are logistic and supply chain processes. In some cases also long running execution processes may fall into this category.

For processes with many short running activities the runtime verification prototype in the non-greedy mode is not suited. The performance evaluation was conducted with exactly such processes. All the business processes that were used during during the test runs include only simulated activities. The enactment of each activity is represented by a waiting period of 5 seconds. In this environment of software-only processes, the enactment of the verification-less test runs was very fast, as listed in Table 6.4. On the other hand as illustrated in Figure 6.16, increased the average execution duration through the usage of the non-greedy runtime verification prototype significantly. In some cases up to 57.136 %.

The proposed greedy mode of the runtime verification prototype was able to mitigate this issue to a certain extend. The overall test run durations of greedily enacted test runs are reduced by a factor of 56 in respect to the non-greedily enacted test runs. Real world business processes that include a series of very short activities can alternatively incorporate runtime verification that operates in the greedy mode. Examples for such business process would be software-only processes which perform a series of software-based calculation steps. While suffering from a reduced security level, the main benefit of the greedy mode is its non-blocking nature. A workflow instance can be enacted with almost no delay until its end is reached. Before these workflow instances are then ultimately considered finished they then have to wait for the collective confirmation of a greedily published workflow transaction.

CHAPTER 7

Conclusion

In choreography-oriented cooperation the control over workflow instances is shared by all participants [75, 77]. When a workflow instance is enacted as a choreography, the company owning the process hands over the control of the instance to other participants for a number of subsequent process steps. This approach increases scalability and robustness but requires the participants to trust each other to a certain extent, even though the involved companies might be potential competitors [58, 93].

For such scenarios the need for distributed verification arises. It is essential for process owners to be able to verify which task was performed by which cooperating partner at what specific time. Unfortunately, the research about choreography-oriented WfMSs is still in its infancy [2]. To the best of our knowledge there are no commonly agreed on standards for the design, implementation or operation of B2B workflow choreographies. Based on this situation, there are also only a small number of solutions to perform runtime verification in a distributed environment, as described in Section 5.2. Both problems mutually influence each other. The lack of a mature choreography-oriented WfMS prevents the inclusion of a runtime verification prototype. On the other hand, the lack of a suited solution for distributed runtime verification might hinder the adoption of choreography-oriented WfMS.

The recent trend about the cryptocurrency Bitcoin and its Blockchain inspires a new solution approach to this problem. Bitcoin was developed as a distributed digital currency. Actors protect the access to their funds through cryptographic safety measures. But in the payment process between two independent actors a documentation mechanism is required. As with every financial transaction, thorough documentation and verification is crucial [85]. The transfer of ownership of Bitcoins must be indisputable and non-reversible. At the same time, this documentation mechanism should be independent. Bitcoin solved this problem with a novel technology, a distributed ledger called the Blockchain [99]. The maintenance of a Blockchain is performed by a large number of independent peers, called *miners* [5].

To foster the development of mature choreography-oriented WfMSs, a novel approach to runtime verification that utilizes a Blockchain was developed in this thesis as scientific contribution. In its first research challenge this thesis highlighted ways to utilize the Blockchain technology for distributed runtime verification.

To address this, a number of existing runtime verification approaches from the areas of workflow management and B2B cooperation have been described in Section 5. The discovered existing approaches can be divided into two categories. Runtime approaches of the first category rely on a cryptographic token that is passed along the choreography participants [19, 46, 54, 60, 68]. This token is used to document the progress and to control the enactment of a workflow instance. Other existing approaches aim to control the message exchange between the participants [13, 46, 89]. Only one other found approach also utilized the Blockchain as trust basis [91].

Furthermore, the properties of the Blockchains of three main operating cryptocurrencies were discussed in Section 5. These cryptocurrencies were selected according to their market capitalization [30]. The selected Blockchains can be divided into first generation and second generation Blockchains. First generation Blockchains are purpose built for a specific use case, for instance the exchange of digital currency. These Blockchains only offer a limited set of features which can be used for other not originally intended use cases. Blockchains of the second generation are designed with a broad set of features to intentionally promote a variety of use cases.

The basic security concept of Blockchains only works if, a large, independent and geographically distributed set of miners supports and maintains it. Furthermore, the underlying cryptographic concept of a Blockchain must be sound. Only for Bitcoin's Blockchain, the oldest first generation Blockchain, both the miner base and the cryptographic concepts have been evaluated.

Based on the findings about existing runtime verification approaches and operating Blockchains a novel prototype which utilizes the Bitcoin Blockchain was developed. Though not intended or supported by its design, the Bitcoin Blockchain is used by the proposed prototype to serve as trust basis for the indisputable documentation of the enactment of a workflow instance to meet these demands [34, 97].

As a second research challenge, this thesis evaluated the functional and non-functional properties of the developed prototype. In a feature comparison, the prototype was compared to the already existing runtime verification proposals. Based on the findings about choreography-oriented WfMSs, general criteria and requirements of runtime verification in the context of distributed workflow enactment were defined. By applying these criteria, the runtime verification approaches were categorized. This categorization was used as basis for a comparative discussion.

Token-based runtime verification approaches suffer from limited fault discovery capability. The designated workflow token might get lost or intercepted. There is no guarantee that the process owner gets the token back. Without the token, no proof for the enactment of a workflow instance exists. Furthermore, in order to provide privacy for the individual participants they always have to be predefined.

Message-control based approaches exhibit great fault discovery capabilities while at the same time providing privacy features and advanced runtime verification features. On the other hand, they are very invasive. The flaw in these approaches lies in their basis of trust. The utilized messaging facilities essentially becomes the new basis of trust. These messaging facilities require resources on their own and must be operated, potentially by a third party.

[91] and the prototype presented in this thesis use a Blockchain as shared trust basis. This Blockchain is used to manage cryptography tokens that represent the status of workflow instances. Furthermore, the Blockchain is used to control the access to the token (i.e. the message flow). [91] and the approach proposed in this thesis use a different Blockchain. While the approach of this thesis relies on the Bitcoin Blockchain, [91] utilizes the Ethereum Blockchain. Both approaches are able to protect the identities of their participating companies. They diverge in their participant selection, data sharing and sequence enforcement.

In order to protect the privacy of the workflow data and to enforce the execution sequence, [91] requires the participants to be selected prior to the enactment. In comparison to that, the approach of this thesis enables on-demand participant selection but is not yet able to ensure the privacy of the workflow data or to enforce the execution sequence.

To further assess the quality of the proposed runtime verification prototype a performance evaluation was conducted. The goal was to evaluate the overhead imposed by the usage of the prototype. Similar to the scientific contribution of [70], the performance of verification-enhanced workflow instances are compared to typical baselines. A simulation scenario was constructed to enable the enactment of workflow instances in an exemplary choreography-oriented WfMS.

The baseline measurements consisted of verification-less workflow instances that contained a number of small tasks. A verification-enhanced workflow instance is considered finished, when all published workflow transactions reached a confirmation depth of 1. In an initial conservative approach for each broadcasted transaction the enactment was halted until a confirmation was received. In this scenario, the utilization of the runtime verification framework led to an immense increase in execution duration. The duration of workflow instances increased up to 57.136 %. Different sub-routines of the runtime verification framework were analyzed to identify the cause for this increase in execution duration. The leading factor that caused the increase was the waiting period required by the published transactions to be confirmed.

Unfortunately the transaction confirmation time is a very unpredictable factor. In the Bitcoin Blockchain, the block creation duration is distributed exponentially [40]. Besides that, the thesis discusses other potential factors that might influence the confirmation time of a transaction.

As an alternative to the conservative approach, a greedy variant of the runtime verification framework was evaluated. By chaining unconfirmed transactions the average test run duration could be reduced by a factor of about 56. This improvement is unfortunately achieved through a reduced level of security. Similar to the conservative approach, the greedy operation mode was impacted the most through the transaction confirmation waiting period.

The transaction confirmation duration exhibited a high standard deviation. The recorded results could not be used to construct a prediction metric. It is therefore not possible to estimate the increase in duration for a verification-enhanced workflow instance.

Besides the measured performance, the proposed runtime verification prototype was able to uncover all incorrect handovers performed by the simulated companies.

During the conservative test runs a median confirmation time of 7.741 minutes was recorded. The Bitcoin network is configured to a block creation time of 10 minutes. Both these measures indicate that the prototype can be best utilized for business processes with long running activities. Real world examples for such business processes are transportation and supply chain processes.

For business processes with many short activities the greedy variant of the runtime verification framework can be used. In this configuration the waiting period for the transaction confirmations is postponed to the end of the enactment. The tasks of the workflow can be done beforehand. This way also real world processes like software-only processes with short calculation steps can be supported.

The runtime verification framework proposed in this thesis can be further enhanced. As described in Section 5, a fault management mechanism can be included by using multi-signature redeem scripts. To avoid unnecessary workflow handovers, the workflow transactions *workflow-start*, *workflow-end*, *workflow-split* and *workflow-join* can be extended to support the transfer of token ownership. Mechanisms to support confidentiality for the data of a workflow instance must be integrated. At last, the prototype should be tested in conjunction with other choreography-oriented WfMSs.

Different fields for future work were uncovered during the course of this thesis. There is currently no quality metric for Blockchains in general. The cryptographic features and miner base of many operating Blockchains has not yet been evaluated. At last, the influencing factors on the transaction confirmation time of the Bitcoin network should be further analyzed.

List of Figures

List of Tables

Appendix

Workflow Start Transactions

Since the have been stored in the Blockchain, all workflow transactions can be viewed online through Blockchain explorers[1].

Table 1: Transaction hashes of workflow-start transactions (part I)

Test Number	Transaction Hash
2	16e549e5e6b64d071a2e5f876d88f7f4eeade9764732e07e367b4b4a25825ef2
2	6fc9e091641e9e052c405d489729e7da51dc3f0ffd542487a7707442e0663b6b
2	70cf626adca38794e5d72988c9a68244f955147826fc578115e3e16b9afb7b49
2	9213fee15b311870c54deaf0f1137449a0c42ff8c38cd587c6be1b482cb2c3c1
3	407bdfc1733c274e9c374446201fd07c8b28398364b46dc3652ef452bf2580ac
3	a388b02c52a3730e27e3546ccee24f68444318a83133d3dd7b379df36495ffdf
5	05c39679a9fb76d2cfc9ccd7b4ec393e53efe430dc77c5a9328725fdedebafab
5	49c1c24b57a31680e30fcf3f5df6df544551e72496d682827fd0d3eb659484f4
5	5ed132c84a0d98ce0644d829d9a87bbf9a6206c75285f23f4c93700f5947e915
5	aecdb191c72b670d45c4c182f76d13a6b093456811dc8375735968afd66b0436
5	bf4abe581b4b34df84107ee34252d54746f0091c18f42f2285840c34df9916c7
5	c02cbbdfbe87430005f274f9f539cae46f4f1fe4a1647ba60226771a3ccf0b46
6	05f8ab32846d6381d2a9b2f15bef340cdea6af73c20c41914ac86c421545f4ad
6	077dc0eddac30450d36a8c3417cca646fda6c8ee7ce415810c83809e25960277
6	38758c8003875947ba9b4aad8a767be330917879eda8d4c9f9bc420600f7e1d3
6	4cc25f6729a7bc7ad49d466fab2cca57068d7b6e2671338eb43943f9d799db96
6	60b9302f69adab8b664c7fc05b7ae28537a0a3763532219095799c577c2db202
6	8db8db0287989c18c798c03a7656275fe976d2ba82e1b8d98cfb4da422e2dace
6	eb6f8b8d1917af8963d5ef2df16960cc5e29ce33c3f988dda3eb43e34aab5692

[1]`https://www.blocktrail.com/`

Table 2: Transaction hashes of workflow-start transactions (part II)

Test Number	Transaction Hash
7	489f3a0f0b0a22bc23cccd64675e56b1a11eb5058e87e1e1b917bbde73ed3f9d
7	4e18271dd104112bc9bf6b8db7afaab8d5fe1454e7180bed855f86e885d0278f
7	818a7d055f9f272a1681e09b600d49f8074705e150809af5f9c9807db6617a33
7	f90a67caf165d7ca8ec3d5efa4b9d5413c71b5055c95bec4fbe03a57a2c618af
8	5f12a6ebbe1d648f389f894f8ff4a05b4288b8e58b1fb60aa30e57e4672b52ac
8	836fce7f8a5156a097824c229d06ce753c5e6535b27083064a1479272364c4cf
8	c1a7134b6afa646fc2e3200665404e219494083fbb55ce67a58da3831f892eb4
8	daf23f43906fb5e598e51dbb787818fb1f8d80a59a34a426acdf93a9d39ec21c
9	0dfee070e09999abe8c664023d75441d7e0530f4979ee58dac3212c9e7c41ffe
9	475214bf321d563a999d37b46a5d1ef65afcd3e4b7647a68f21d18ef74232559
9	70b3e54258d22a63ca516d1577142a8557a8b8e852144db29901291e42c8fe03
9	87febe802e6677d59f9ebf29c9382ddebce012a0497a70ee08b3a53f08549091
9	9d20fc388bc48e85ab85cf640bc39a8ca51947f63864fad9b40a7d82e24cc34b
9	d6ee6470f4bf1b273659c5f1ef39d533c67f2afb5ef58146f8111e676a628285
9	fb8ce6d04ff509d9e252cc253211a0d8b062bd9bdf7a3aa5966669632422898c
10	122a0d8d187245c5727c3d2d697ee08b54234e2cc2fbcf01db133e28ed686e84
10	2d6e0b05a2b4b56c91d740140887e8e023ce288a02d7440b86f86e6bc1311e70
10	625d3ec50a9037005a1e420e9362032adbcf6d63b429c703b2dda4a47fcf01e0
10	cbc4f792ddca70991c22f48a5e3e98e84af85c1f8d13389c14d012295f67d44a
10	f615e1918f0f3645fcfbab89eda05d42557e932a9843d2da1a7aaf3faaa9d2c8
15	19e78683e71c45ea4e1359e7899a43e80ea80f7d7d11b3cef5ac45a07f3475e5
15	3b210d5bb25ec2bf4ccbcb0e079b94df22423fe33f75556e8909051bd3ff30b7
15	4bef472827091739cf859edaed719d46164a27e25977e70670eb8d37f4c189f2
15	be96407de4aa6ecf3913a73909c36c1f22f06a3738017d22a8ec24677dbb4668
16	20a911821585540a718b2d265249bd704831d69de3a30790eeac8c64f888f4e0
16	450c55058d942ed5733adc73dd5bc71ab23e24383084f7955b64ce25747db53b
16	826fe60c7e6835dcfe4e25c0774be61643634e9d81d318e8b2c196d643ba3e50
16	da9a2082ede3245f17fb3a916419d43ca04612a7caf923aa8fa4d416893d0a01
17	242e5cca2fd1b46b65e81469991392b831af5b7a866f7642492ccf224565fbb2
17	48e88b71dfae1be7af495bc065d974d3fdde9eb34bca019ae9ace25b7a937851
17	544712615c9485d8549a25d363996048e2b3abded6474b527865a85361aa13c9
17	5c848bdc8c700c596456582424b9c7c6f418606bf424b1253cf72ef621da0689
17	63ac3a839031f247bf37e1bec8f50c9f1084d86bee5a6d2f075096aabd6914d7
17	c567fc696abc03c8c7176d86ba985e10cfc26592a4856233e846780f3ef0fd5b
17	cbf530e636394ea87369268c63c747acf4c817e44f28d01898bc6fdec697966b

Table 3: Transaction hashes of workflow-start transactions (part III)

Test Number	Transaction Hash
18	08992fe9e0a950f76ccbf1932542dfd071b15f5c4b6aea89b71b959129acd43d
18	3dbcf2d72a7d4480ed56716e106f7913c2005d8ac73396cebd3f876461d03888
18	46512a6f0fcda7b1743235e8713cac1b91465e2a8f460bfe4e19ab01865ee2d8
18	7179609c60868e464fe519989aac259539f8b12a62fe6bcccefc86e0a41803d3
23	014b39e97c955dc2b2f3718293e4c678b9d96fb1caa62bcaf63b1ca049e3c574
23	0e7d93666cd6f3882ffa50f72dacd5ccbb034ef87adeeb98ac1614aad459ee44
23	a0a8bad745f7cf6f3e2eaade7f807b5abe456eb1e2576c664c80dbc8bc6340c3
24	03e01049619e6a9b571dd8654c162ea75a058805fe021af1e722c68c850b677e
24	09cc3c43d74ef6d9d74259e317b08064d9dbc3d7dd59a8278dec00046d3cea2c
24	3fb86f6b40746843fd380e135a9142f2bcbb22b1f273f5b58ec3557b8b5e6bd6
24	5ef6f04dcbb99ec16645bc557568f87332443dcb35cc48a0e9a672fd8b562963
24	d134cf24591d50252e3da22370a0efdc3e2fa2eb960a914dbf68456b01adda6a
25	1158074ad81ce985871c8170f29557007a7105a6bc8e72ee6c9957c16f8de10e
25	5e53a1462fd34ff0e9af1fe4508e7bd650d35702d377e9fd0f49d1b90adca0e9
25	8810835c19077f01201d3fcabbaaa698b49fd4cbe34f2c016d3c5570910c47a1
25	e3d60ec37bf113c56eb3ed0f679e1a8007024367a87d60cf78e198a55574b9b2
26	07ac6aed5de520a4fefbd0f9b520bbf6e447f13ddc291dadec3745680abfdbb5
26	556cfd1765b3a899d3784b3602f32d29bf129cf3a7712f1ddeb7d653c287a78e
26	5bf6c1dae6af6c5407e0c49a8e4ad0e475ffa8bdc625d9dca482aa2ba551286b
26	7c83b01a9d700f74c9546c8acf34c88416592ffb343c0d6f7d3579c1cbf72c22
31	2154dfb40705dd9436556eab112ecfdbc8bd714e29a186b5d07655b8868a2de1
31	433dc4939670df5dc0cc0035beb4311ddbb063efd2fd61b748dcd1b600950e0e
32	13e0fc93777f5095f89b82c9b6f19921d44d52e9e969d5212ffa86c90df88246
32	35dc078c0d9e81f1dbf987e12d9988eba381ff782fe1fd4d189bc6ebfd9c91bb
32	cf33a06d2ee104f31bcb306e8f6b384e60745317ad870cc64a876856c686b019
32	ea90e0546031e041b5b5d75f928b50bd08519ee4eecb8d4eacc8475b6cefc3ac

Bibliography

[1] G. Aagesen and J. Krogstie. *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems*, chapter BPMN 2.0 for Modeling Business Processes, pages 219–250. Springer, 2015. ISBN 978-3-642-45100-3. doi: 10.1007/978-3-642-45100-3_10. URL `http://dx.doi.org/10.1007/978-3-642-45100-3_10`.

[2] T. Ahmed and A. Srivastava. Service choreography: Present and future. In *International Conference on Services Computing*, pages 863–864, 2014. doi: 10.1109/SCC.2014.126. URL `http://dx.doi.org/10.1109/SCC.2014.126`.

[3] E. Aitenbichler, S. Borgert, and M. Mühlhäuser. *2nd International Conference on Subject-Oriented Business Process Management*, chapter Distributed Execution of S-BPM Business Processes, pages 19–35. Springer, 2011. ISBN 978-3-642-23135-3. doi: 10.1007/978-3-642-23135-3_2. URL `http://dx.doi.org/10.1007/978-3-642-23135-3_2`.

[4] L. Allen. *The Global Economic System Since 1945*. Reaktion Books, 2005. ISBN 978-1-86189-242-3.

[5] I. Alqassem and D. Svetinovic. Towards reference architecture for cryptocurrencies: Bitcoin architectural analysis. In *2014 IEEE International Conference on Internet of Things, Green Computing and Communications, and Cyber, Physical and Social Computing*, pages 436–443, 2014. doi: 10.1109/iThings.2014.78. URL `http://dx.doi.org/10.1109/iThings.2014.78`.

[6] L. Anderson, R. Holz, A. Ponomarev, P. Rimba, and I. Weber. New kids on the block: an analysis of modern blockchains. *Computing Research Repository*, abs/1606.06530, 2016. URL `http://arxiv.org/abs/1606.06530`.

[7] E. Androulaki and G. O. Karame. Hiding transaction amounts and balances in bitcoin. In *7th International Conference on Trust and Trustworthy Computing*, pages 161–178, 2014. doi: 10.1007/978-3-319-08593-7_11. URL `http://dx.doi.org/10.1007/978-3-319-08593-7_11`.

[8] R. J. Annette, A. W. Banu, and S. P. Chandran. Rendering-as-a-service: Taxonomy and comparison. *Procedia Computer Science*, 50:276–281, 2015. ISSN 1877-0509. doi: 10.1016/j.procs.2015.04.048. URL `http://dx.doi.org/10.1016/j.procs.2015.04.048`.

[9] J. Anseeuw, G. van Seghbroeck, B. Volckaert, and F. De Turck. BPMN extensions for decentralized execution and monitoring of business processes. In *5th International Conference on Cloud Computing and Services Science*, pages 304–309, 2015. doi: 10.5220/0005492303040309. URL `http://dx.doi.org/10.5220/0005492303040309`.

[10] A. M. Antonopoulos. *Mastering Bitcoin - Unlocking Digital Cryptocurrencies*. O'Reilly Media, 2014. ISBN 978-1-4493-7404-4.

[11] E. Asnina and G. Alksnis. Survey on information monitoring and control in cross-enterprise collaborative business processes. In *7th International Workshop on Information Logistics and Knowledge Supply*, pages 1–12, 2014. URL `http://ceur-ws.org/Vol-1246/paper-01.pdf`.

[12] J. Bacon, D. M. Eyers, J. Singh, B. Shand, M. Migliavacca, and P. R. Pietzuch. Security in multi-domain event-based systems. *it - Information Technology*, 51:277–284, 2009. doi: 10.1524/itit.2009.0552. URL `http://dx.doi.org/10.1524/itit.2009.0552`.

[13] A. Baouab, O. Perrin, and C. Godart. An event-driven approach for runtime verification of inter-organizational choreographies. In *IEEE International Conference on Services Computing*, pages 640–647, 2011. ISBN 978-1-4577-0863-3. doi: 10.1109/SCC.2011.55. URL `http://dx.doi.org/10.1109/SCC.2011.55`.

[14] S. Barber, X. Boyen, E. Shi, and E. Uzun. Bitter to better - how to make bitcoin a better currency. In *16th International Conference on Financial Cryptography and Data Security*, pages 399–414, 2012. doi: 10.1007/978-3-642-32946-3_29. URL `http://dx.doi.org/10.1007/978-3-642-32946-3_29`.

[15] L. Baresi, A. Maurino, and S. Modafferi. Towards distributed BPEL orchestrations. *European Association of Software Science and Technology*, 3, 2006. doi: 10.14279/tuj.eceasst.3.7. URL `http://dx.doi.org/10.14279/tuj.eceasst.3.7`.

[16] A. Barker, P. Besana, D. Robertson, and J. B. Weissman. The benefits of service choreography for data-intensive computing. In *7th International Workshop on Challenges of Large Applications in Distributed Environments*, pages 1–10. ACM, 2009. doi: 10.1145/1552315.1552317. URL `http://dx.doi.org/10.1145/1552315.1552317`.

[17] C. Bartolini, A. Bertolino, G. De Angelis, A. Ciancone, and R. Mirandola. Apprehensive qos monitoring of service choreographies. In *28th Annual ACM Symposium on*

Applied Computing, pages 1893–1899, 2013. ISBN 978-1-4503-1656-9. doi: 10.1145/2480362.2480713. URL http://dx.doi.org/10.1145/2480362.2480713.

[18] A. Ben Hamida, F. Kon, N. Lago, A. Zarras, and D. Athanasopoulos. Integrated choreos middleware - enabling large-scale, qos-aware adaptive choreographies, 2013. URL https://hal.inria.fr/hal-00912882/document. [ONLINE], Accessed: 2016-04-07.

[19] A. Bengtsson and L. Westerdahl. Secure choreography of cooperating web services. In *IEEE International Conference on Web Services*, pages 152–159, 2005. ISBN 0-7695-2484-2. doi: 10.1109/ECOWS.2005.21. URL http://dx.doi.org/10.1109/ECOWS.2005.21.

[20] Bitcoin. Dust definition of the bitcoin client, 2016. URL https://github.com/bitcoin/bitcoin/blob/v0.10.0rc3/src/primitives/transaction.h#L137. [ONLINE], Accessed: 2016-11-25.

[21] F. Bitcoin. Scalability, 2016. URL https://en.bitcoin.it/wiki/Scalability. [ONLINE], Accessed: 2016-01-11.

[22] F. Bitcoin. Finney attack, 2016. URL https://en.bitcoin.it/wiki/Double-spending#Finney_attack. [ONLINE], Accessed: 2016-09-16.

[23] F. Bitcoin. Simplified payment verification, 2016. URL https://en.bitcoin.it/wiki/Thin_Client_Security. [ONLINE], Accessed: 2016-09-19.

[24] F. Bitcoin. Satoshi (unit), 2016. URL https://en.bitcoin.it/wiki/Satoshi_%28unit%29. [ONLINE], Accessed: 2016-01-05.

[25] L. S. Blockchain. Average number of transactions per block, 2016. URL https://blockchain.info/charts/n-transactions-per-block?timespan=1year. [ONLINE], Accessed: 2016-09-19.

[26] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970. ISSN 0001-0782. doi: 10.1145/362686.362692. URL http://dx.doi.org/10.1145/362686.362692.

[27] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *18th International Conference on Financial Cryptography and Data Security*, pages 486–504, 2014. doi: 10.1007/978-3-662-45472-5_31. URL http://dx.doi.org/10.1007/978-3-662-45472-5_31.

[28] A. P. Buchmann and B. Koldehofe. Complex event processing. *it - Information Technology*, 51:241–242, 2009. doi: 10.1524/itit.2009.9058. URL http://dx.doi.org/10.1524/itit.2009.9058.

[29] A. Calabrò, F. Lonetti, and E. Marchetti. KPI evaluation of the business process execution through event monitoring activity. In *International Conference on Enterprise Systems*, pages 169–176, 2015. doi: 10.1109/ES.2015.23. URL `http://dx.doi.org/10.1109/ES.2015.23`.

[30] CoinMarketCap. Crypto-currency market capitalizations - august 28, 2016, 2016. URL `https://coinmarketcap.com/historical/20160828`. [ONLINE], Accessed: 2016-08-28.

[31] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *13th IEEE International Conference on Peer-to-Peer Computing*, pages 1–10, 2013. doi: 10.1109/P2P.2013.6688704. URL `http://dx.doi.org/10.1109/P2P.2013.6688704`.

[32] G. Decker, O. Kopp, F. Leymann, and M. Weske. Bpel4chor: Extending bpel for modeling choreographies. In *Proceedings of the IEEE 2007 International Conference on Web Services*, pages 296–303. IEEE Computer Society, 2007. doi: 10.1109/ICWS.2007.59. URL `http://doi.acm.org/10.1109/ICWS.2007.59`.

[33] R. Dermody, A. Krellenstein, and E. Wagner. Counterparty, 2016. URL `http://counterparty.io`. [ONLINE], Accessed: 2016-09-16.

[34] J. A. Donet Donet, C. Pérez-Solà, and J. Herrera-Joancomartí. The bitcoin P2P network. In *Financial Cryptography 2014: Workshops*, pages 87–102, 2014. doi: 10.1007/978-3-662-44774-1_7. URL `http://dx.doi.org/10.1007/978-3-662-44774-1_7`.

[35] C. Duhart, P. Sauvage, and C. Bertelle. EMMA: A resource oriented framework for service choreography over wireless sensor and actor networks. *Computing Research Repository*, 2015. URL `http://arxiv.org/abs/1506.02531`.

[36] J. Eder and A. Tahamtan. *19th International Conference on Database and Expert Systems Applications*, chapter Temporal Conformance of Federated Choreographies, pages 668–675. Springer, 2008. ISBN 978-3-540-85654-2. doi: 10.1007/978-3-540-85654-2_57. URL `http://dx.doi.org/10.1007/978-3-540-85654-2_57`.

[37] S. Ethereum. Ethereum, 2016. URL `https://www.ethereum.org/`. [ONLINE], Accessed: 2016-01-12.

[38] S. Ethereum. Ethereum white paper, 2016. URL `https://github.com/ethereum/wiki/wiki/White-Paper`. [ONLINE], Accessed: 2016-09-19.

[39] W. Fdhila, S. Rinderle-Ma, and M. Reichert. Change propagation in collaborative processes scenarios. In *8th International Conference on Collaborative Computing*, pages 452–461, 2012. doi: 10.4108/icst.collaboratecom.2012.250408. URL `http://dx.doi.org/10.4108/icst.collaboratecom.2012.250408`.

[40] P. Franco. *Understanding Bitcoin: Cryptography, Engineering and Economics*. Wiley, 2014. ISBN 978-1-119-01916-9.

[41] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology – EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310, 2015. doi: 10.1007/978-3-662-46803-6_10. URL http://dx.doi.org/10.1007/978-3-662-46803-6_10.

[42] B. Gipp, N. Meuschke, and A. Gernandt. Decentralized trusted timestamping using the crypto currency bitcoin. *Computing Research Repository*, abs/1502.04015, 2015. URL http://arxiv.org/abs/1502.04015.

[43] N. Herzberg, A. Meyer, and M. Weske. An event processing platform for business process management. In *17th IEEE International Conference on Enterprise Distributed Object Computing*, pages 107–116, 2013. doi: 10.1109/EDOC.2013.20. URL http://dx.doi.org/10.1109/EDOC.2013.20.

[44] D. Hobson. What is bitcoin? *XRDS: Crossroads*, 20:40–44, 2013. ISSN 1528-4972. doi: 10.1145/2510124. URL http://doi.acm.org/10.1145/2510124.

[45] C. N. Höfer and G. Karagiannis. Cloud computing services: taxonomy and comparison. *Journal of Internet Services and Applications*, 2:81–94, 2011. ISSN 1869-0238. doi: 10.1007/s13174-011-0027-x. URL http://dx.doi.org/10.1007/s13174-011-0027-x.

[46] G. Hwang, Y. Kao, and Y. Hsiao. Scalable and trustworthy cross-enterprise wfmss by cloud collaboration. In *IEEE International Congress on Big Data*, pages 70–77, 2013. ISBN 2379-7703. doi: 10.1109/BigData.Congress.2013.19. URL http://dx.doi.org/10.1109/BigData.Congress.2013.19.

[47] A. Intervista. Fmc, 2016. URL http://www.fmc-modeling.org/notation_reference. [ONLINE], Accessed: 2016-11-04.

[48] F. Jacobs. Providing better confidentiality and authentication on the internet using namecoin and minimalt. *Computing Research Repository*, abs/1407.6453, 2014. URL http://arxiv.org/abs/1407.6453.

[49] K. Jander and W. Lamersdorf. Jadex wfms: Distributed workflow management for private clouds. In *Conference on Networked Systems*, pages 84–91, 2013. ISBN 978-1-4673-5645-9. doi: 10.1109/NetSys.2013.20. URL http://dx.doi.org/10.1109/NetSys.2013.20.

[50] C. Janiesch, M. Matzner, and O. Müller. Beyond process monitoring: a proof-of-concept of event-driven business activity management. *Business Process Management Journal*, 18:625–643, 2012. doi: 10.1108/14637151211253765. URL http://dx.doi.org/10.1108/14637151211253765.

[51] R. Khalaf and F. Leymann. *8th International Conference on Business Process Management*, chapter Coordination for Fragmented Loops and Scopes in a Distributed Business Process, pages 178–194. Springer, 2010. ISBN 978-3-642-15618-2. doi: 10.1007/978-3-642-15618-2_14. URL http://dx.doi.org/10.1007/978-3-642-15618-2_14.

[52] S. King. Primecoin, 2016. URL http://primecoin.io/bin/primecoin-paper.pdf. [ONLINE], Accessed: 2016-01-12.

[53] A. Krohn-Grimberghe and C. Sorge. Practical aspects of the bitcoin system. *Computing Research Repository*, abs/1308.6760, 2013. URL http://arxiv.org/abs/1308.6760.

[54] N. Kuntze, A. U. Schmidt, Z. Velikova, and C. Rudolph. Trust in business processes. In *9th International Conference for Young Computer Scientists*, pages 1992–1997, 2008. ISBN 978-0-7695-3398-8. doi: 10.1109/ICYCS.2008.78. URL http://dx.doi.org/10.1109/ICYCS.2008.78.

[55] T. Lange. *Encyclopedia of Cryptography and Security*, chapter Hash-Based Signatures, pages 540–542. Springer, 2011. ISBN 978-1-4419-5906-5. doi: 10.1007/978-1-4419-5906-5_413. URL http://dx.doi.org/10.1007/978-1-4419-5906-5_413.

[56] C. Lee. Litecoin, 2016. URL https://litecoin.org/. [ONLINE], Accessed: 2016-01-12.

[57] I. Legrand, H. Newman, R. Voicu, C. Dobre, and C. Grigoras. Workflow management in large distributed systems. *Journal of Physics: Conference Series*, 331:072022, 2011. doi: 10.1088/1742-6596/331/7/072022. URL http://dx.doi.org/10.1088/1742-6596/331/7/072022.

[58] L. A. F. Leite, G. Ansaldi Oliva, G. M. Nogueira, M. A. Gerosa, F. Kon, and D. S. Milojicic. A systematic literature review of service choreography adaptation. *Service Oriented Computing and Applications*, 7(3):199–216, 2012. ISSN 1863-2394. doi: 10.1007/s11761-012-0125-z. URL http://dx.doi.org/10.1007/s11761-012-0125-z".

[59] F. Leymann. Bpel vs. bpmn 2.0: Should you care? In *Business Process Modeling Notation*, pages 8–13. Springer, 2010. ISBN 978-3-642-16298-5. doi: 10.1007/978-3-642-16298-5_2. URL http://dx.doi.org/10.1007/978-3-642-16298-5_2.

[60] H. W. Lim, F. Kerschbaum, and H. Wang. Workflow signatures for business process compliance. *IEEE Transactions on Dependable and Secure Computing*, 9:756–769, 2012. ISSN 1545-5971. doi: 10.1109/TDSC.2012.38. URL http://dx.doi.org/10.1109/TDSC.2012.38.

[61] X. Liu, D. Yuan, G. Zhang, J. Chen, and Y. Yang. *Handbook of Cloud Computing*, chapter SwinDeW-C: A Peer-to-Peer Based Cloud Workflow System, pages 309–332. Springer, 2010. ISBN 978-1-4419-6524-0. doi: 10.1007/978-1-4419-6524-0_13. URL http://dx.doi.org/10.1007/978-1-4419-6524-0_13.

[62] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0-201-72789-7.

[63] B. Ludäscher, M. Weske, T. McPhillips, and S. Bowers. Scientific workflows: Business as usual? In *7th International Conference on Business Process Management*, pages 31–47, 2009. doi: 10.1007/978-3-642-03848-8_4. URL http://dx.doi.org/10.1007/978-3-642-03848-8_4.

[64] D. Martin, D. Wutke, and F. Leymann. A novel approach to decentralized workflow enactment. In *12th International IEEE Conference on Enterprise Distributed Object Computing*, pages 127–136. IEEE Computer Society, 2008. doi: 10.1109/EDOC.2008.22. URL http://dx.doi.org/10.1109/EDOC.2008.22.

[65] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 Internet Measurement Conference*, pages 127–140, 2013. doi: 10.1145/2504730.2504747. URL http://doi.acm.org/10.1145/2504730.2504747.

[66] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *34th IEEE Symposium on Security and Privacy*, pages 397–411, 2013. doi: 10.1109/SP.2013.34. URL http://dx.doi.org/10.1109/SP.2013.34.

[67] A. Mohaisen and J. Kim. The sybil attacks and defenses: A survey. *Computing Research Repository*, abs/1312.6349, 2013. URL http://arxiv.org/abs/1312.6349.

[68] F. Montagut and R. Molva. Bridging security and fault management within distributed workflow management systems. *IEEE Transactions on Services Computing*, 1:33–48, 2008. ISSN 1939-1374. doi: 10.1109/TSC.2008.3. URL http://dx.doi.org/10.1109/TSC.2008.3.

[69] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL https://bitcoin.org/bitcoin.pdf. [ONLINE], Accessed: 2016-01-05.

[70] F. Nerieri, R. Prodan, T. Fahringer, and H. L. Truong. Overhead analysis of grid workflow applications. In *7th IEEE/ACM International Conference on Grid Computing*, pages 17–24, 2006. doi: 10.1109/ICGRID.2006.310993. URL http://dx.doi.org/10.1109/ICGRID.2006.310993.

[71] A. Norta, L. Ma, Y. Duan, A. Rull, M. Kõlvart, and K. Taveter. econtractual choreography-language properties towards cross-organizational business collaboration. *Journal of Internet Services and Applications*, 6:8:1–8:23, 2015. doi: 10.1186/s13174-015-0023-7. URL `http://dx.doi.org/10.1186/s13174-015-0023-7`.

[72] I. Object Management Group. Bpmn, 2016. URL `http://www.omg.org/spec/BPMN/2.0/PDF`. [ONLINE], Accessed: 2016-01-12.

[73] S. Omohundro. Cryptocurrencies, smart contracts, and artificial intelligence. *AI Matters*, 1:19–21, 2014. ISSN 2372-3483. doi: 10.1145/2685328.2685334. URL `http://doi.acm.org/10.1145/2685328.2685334`.

[74] M. Pantazoglou, I. Pogkas, and A. Tsalgatidou. Decentralized enactment of BPEL processes. *IEEE Transactions on Services Computing*, 7:184–197, 2014. ISSN 1939-1374. doi: 10.1109/TSC.2013.6. URL `http://dx.doi.org/10.1109/TSC.2013.6`.

[75] G. Pedraza and J. Estublier. *International Conference on Software and Systems Process*, chapter Distributed Orchestration Versus Choreography: The FOCAS Approach, pages 75–86. Springer, 2009. ISBN 978-3-642-01680-6. doi: 10.1007/978-3-642-01680-6_9. URL `http://dx.doi.org/10.1007/978-3-642-01680-6_9`.

[76] C. Peltz. Web services orchestration and choreography. *Communications of the ACM*, 36:46–52, 2003. ISSN 0018-9162. doi: 10.1109/MC.2003.1236471. URL `http://dx.doi.org/10.1109/MC.2003.1236471`.

[77] M. Poulin. Collaboration patterns in the soa ecosystem. In *Proceedings of the 3rd Workshop on Behavioural Modelling*, pages 12–16. ACM, 2011. ISBN 978-1-4503-0617-1. doi: 10.1145/1993956.1993958. URL `http://doi.acm.org/10.1145/1993956.1993958`.

[78] A. Röder, M. Lehmann, and K. Kabitzsch. Monitoring service choreographies. In *9th IEEE International Conference on Industrial Informatics*, pages 142–147, 2011. doi: 10.1109/INDIN.2011.6034852. URL `http://dx.doi.org/10.1109/INDIN.2011.6034852`.

[79] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer. *Applied SOA: SERVICE-ORIENTED ARCHITECTURE AND DESIGN STRATEGIES*. John Wiley & Sons, 2008. ISBN 978-0-470-22365-9.

[80] M. Schermann, K. Dongus, P. Yetton, and H. Krcmar. The role of transaction cost economics in information technology outsourcing research: A meta-analysis of the choice of contract type. *The Journal of Strategic Information Systems*, 2016. ISSN 0963-8687. doi: 10.1016/j.jsis.2016.02.004. URL `http://dx.doi.org/10.1016/j.jsis.2016.02.004`.

[81] S. Schulte, C. Janiesch, S. Venugopal, I. Weber, and P. Hoenisch. Elastic business process management: State of the art and open challenges for bpm in the cloud. *Future Generation Computer Systems*, 46:36–50, 2015. ISSN 0167-739X. doi: 10.1016/j.future.2014.09.005. URL `http://www.sciencedirect.com/science/article/pii/S0167739X1400168X`.

[82] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in bitcoin. In *19th International Conference on Financial Cryptography and Data Security*, pages 507–527, 2015. doi: 10.1007/978-3-662-47854-7_32. URL `http://dx.doi.org/10.1007/978-3-662-47854-7_32`.

[83] N. Stojnić and H. Schuldt. Osiris-sr: A scalable yet reliable distributed workflow execution engine. In *Workshop on Scalable Workflow Execution Engines and Technologies*, pages 3:1–3:12, 2013. ISBN 978-1-4503-2349-9. doi: 10.1145/2499896.2499899. URL `http://dx.doi.org/10.1145/2499896.2499899`.

[84] C.-H. Tsai, K.-C. Huang, F.-J. Wang, and C.-H. Chen. A distributed server architecture supporting dynamic resource provisioning for bpm-oriented workflow management systems. *Journal of Systems and Software*, 83(8):1538 – 1552, 2010. ISSN 0164-1212. doi: 10.1016/j.jss.2010.04.001. URL `http://dx.doi.org/10.1016/j.jss.2010.04.001`.

[85] F. Tschorsch and B. Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. Technical report, Cryptology ePrint Archive, 2015. URL `https://eprint.iacr.org/2015/464`.

[86] W. M. P. van der Aalst. Loosely coupled interorganizational workflows:: modeling and analyzing workflows crossing organizational boundaries. *Information & Management*, 37(2):67 – 75, 2000. ISSN 0378-7206. doi: 10.1016/S0378-7206(99)00038-5. URL `http://dx.doi.org/10.1016/S0378-7206(99)00038-5`.

[87] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske. Business process management: A survey. In *2nd International Conference on Business Process Management*, pages 1–12, 2003. doi: 10.1007/3-540-44895-0_1. URL `http://dx.doi.org/10.1007/3-540-44895-0_1`.

[88] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: Towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39:50–55, 2008. ISSN 0146-4833. doi: 10.1145/1496091.1496100. URL `http://doi.acm.org/10.1145/1496091.1496100`.

[89] M. Von Riegen and N. Ritter. Reliable monitoring for runtime validation of choreographies. In *4th International Conference on Internet and Web Applications and Services*, pages 310–315, 2009. ISBN 978-1-4244-3851-8. doi: 10.1109/ICIW.2009.52. URL `http://dx.doi.org/10.1109/ICIW.2009.52`.

[90] W3C. Web services choreography description language (wscdl), 2005. URL `https://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/`. [ONLINE], Accessed: 2016-03-04.

[91] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling. Untrusted business process monitoring and execution using blockchain. In *14th International Conference on Business Process Management*, pages 329–347, 2016. doi: 10.1007/978-3-319-45348-4_19. URL `http://dx.doi.org/10.1007/978-3-319-45348-4_19`.

[92] A. Weiß, S. G. Sáez, M. Hahn, and D. Karastoyanova. *Confederated International Conferences: On the Move to Meaningful Internet Systems*, chapter Approach and Refinement Strategies for Flexible Choreography Enactment, pages 93–111. Springer, 2014. ISBN 978-3-662-45563-0. doi: 978-3-662-45563-0. URL `http://dx.doi.org/10.1007/978-3-662-45563-0`.

[93] M. Weske. *Business Process Management - Concepts, Languages, Architectures, second edition*. Springer, 2012. ISBN 978-3-642-28615-5. doi: 10.1007/978-3-642-28616-2. URL `http://dx.doi.org/10.1007/978-3-642-28616-2`.

[94] B. Wetzstein, D. Karastoyanova, O. Kopp, F. Leymann, and D. Zwink. Cross-organizational process monitoring based on service choreographies. In *ACM Symposium on Applied Computing*, pages 2485–2490, 2010. ISBN 978-1-60558-639-7. doi: 10.1145/1774088.1774601. URL `http://dx.doi.org/10.1145/1774088.1774601`.

[95] A. S. White and D. Miers. *BPMN Modeling and Reference Guide*. Future Strategies Inc., 2008. ISBN 978-0977752720.

[96] D. Wutke. *Eine Infrastruktur für die dezentrale Ausführung von BPEL-Prozessen*. PhD thesis, Universität Stuttgart, 2010. URL `http://elib.uni-stuttgart.de/opus/volltexte/2010/5677`.

[97] A. Yeow. Bitnodes - global bitcoin nodes distribution, 2016. URL `https://bitnodes.21.co`. [ONLINE], Accessed: 2016-09-15.

[98] S. Zaplata, D. Bade, K. Hamann, and W. Lamersdorf. Ad-hoc management capabilities for distributed business processes. In *Business Process and Service Science - Proceedings of ISSS and BPSC*, pages 139–152, 2015. URL `http://subs.emis.de/LNI/Proceedings/Proceedings177/article6200.html`.

[99] A. Zohar. Bitcoin: Under the hood. *Communications of the ACM*, 58:104–113, 2015. ISSN 0001-0782. doi: 10.1145/2701411. URL `http://doi.acm.org/10.1145/2701411`.