INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Analysis and Discovery of Service Orchestrations

Carlos Miguel Delgado Libório

Dissertation for the degree of

Master of Science in Information Systems and Computer Engineering

Committee

October 2010

Analysis and Discovery of Service Orchestrations

Carlos M. D. Libório

“The key to completely read one organization, is to fully realize its organizational DNA, the sequence of activities that use, alter or process several resources, physical, informational or human, i.e. the organization’s business processes”

José Tribolet

To Conceição and Benjamim,

the better parts of the whole...

Acknowledgements

First of all I would like to thank Prof. Diogo R. Ferreira for providing me with this opportunity and for his excellent guidance, support, motivation, and knowledge sharing. I would also like to thank Gil Aires and Álvaro Rebuge, for their optimistic support and for the many conversations, work meetings and helpful insights along the course of this work. Their help and collaboration strongly improved the journey. Furthermore, my thanks go to every author for their work and contribution with inestimable background knowledge.

To my friends, a special thank you, for helping and providing their own knowledge and expertise, even when it was not related with this work. To Michel and Ricardo, for their continuous help and valuable feedback. To Ritocas, Tomé, Cláudia, Pedrocas, Helena, Renato and Moreira for their everlasting help, support and motivation. To Loquinhas, whose will of heart and dedication firmly encouraged me to move on.

Lastly, but definitely not least, to my parents and family, a special thank you very much for all these years of comfort and encouragement that made this work possible.

Carlos M. D. Libório

July 2010

Abstract

Service orchestrations are used by organizations to develop, implement and deploy business processes based on the linkage, composition and reutilization of services. These service orchestrations are modelled using a workflow technology prescribing service behaviour and interactions, and constitute themselves reusable services. Such services are used to implement organizational processes and can be invoked by other organizations, enabling business networkability, organization interoperability, and service globalization.

This work aims at analysing and discovering service orchestrations by means of applying process mining techniques. This way, it will be possible to analyse the run-time behaviour of service orchestrations, extract their run-time process models and compare them with the original models in order to improve or redesign business processes for an effective use of services in an enterprise environment.

Keywords: Service Orchestrations, Process Mining, Business Process Management, Service Oriented Architectures, Enterprise Application Integration, Choreographies, Interoperability

Resumo

As orquestrações de serviços são utilizadas pelas organizações para implementar, desenvolver e executar processos de negócio, através da composição e reutilização de serviços. Estas orquestrações de serviços são modeladas através de uma tecnologia de *workflow*, desenhando o seu comportamento e interacções, constituindo elas próprias serviços reutilizáveis. Os serviços são utilizados para implementar funcionalidades de uma organização que podem ser utilizadas por outras organizações, permitindo desta forma maior interoperabilidade e integração na rede global. Os processos de negócio passam a ser transversais não apenas a uma organização, mas incluindo vários nós de comunicação inter-organizacional.

Este trabalho pretende analisar e descobrir as orquestrações de serviços, através da aplicação de técnicas de *process mining*, ou extracção de processos. Desta forma, será possível analisar o comportamento em tempo de execução das orquestrações de serviços, extrair os modelos de processo executados e compará-los com os modelos originais, por forma a melhorar o seu desenho e redefinir os processos de negócio para uma melhor e mais eficaz utilização de serviços num ambiente empresarial.

Palavras chave: Orquestrações de Serviços, Extracção de Processos, Gestão de Processos de Negócio, Arquitecturas Orientadas aos Serviços, Integração de Aplicações Empresariais, Coreografias, Interoperabilidade

Contents

1 Introduction **1**

1.1 Goals . . . 1

1.2 Related Areas . . . 2

1.3 Outline . . . 2

2 Integration Platforms **3**

2.1 Enterprise Application Integration . . . 3

2.2 Enterprise Service Bus . . . 7

2.3 Service Orchestrations . . . 8

2.4 Choreographies . . . 10

2.5 Conclusion . . . 11

3 Process Mining **13**

3.1 Domain . . . 13

3.2 Mining Techniques . . . 18

3.3 Mining Web Services . . . 19

3.4 ProM: a Process Mining Framework . . . 20

3.5 Conclusion . . . 21

4 Mining Service Orchestrations **23**

4.1 Methodology . . . 23

4.2 Log Extraction . . . 24

4.3 Log Inspection . . . 26

4.4 Log Aggregation . . . 30

4.4.1 MXML Log File . . . 31

4.4.2 Process Model . . . 33

4.5 ProM Analysis . . . 33

4.6 Model Visualization . . . 35

4.7 Performance Analysis . . . 37

4.8 Conformance analysis . . . 39

4.8.1 Instance Conformance . . . 41

4.8.2 Model Conformance . . . 43

4.9 Service Network Architecture . . . 43

4.10 Conclusion . . . 44

5 OrchInsider: a Service Orchestrations' Mining Application 45

5.1 Versioning . . . 46
5.2 Communication Ports . . . 47
5.3 Design Model . . . 48
5.4 Performance Analysis . . . 49
5.5 Conformance Analysis . . . 51
5.6 Conclusion . . . 54

6 Case Study: *SportTicket Online* 55

6.1 *SportTicket Online's* Orchestrations . . . 55
6.1.1 Buying Process . . . 56
6.1.2 Returning Process . . . 56
6.2 Mining *SportTicket Online* . . . 56
6.2.1 Analysis and Discovery . . . 57
6.2.2 Design Model Visualization . . . 60
6.2.3 Performance Analysis . . . 65
6.2.4 Service Network Architecture . . . 67
6.3 Discussion . . . 68

7 Conclusion 69

7.1 Contributions . . . 69
7.2 Future Work . . . 70

Bibliography 71

List of Figures

2.1 Business Process Management extends typical Workflow Management Systems . . . 4
2.2 Service Oriented Architecture (SOA) (Ferreira, 2008) 6
2.3 Enterprise Service Bus architecture, connecting several applications and technologies 7
2.4 Service orchestration as a workflow logic with service composition 9
2.5 Service orchestration design in Microsoft Biztalk integration platform 9
2.6 Choreography as a structured interaction between service orchestrations in different organizations 10

3.1 The Process Mining domain, adapted from (Aalst et al., 2009) 15
3.2 UML diagram of the MXML format (Günther and van der Aalst, 2006) 16
3.3 ProM framework (Dongen et al., 2005) 21

4.1 Analysis and discovery of service orchestrations (ADSO), the mining methodology . 23
4.2 Microsoft Biztalk simplified database model with orchestration instance information 26
4.3 Service orchestration's instance information retrieval in Microsoft Biztalk integration platform 28
4.4 Service orchestration's activity information discovery in Microsoft Biztalk integration platform 29
4.5 Service orchestration's activity information in Microsoft Biztalk integration platform 30
4.6 Log aggregation's generated MXML log file 32
4.7 ProM's control-flow analysis of a service orchestration's MXML log file 34
4.8 Model dependency graph 35
4.9 Model visualization phase with orchestration-oriented information 37
4.10 Log replay during instance conformance checking 42

5.1 Deployed orchestrations and instance information, using OrchInsider 46
5.2 Version differences analysis, using OrchInsider 47
5.3 Service orchestration's external communications, using OrchInsider 48
5.4 Design model enhanced with performance analysis, using OrchInsider 49
5.5 Performance analysis with OrchInsider 50
5.6 Path analysis with OrchInsider 51
5.7 Analysis and discovery of service orchestrations (ADSO), the full cycle 52
5.8 Conformance analysis with OrchInsider 53
5.9 Service orchestration's activity conformance model, using OrchInsider 54

6.1 Process conformance analysis of *SportTicket Online's* buying process (a) 58
6.2 Process conformance analysis of *SportTicket Online's* buying process (b) 59
6.3 *SportTicket Online's* mined buying process enhanced with performance analysis (a) . 61
6.4 *SportTicket Online's* mined buying process enhanced with performance analysis (b) 62
6.5 *SportTicket Online's* mined buying process enhanced with performance analysis (c) . 63
6.6 *SportTicket Online's* buying process' performance information 65
6.7 Service network architecture of *SportTicket Online's* buying process 67
6.8 Service network architecture of *SportTicket Online's* returning process 67

List of Tables

3.1 Service orchestration's event log 14

4.1 Service orchestration's instance information in Microsoft Biztalk integration platform 28
4.2 Pre-processed event log from the log inspection phase 29
4.3 Log aggregation's dependency frequency table 33
4.4 Orchestration-oriented activity information 36
4.5 Service orchestration's activity performance information 38
4.6 Service orchestration's start and end activities 39
4.7 Activity mapping using event type information 41

6.1 Analysis questions concerning service orchestrations 55
6.2 Version information of *SportTicket Online's* buying process 57
6.3 Activity performance indicators *of SportTicket Online's* buying process 65
6.4 Non-predicted end activities retrieved from *SportTicket Online's* buying process . . . 66

Chapter 1

Introduction

Today, information systems are at the core of any organization, allowing, not only better adaptability but also interoperability. Globalization reinforces the need of organizations to perform better, faster, more efficiently, but also, the urge to communicate and develop partnerships (Laudon and Laudon, 2009; Legner and Wende, 2006). Business processes are no longer a one communication node only, they have the ability to "travel" through several organizations, becoming cross-organizational processes.

Service Oriented Architectures allow organizations to implement such business processes in a flexible way, modelling the behaviour of service orchestrations as workflows (Weijters and van der Aalst, 2002; Aalst et al., 2003a). These service orchestrations represent or model the implementation of one or more business processes along the organization's service and application portfolio. After deployment and during their execution time, service orchestrations generate vast amounts of run-time information that can be used to monitor and analyse an organization's business processes. Applying mining techniques to execution generated event logs, and developing a methodology to help at the discovery of service orchestrations, are the main goals of this work.

The problem addressed in this dissertation can be summarized by the following statement:

Problem statement: *Service orchestrations are used by organizations to develop and implement their business processes. Organizations need to seize control over their services and business processes. How should one take advantage of the vast amounts of run-time information generated by integration platforms, so it is possible to analyse the run-time behaviour of service orchestrations?*

1.1 Goals

This work introduces a process mining methodology to analyse and discover deployed service orchestrations using their run-time generated event logs, with the purpose of studying and unveiling run-time behaviours and performance issues. A mining methodology for the analysis and discovery of service orchestrations, referred to as the ADSO[1] mining methodology, will be proposed and introduced, along with a software application that implements such methodology, and a case study built, in which developed techniques are validated using one integration platform, but application is also possible in other integration scenarios.

[1]ADSO - "Analysis and Discovery of Service Orchestrations"

The proposed ADSO mining methodology aims to retrieve process-aware information from event logs, generated when service orchestrations are deployed and executed, this way enabling organizations to truly and effectively monitor, manage and control their business processes;

- Understanding executed behaviours and comparing discovered models with previously designed processes;
- Measuring the discovered model's conformance;
- Analysing process modelling and development along time;
- Measuring performance, and understanding possible improvements;
- Identifying and correctly evaluating existing problems and inefficiencies;
- Inspecting failures and understanding why and when then occurred;
- Realizing the orchestration's external communications and network architecture.

This work will show that process mining techniques and paradigms can be applied to service orchestrations, and valuable results achieved that can help organizations to seize control over their business processes.

The presented statement and thesis can be outlined as follows:

Thesis: *Process mining techniques can be utilized by organizations to analyse and discover the run-time behaviour of service orchestrations, unveiling process-aware and performance information.*

1.2 Related Areas

Adopting process mining techniques to integration platforms outlines the main purpose and course of this work. Integration platforms enable the construction of integration solutions and Service Oriented Architectures through the usage of service orchestrations. Process mining aims to discover underlying processes and process-aware information in event logs. Integration platforms and process mining are the two major representative areas of this work.

1.3 Outline

First, integration platforms will be presented and some of their major research areas, in Chapter 2. Service orchestrations are introduced and discussed along with service collaboration issues, service choreographies. Chapter 3 refers to process mining, its goals and the most relevant techniques concerning Service Oriented Architectures, and focusing on service orchestrations. ProM, a specific process mining framework is also presented. The proposed ADSO methodology and solution is addressed in Chapter 4. Chapter 5 presents a service orchestrations' mining application that applies the proposed mining methodology. A case study is performed, where deployed service orchestrations are used as application examples, so as to validate the proposed solution and test the mining application, in Chapter 6. Finally, conclusions are drawn in Chapter 7.

Chapter 2

Integration Platforms

Presently, organizations need to communicate and develop partnerships, stretching their business processes across distinct organizations. Process-Aware Information Systems (PAIS) have emerged in the last years, changing the way information systems were developed, shifting from data orientation approaches to process orientation approaches. The initial focus of information technology (IT) was on storing, retrieving and presenting information, data modelling. Modelling business processes was often neglected, which resulted on confusing process logic, weak optimization, low efficiency and adaptation. PAIS are "software systems that manage and execute operational processes involving people, applications, and/or information sources on the basis of process models" (Dumas et al., 2005). With the advent of the internet, business processes are required to adapt easily and more frequently, within tight deadlines, responding to changes in the organization's environment. Furthermore, fewer systems are designed and built from scratch. Existing applications are dynamically reused and composed to build other applications and services, through integration platforms.

Integration platforms enable business and service integration, and are introduced in this chapter, with some of their major representative and research areas that led to service orchestrations. Notions like Enterprise Application Integration (EAI) will be presented and discussed, along with Service Oriented Architectures (SOA) and their important role while developing business processes and implementing interoperability. Service oriented business processes, together with Business Process Management (BPM), have emerged in the last years as the new foundations to achieve integration of enterprise business processes or services, dictating and realizing a new paradigm, that of the service orchestration. The purpose of service orchestration will be introduced, and its relevance explained. Finally, the notion of service choreography, and how it can work together with service orchestrations to deliver business and service integration, is presented and discussed.

2.1 Enterprise Application Integration

Business Process Management (BPM) is defined, according to (Aalst et al., 2003a), as a "generic software system that is driven by explicit process designs to enact and manage operational business processes involving humans, organizations, applications, documents and other sources of information". In other words, it is the collection of tools, methods, techniques that enable designing, enactment and further analysis of business processes (Leymann et al., 2002). Typical Workflow Management Systems (WfMS) (Aalst and Hee, 2002; Dumas et al., 2005; Sayal et al., 2002) deliver business process design and deployment, but BPM aims to extend this with business process awareness. Figure 2.1 pictures the main differences between workflow systems and BPM, which are process observation and analysis. In the modelling phase, processes are designed[1] and implemented using a process-aware application (e.g., a WfMS). Afterwards, the modelled business process is deployed and executed. These two first phases may be seen as joint WfMS phases. The last two phases, observation and analysis, consist on gathering pertinent information about the business process execution, and applying this information to identify problems, measure performance and improve the process model.

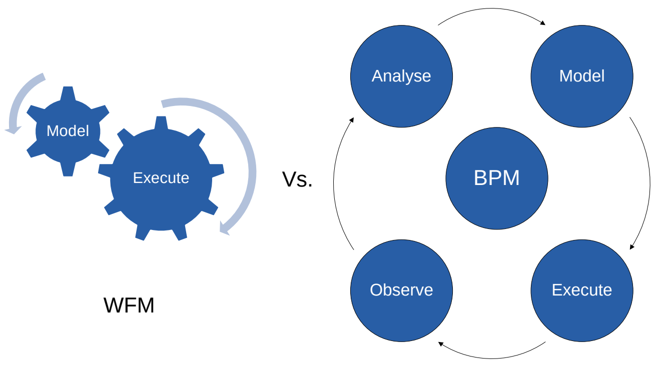


Figure 2.1: Business Process Management extends typical Workflow Management Systems

Cross-organizational business processes are becoming more important these days, along with Service Oriented Architectures (SOA), supporting business processes within an organization and between business partners[2]. Services are designed and deployed to perform tasks and processes, so in a sense, business processes are conducted and executed by services (Decker et al., 2006). Organizations are expanding their application portfolio every day, because they need to improve quality, customer's knowledge, increase supply chain efficiency, reduce market time. Keywords like globalization, customization, digitalization, virtualization, agility and networkability are gaining importance within organizations, and Enterprise Application Integration (EAI) has emerged as the enabling springboard for business integration.

According to (Huang and Fan, 2007), EAI's research level of integration can be classified as:

- Organization integration: Concerns about networked organizations and value models. Addresses the interactions between business strategy, organizational design, and information system design.

- Process integration: Concerns about integration at the business level, i.e. business process integration (BPI) and cross-organizational workflow management.

- Data integration: Relates to different data models (hierarchical, relational, etc), the syntactic and semantic differences between exchanged information.

[1]There is a close relationship between process design and modelling. The former refers to the overall design process and the latter refers to the representation of the business process model using a process language. In the course of this work, process design and modelling shall be referred as the process representation concerning activity and information flow, and also specific business rules.

[2]IBM - Web services architecture overview, http://www.ibm.com/developerworks/webservices/library/w-ovr/

- Application integration: Concerns about applications integration on heterogeneous platforms.
- Service integration: Deals with making services work together as a whole, solving their syntactic and semantic differences. SOA has been the key reference for some years and service orchestrations are amongst integration services.
- Semantic integration: Crosses all levels of EAI and deals with semantic equivalence between organizations so that related entities can communicate and understand each other. Ontologies and Semantic Web are two of the core technologies.

BPM and SOA have emerged in the last years as the "standards" for Enterprise Application Integration. These approaches separate design and business logic from implementation details, and are proposed by the Object Management Group (OMG)[3] as the foundations to achieve integration of enterprise business processes and services. BPM, by revealing the activities that constitute the process business flow, and the relations between them, delivers the necessary context for the definition of services. SOA provides a way of implementing and executing such services.

The World Wide Web Consortium (W3C)[4] refers to SOA as "a set of components which can be invoked, and whose interface descriptions can be published and discovered". Service Oriented Architecture consists on several software applications, representing well defined and independent services that may use other registered services, and are available in a network. Services are defined by a standard definition language and communicate with each other. For instance, web services, the most popular type of service today, uses Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP) and Universal Description, Discovery and Integration Registry (UDDI) as standards (Leymann et al., 2002; Papazoglou and Heuvel, 2007; Pusnik et al., 2003; Aalst, 2003; Di Lorenzo, 2008). Web services and their associated standards can be used to implement an organization's functionality that can henceforth be used by other applications. This way it is possible to integrate a miscellaneous of services, and enact business processes. Thus, using SOA, an organization can create, deploy and integrate multiple services and even choreograph business functions combining new and existing application assets into a logical business process flow. Figure 2.2 depicts this rationale, presenting the business process layer as an integration workflow of multiple services, the service interface layer, and each service enabling an application resource from the application layer.

To successfully enable SOA, some issues need to be addressed (Papazoglou and Heuvel, 2007):

- Enable each application as a service.
- Distributed services need to be orchestrated in order to build a distributed process.
- Deploy services and address security, reliability and scalability.
- Manage processes in a transparent way, without visibility to the underlying services.

Service orientation, as opposed to distributed architectures, reflects real-world processes and relationships more closely. This common belief addresses that SOA can build a much closer to reality model that solves real-world business process needs. SOA is a design technology and therefore technology independent. The most important aspect of SOA services is their reusability. (Papazoglou and Heuvel, 2007) and (Ross-Talbot, 2005) refer to SOA services as having three main properties:

[3]Object Management Group (OMG), http://www.omg.org/
[4]W3C, World Wide Web Consortium, http://www.w3.org/TR/ws-gloss/

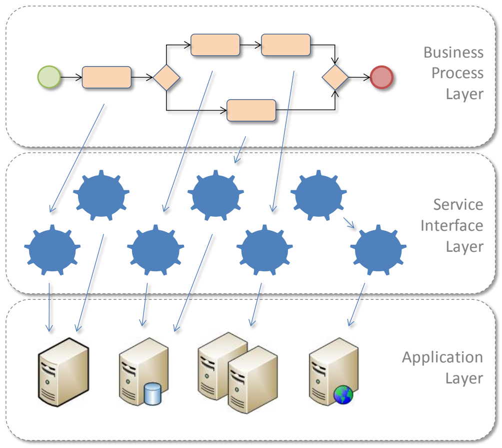


Figure 2.2: Service Oriented Architecture (SOA) (Ferreira, 2008)

1. Autonomous and self-contained, i.e., services keep their own state. They remain stateless across invocations.
2. Platform independent. SOA's loose-coupling principle states that there is a clear separation between the service interface and its implementation. Interfaces must be externally visible so that available linkages are understood.
3. Dynamically located, invoked and re-combined. Services can be located based on some discriminating factor, like security and behaviour, using, for instance, service interfaces in WSDL and UDDI lookups.

It is up to the workflow system to locate and bind services from one service to the other, structure interactions between them, so as to achieve a business process. This is often referred to as a service orchestration, a service broker that manages interactions between services, and sometimes human agents, creating an activity and information workflow.

In (Busi et al., 2005; Ross-Talbot, 2005; Papazoglou and Heuvel, 2007; Pedraza and Estublier, 2009; Aalst and Verbeek, 2008), the notions of orchestration and choreography are taken into account when designing service-oriented integration. Orchestrations are seen like single peer services that can be invoked by other services at different times, and choreographies describe the overall system. The notion of conformance between orchestrations and choreographies is also presented, in the sense that the overall behaviour, the choreography, and every peer's behaviour, the orchestrated system, are aligned. According to (Pedraza and Estublier, 2009), a single orchestration system has scalability problems, since a single computer machine is at the heart of the system, thus the hub of all communications, and potentially becoming a bottleneck, whereas choreographies represent service communication, since there is no central service. Orchestrations and choreographies are different, but also complementary ways to define workflows and implement EAI, creating cross-organizational business processes. These concepts will be addressed in the following sections.

2.2 Enterprise Service Bus

When connecting several applications and different technologies one might come across with two mismatch types, technology or information model mismatches. To solve them, there are two possible approaches;

- Change every client module to comply to every server module invoked, which is potentially troublesome an unmanageable or;
- Insert a layer of communication and integration logic between modules, viz. Enterprise Service Bus (ESB).

ESB is a highly distributable communication and integration backbone, a platform that provides interoperability between SOA based solutions, establishing proper control of messages as well as applying security, policy, reliability and accounting rules. ESBs, as depicted in Figure 2.3, aggregate services, with a more efficient integration, to form composite business processes, which in turn automate business functions in an organization.

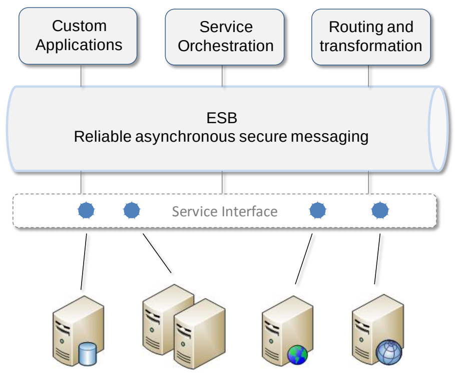

Custom Applications
Service Orchestration
Routing and transformation
ESB
Reliable asynchronous secure messaging
Service Interface

Figure 2.3: Enterprise Service Bus architecture, connecting several applications and technologies

Delivering Enterprise Application Integration (EAI) means that ESB's enable broker functionality, by providing integration services such as connectivity, message translation and routing based on business rules, data transformation, and application adaptors. These capabilities are themselves SOA-based in the sense that they are spread across the bus in a highly distributed fashion, hosted in separately deployable service containers. This means that ESBs have evolved from the primordial store-and-forward mechanism found in middleware products, e.g., message oriented middleware, to an EAI technology embracing web services, Extensible Stylesheet Language (XSLT) data transformation standards, orchestration, and choreography technologies. Also, becoming less centralized than traditional integration brokers, allows for event-driven services to be plugged into the ESB easily, whenever needed, and be scaled independently from each other. This is portrayed in Figure 2.3, where several applications running on different platforms are abstractly decoupled from each other, but connected together through the ESB as logical endpoints and exposed as event-driven services (Papazoglou and Heuvel, 2007; Sterff, 2006).

The advent of event-driven services has shortened the gap between the business context and the business process design. In an organization, any business event such as a customer or production order, may affect the course of a business process. This implies that business processes cannot be designed assuming that all events are predetermined and follow a particular flow, but must be driven by incoming event flows. Consequently, EAI must be built using an event-driven SOA, where services are seen as abstract service endpoints and can respond to asynchronous events. There is no longer any concern about protocol implementations or routing of messages, the ESB just simply publishes the messages to the services that have subscribed to the events. In some cases, when implementing SOA, service interfaces in WSDL and UDDI lookups are unavailable, disabling the service discovery phase and providing a more lightweight and straightforward integration platform (Papazoglou and Heuvel, 2007; Sterff, 2006). Event-driven SOA provides fully decoupled service exchanges, not just loosely coupled, in the sense that any participant services do not need to have any knowledge about each other, so there will no longer exist the need for a service contract (WSDL). Every service subscriber must only have knowledge about the event meta-data, normally XML-based, since the ESB will manage service relationships, in which every service can participate as a subscriber, or publisher of events. So, the ESB needs to effectively orchestrate the service behaviour enabling a distributed business process, using a distributed processing framework, and XML-based services to overcome heterogeneous services.

ESBs play a crucial role in BPM solutions today, enabling sophisticated process orchestration, thus creating a SOA capable of solving complex integration problems. Nowadays, there are several ESB products developed by BPM vendors, such as IBM's Websphere (Aalst and Verbeek, 2008), Sun's Glassfish[5], HP's HP Process Manager[6], BEA's WebLogic[7], Vitria's BusinessWare[8], Microsoft's Biztalk Server[9] and Software AG's WebMethods[10], which allow organizations to model, deploy, analyse, and refine process-driven integration solutions.

2.3 Service Orchestrations

According to (Ross-Talbot, 2005; Sterff, 2006; Peltz, 2003b,a; Di Lorenzo, 2008), a service orchestration is a recursive composition of services, in the sense that a service is built upon existing services. Activity and message flow are controlled by an orchestration service, also encapsulating workflow logic, the service logic. This can be depicted in Figure 2.4, where message exchanges and business logic is present. WS-BPEL[11] is the standard for orchestration language definition (Aalst, 2003; Pusnik et al., 2003; Moscato et al., 2005; Wassermann et al., 2006; Courbis and Heath, 2005).

Service orchestrations are used to compose and create services that belong to a specific process participant, describing how services can interact with each other at the message level, and the execution order of such interactions, thus creating business processes from the composition of services. Using service orchestrations, it is possible to develop, deploy and execute business processes that interact with external services. Enabling service and process collaboration, and therefore providing potential integration endpoints for other processes or services, service orchestrations become integration enablers.

Figure 2.5 illustrates a service orchestration, designed using Microsoft Biztalk integration server.

[5]https://glassfish.dev.java.net/

[6]HP Press release, http://www.hp.com/hpinfo/newsroom/press/2001/010723b.html

[7]Oracle has acquired BEA Systems, http://www.oracle.com/appserver/weblogic/weblogic-suite.html

[8]http://www.vitria.com/products/businessware/

[9]http://www.microsoft.com/biztalk/en/us/Default.aspx

[10]http://www.softwareag.com/corporate/products/wm/default.asp

[11]Web Services Business Process Execution Language, specs. available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

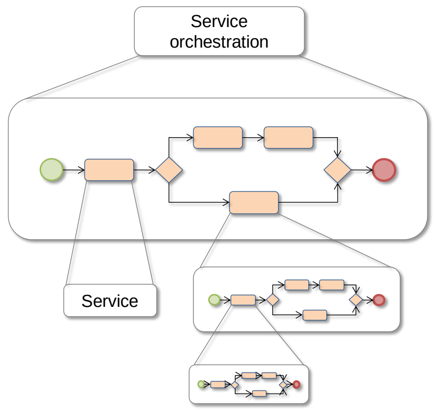
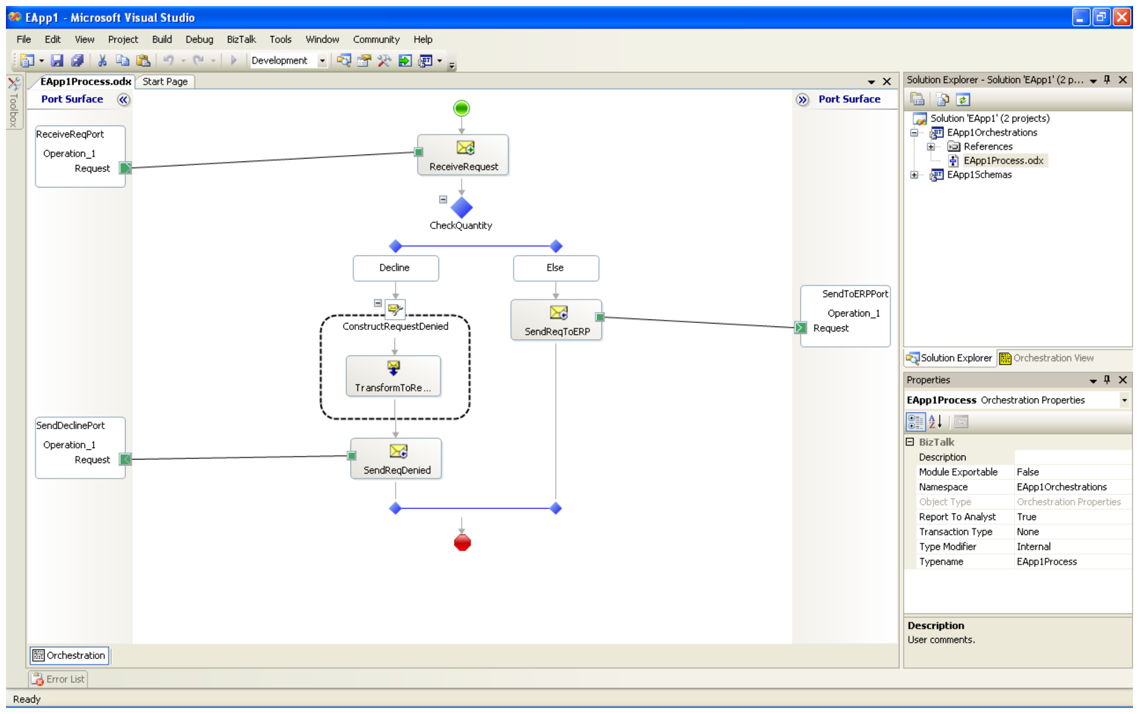


Figure 2.4: Service orchestration as a workflow logic with service composition

As one can observe, there is an activity flow with message exchanges and some service logic. The process model depicted in Figure 2.5 is easily understood, starting by receiving a request message, in the “ReceiveRequest” activity, through some communication port. This message contains an order request with the quantity for some product. The “CheckQuantity” activity is no more than a business rule that simply investigates this quantity and follows some rule. If the request is accepted, the request is sent to the ERP system through the “SendReqToERP” activity. Otherwise, a request denied message is constructed and sent back using some other communication port.



EApp1Process Orchestration Properties

BizTalk

Description	
Module Exportable	False
Namespace	EApp1Orchestrations
Object Type	Orchestration Properties
Report To Analyst	True
Transaction Type	None
Type Modifier	Internal
Typename	EApp1Process

Description
User comments.

Figure 2.5: Service orchestration design in Microsoft Biztalk integration platform

As mentioned before in Section 2.1, service orchestrations are centralized services, thus potentially becoming a bottleneck, but they can be deployed as web services or communicate with other orchestrations, which can implement a more distributed service orchestration, enabling scalability and improving performance; “Through the use of orchestrations, service-oriented solution environments become inherently extensible and adaptive, and, for many environments, orchestrations become the heart of SOA” (Erl, 2005).

2.4 Choreographies

Organizations use SOA to create logical business process flows that enable collaboration and interoperability. However, collaboration between different organizations requires rules and understandability. Service choreography addresses this issue, defining message exchanges from several services, controlling the collaboration logic.

Service choreography is a description of the externally observable interactions between existing services or orchestrations. These interactions are viewed from a global perspective and not from a service perspective (Peltz, 2003b; Fortis and Fortis, 2009; Peltz, 2003a; Di Lorenzo, 2008). Whereas an orchestration dictates a process execution flow, that of the broker service, describing business workflows within one organization, choreographies do not. Rather they describe the common observable interactions, the collaboration message flows between different organizations and services, without mandating any execution flow. The Web Services Choreography Description Language WS-CDL[12] is the standard for composed interoperability and peer-to-peer collaboration (Pedraza and Estublier, 2009; Decker et al., 2006; Sterff, 2006; Fortis and Fortis, 2009; Peltz, 2003b).

Figure 2.6 shows an illustration of an organization’s service architecture, where several services work together based on a SOA. Some of the represented services may belong to the organization

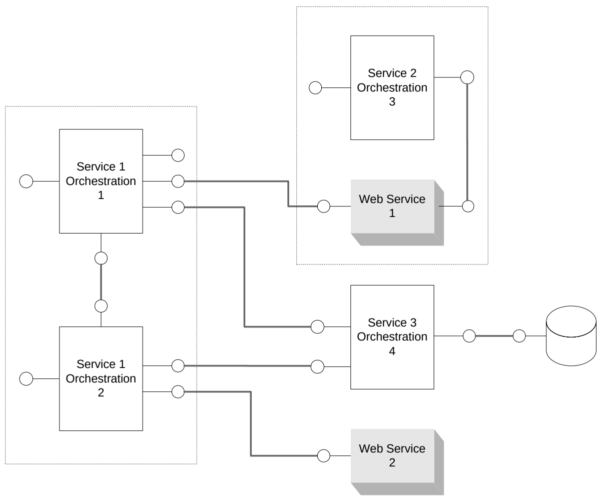


Figure 2.6: Choreography as a structured interaction between service orchestrations in different organizations

while others do not, so the collaboration logic is not controlled by one single organization. Instead, there is a community interchange pattern (Sterff, 2006) that enables services from different organizations to expose their functionalities and collaborate, establishing a choreography that can help EAI become more agile, since it is not very hard to compose, extend and reuse services.

[12]Web Services Choreography Description Language, specs. available at http://www.w3.org/TR/ws-cdl-10/

In choreography, there is no central hub, all services are distributed along several nodes, this way enabling scalability. But there is also the discoverability issue, since participant services need to find and use other provided services, making choreographies difficult to specify and implement. In (Pedraza and Estublier, 2009), a distributed orchestration approach[13] to choreographies is introduced to deal with these issues. A flexible decentralized orchestration is proposed, in which a traditional orchestration model is executed in a number of nodes, so called as choreography servers. This approach tries to fill the gap between "pure" orchestration and "pure" choreography, since choreography servers are implemented using traditional orchestration engines, enabling a distributed orchestration similar to the one depicted in Figure 2.6.

Choreography is the blueprint of the overall system, the service collaboration, not through a single workflow model, but by the set of messages that are exchanged between services, whilst orchestrations are a means of realizing the system, "business-specific applications of a choreography" (Erl, 2005). Together they can be used to implement EAI and build business processes from composite services, complementing each other.

2.5 Conclusion

This chapter has introduced integration platforms and some of their major challenges, research areas, and paradigms concerning service integration. Service orchestrations were presented as recursive composition of services, and therefore integration enablers, providing service and process collaboration. Although service orchestrations are centralized services, they can be deployed as web services or communicate with other service orchestrations, enabling service-oriented solution environments. Service collaboration issues, along with service choreographies, were also discussed, as descriptions of the externally observable interactions between services or service orchestrations. Choreographies describe the collaboration message flows between different endpoints, or exposed services, providing the blueprint of the overall system. In the following chapter, process mining will be presented, and shown how it can help to deliver "hidden" knowledge about the run-time behaviour of deployed service orchestrations.

[13]FOCAS, Framework for Orchestration, Composition and Aggregation of Services.

Chapter 3

Process Mining

The relevance of business process awareness has become increasingly important inside organizations in recent years. Business Process Management extended process design and deployment with process observation and analysis, not only because effective process cost reductions are important, but also to identify problems, measure and improve process performance, and redesign business processes. Process mining is introduced in this chapter, along with its main goals and most relevant and useful techniques and research areas, having in mind Service Oriented Architectures, and more specifically, service orchestrations, and how such techniques are applicable. ProM, a process mining framework is also presented, and some of its mining plugins and features revealed, contextualizing the framework with service orchestrations.

3.1 Domain

Process mining, business process mining, workflow mining or even automated business process discovery (ABPD)[1], aims to extract information from transaction or event logs, and "build" the process models most suitable to represent the behaviour found in such set of real executions. Process mining is the retrieval of "hidden" knowledge about a process or service, under some perspectives, improving the process model itself and its awareness. Its goal is the automated process information discovery from a process event log.

Any information systems such as Customer Relationship Management (CRM), Workflow Management (WfMS), Enterprise Resource Planning (ERP), or Business-to-Business (B2B) systems, have some kind of event log, sometimes referred to as "history", "audit trail", or "transaction log". Often these logs are placed in a single file, several files, or databases.

Table 3.1 shows the process log retrieved from the deployed service orchestration presented in Section 2.3; (i) Process instance refers to each orchestration's case or instance[2]; (ii) Activity refers to each task or instruction. In the case of service orchestrations, these tasks are mainly system activities, decision points or communication ports, but there can be human interaction activities also; (iii) The event type represents the transactional model of each event, and is useful to service

[1]According to Gartner's article, http://blogs.gartner.com/jim_sinur/2009/03/12/automated-business-process-discovery-helps-visually-optimize-processes/

[2]Instance numbers used in Table 3.1 are for simplicity reasons only, since process instance numbers retrieved from the event log are unique identifiers (UUID).

Process Instance	Activity Name	Event Type	Originator	Timestamp
1	Initialization	start	...	28-10-2009 19:33:51.293
1	ReceiveRequest	start	...	28-10-2009 19:33:51.357
2	Initialization	start	...	28-10-2009 19:33:51.370
2	ReceiveRequest	start	...	28-10-2009 19:33:51.370
2	ReceiveRequest	complete	...	28-10-2009 19:33:51.370
2	CheckQuantity	start	...	28-10-2009 19:33:51.370
2	ConstructRequestDenied	start	...	28-10-2009 19:33:51.370
1	ReceiveRequest	complete	...	28-10-2009 19:33:51.573
1	CheckQuantity	start	...	28-10-2009 19:33:51.573
1	SendReqToERP	start	...	28-10-2009 19:33:51.590
2	ConstructRequestDenied	complete	...	28-10-2009 19:33:51.600
2	SendReqDenied	start	...	28-10-2009 19:33:51.600
2	SendReqDenied	complete	...	28-10-2009 19:33:51.600
2	CheckQuantity	complete	...	28-10-2009 19:33:51.600
2	Initialization	complete	...	28-10-2009 19:33:51.600
1	SendReqToERP	complete	...	28-10-2009 19:33:51.620
1	CheckQuantity	complete	...	28-10-2009 19:33:51.620
1	Initialization	complete	...	28-10-2009 19:33:51.620

Table 3.1: Service orchestration's event log

orchestrations when dealing with parallelism, since information about the beginning and ending of each activity is available, as start and complete event types (iv) Originator gives reference to whom has started or executed the activity, the performer; (v) Timestamp is the execution time of each activity.

As one can see from examining the event log, there are two process instances with different activities, hence different paths. All events are totally ordered and their timestamps are quite similar, only diverging from milliseconds. Even though there are two process instances executing, both start and end their execution within the same second, and some different activities are processed at the exact same timestamp. Event logs such as the one presented in Table 3.1 are the starting point for process mining, but it is very important to apply and confront mining tools, and techniques, using event logs taken from real-life applications which model real-life processes. Only this way one can start to overcome process mining challenges and really check its effectiveness (Aalst et al., 2007).

Figure 3.1 illustrates process mining domain and resulting interactions. Considering this reality and introducing it into the service orchestrations domain, the following challenges were identified (Aalst and Weijters, 2004, 2005):

1. Different mining perspectives - The dominant perspective of process mining is the control-flow or process perspective, which is concerned about the ordering of tasks, the process flow. Nevertheless, there are other perspectives of interest, such as: organization, information and application perspectives:

   (a) Control-flow perspective is concerned about the ordering of tasks, the logical flow.

   (b) Organization perspective deals with the organization structure, and can be used to derive work relationships, responsibility, availability, accountability, etc.

   (c) Information perspective refers to management data or production data, like additional process data flow information.

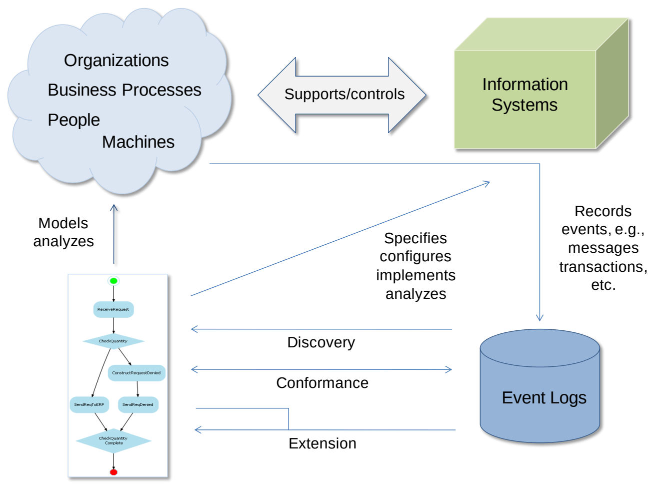


Figure 3.1: The Process Mining domain, adapted from (Aalst et al., 2009)

(d) Application perspective deals with the applications used to execute tasks.

Process mining results are orthogonal to these perspectives, since they may refer to the logical process structure, the process model, or other germane performance issues.

2. Incomplete logs or incompleteness - Rare or inexistent paths may lead to an incorrect process model. This happens when a log is incomplete, not containing enough information to derive the process, and heuristics are needed to undertake this problem. These heuristics are typically based on Occam's razor principle[3]. Sometimes one must assume that what is not present in the event log, does not belong to the process, recognizing a possible underlying mistake (Aalst et al., 2007).

3. Concurrency or parallelism - Distinction between start and end events in the event log helps detecting concurrent activity execution (Wen et al., 2004).

4. Information and log integration from heterogeneous sources - MXML, as in Mining-XML, is a standard XML format that can be used to tackle this issue. This format permits to import event logs from distinct systems and this way allowing an abstraction of each heterogeneous source (Günther and van der Aalst, 2006; Aalst et al., 2007). Figure 3.2 illustrates the MXML format used in process mining, using a UML class diagram. Considering the event log from Table 3.1, the mappings for each element are as follows; Source element contains information about the software or system that was used to record the log; The process element indicates the process holding multiple cases, or instances, so this way it is possible to include several processes in one MXML log file; Every *ProcessInstance* can contain several *AuditTrailEntry* elements, each representing one activity. *WorkflowModelElement* represents the activity name in the event log; The *Data* element can be used at various levels to add additional information.

[3]Occam's razor principle, *entia non sunt multiplicanda praeter necessitatem*, is the principle that states, "When you have two competing theories which make the same predictions, the simplest explanation or strategy tends to be the best one".

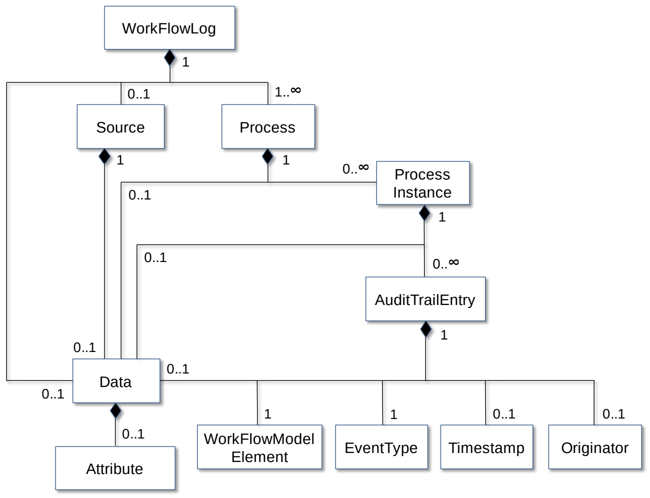


Figure 3.2: UML diagram of the MXML format (Günther and van der Aalst, 2006)

5. Visualize and interpret results - Visualizing the complete control-flow or other perspectives is not as straightforward as one may think. Presenting process mining results in a way that makes possible to gain clear insight of the process is a challenge, often mentioned as "management cockpit"[4].

6. Model analysis and conformance - Aims to detect discrepancies between process model and deployed process execution (viz. Delta Analysis). In (Aalst et al., 2003b, 2004), a workflow mining technique is introduced, with the purpose of collecting data from Workflow Management Systems (WfMS), and applying delta analysis to diagnose and redesign the model. This approach results in an "*a posteriori*" process model that can be compared with the *"a priori"* model. In (Rozinat and van der Aalst, 2008), an incremental approach to check the conformance of processes based on monitoring real behaviour is presented. The notions of fitness and appropriateness are introduced. Fitness measures how the executed process complies with the specified behaviour found in the previously designed model, hence how the event log fits the model. Appropriateness measures if the model describes the observed behaviour in a suitable way. The idea of Occam's razor is once again captured, i.e., "one should not increase, beyond what is necessary, the number of entities required to explain anything". Appropriateness can be distinguished between two categories:

   (a) Structural - The simplest model should be chosen to explain the retrieved behaviour from the event log.

   (b) Behavioural - The model should not be too generic and allow for too much behaviour.

7. Model extension - Discover information that will enhance the process model.

Unlike classical data mining, the focus of process mining is the process, aiming to extract, from a set of event logs, several process-oriented information. This information may be used, amongst other things, to detect the conformance between the existing and conceived process model, when it exists,

[4]See Fujitsu's Automated Process Discovery and Visualization Service, http://www.fujitsu.com/global/services/software/interstage/abpd/

and the “real” process, learned from real performance. However, process mining must often work side by side with the organization’s knowledge about the modelled process to accomplish positive results (Aalst et al., 2007, 2009). Dissimilarities between real and idealized models often happen since it is not easy to predict every possible situation when designing one business process. Moreover, actual times require high flexibility and strong dynamism, meaning that business processes must be ready to adapt easily.

The objective usefulness of mined process information for an organization can be classified by three different perspectives:

- Support the execution of new systems.
- Facilitate business process improvement and redesign.
- Achieve control and knowledge over business processes.

Through process mining, it is possible for an organization to attain feedback to some questions and understand what is really going on, control is seized. Considering service orchestrations, these are some of the relevant questions that can be answered using process mining techniques:

1. How is the service orchestration modelled? What are the business rules? Are they coherent and executed correctly?
2. How was the service orchestration built? How many versions does it have? Can one see its growth along time?
3. What are the most frequent paths and their execution probabilities along the service orchestration?
4. What is the average/maximum/minimum performance time for each activity and for the whole orchestration? And for external services, whenever present?
5. Is the process model efficient and adequate for its purpose? Are there any bottlenecks? What are the critical paths?
6. How does the service orchestration’s run-time behaviour conform with the previously designed model?
7. What are the external service communications?
8. What is the service architecture in the organization?

The following section will present some techniques that will help to tackle these questions, and find some meaningful answers.

The interest to monitor and seize control over business processes has been growing inside organizations over the last years. Furthermore, new legislation and regulation acts start to impose more internal control requirements, therefore increasing the pressure for higher governance maturity levels inside organizations (Ferreira and Mira da Silva, 2008; Zhang, 2007).

Top management needs to better understand their organizations, how they really work, and not how they think they work. “The key to achieve this goal and completely read one organization, is to fully realize its organizational DNA, the sequence of activities that use, alter or process several resources, physical, informational or human, i.e. the organization’s business processes” (Tribolet, 2005). Process mining is here to help us grasp this knowledge.

3.2 Mining Techniques

When applying process mining techniques, it is of most relevance to understand that there is a strong relationship between the mining algorithm and the type of challenges it can successfully handle (cf. Section 3.1). Although every process mining technique extracts process-oriented information from an event log, not all mining techniques are best suited for every perspective and log related issues. Also, it is very important to realize at the starting point which is the type of process model that best fits with the data from the workflow log, so it is possible to choose the best process representation language (Petri net, block-oriented process models, event dependency models, etc). This choice strongly influences the mining algorithm[5]. Local-global dimension also has a strong role when selecting the most adequate mining algorithm. Local strategies are based on local information and build the process model on a step by step basis. Global strategies try to gather every possible information and build the process model with only one search (Aalst and Weijters, 2004; Medeiros et al., 2007).

Taking this information into consideration and applying it into the service orchestrations analysis domain, one can find some issues that need to be regarded. Service orchestrations are often built and deployed within several iterations, which leads us to the incompleteness of some log events. This issue can be dealt with by using two different approaches:

1. Considering several orchestration service versions.

2. Using robust heuristic mining algorithms.

This way it is possible to focus the analysis on the process instead of modelling every behaviour learned from the event log. Considering the control-flow perspective, one need only consider the process instance and its activities. Since logs come originally ordered, using timestamp, although it might help to gather dependency and causality, is not relevant. With this information, different techniques may be applied, such as:

1. α-algorithm - Builds a Petri net that models the process learned from the event log, but has some limitations, like concurrency and incompleteness problems (Aalst and Weijters, 2005; Aalst et al., 2004; Medeiros, 2006).

2. β or Tsinghua-α algorithm - Exploits the fact that tasks have execution timestamp information, to build modelled processes using Petri nets. It is related to the α algorithm but with a different approach, since it uses start and complete event types to explicitly detect parallelism (Wen et al., 2004).

3. Heuristic miner - Uses a frequency based metric to indicate certainty of dependency or causality relations. It is robust for concurrent processes and real-life logs (Weijters and van der Aalst, 2003, 2002; Aalst et al., 2007).

4. Genetic miner - Is an adaptive global search method in which every process instance is evaluated according to a fitness measure that quantifies how well does every process instance exhibits the behaviour "learned" from the log. It deals well with noise and incompleteness (Medeiros et al., 2005; Aalst et al., 2005; Medeiros et al., 2004, 2007; Medeiros, 2006).

[5]Inductive Bias - Guessed outputs are strongly refined by prior information, or inputs. In practice, many process mining algorithms have a strong inductive bias.

Process mining's organization perspective is concerned about the resource that executed or initiated one specific activity, also called originator. This information can be used to derive knowledge about roles and groups, collaboration relationships and efficiency, hence, the organizational structure. Service orchestrations are mainly system tasks that control the information flow, apply business rules, and interconnect other services, so it seems more interesting to consider a different perspective that could be mentioned as an interoperability or even choreography perspective (cf. Section 2.4). Both metrics, work transfer and subcontracting (Bozkaya et al., 2009), could still be used to discover the organization's service architecture network. Therefore, this analysis must be done at the process level and not at the activity level, unless there is some kind of integration platform for heterogeneous services or logs. If such a log integrator was implemented, then the originator field could have a different meaning. It would be important to know, in an interoperability perspective, which service or node executed each activity, so, using this approach, the originator field can be used to keep this trace. Considering each service as a node, it would also be interesting to verify and study each node's links and centrality, using centrality metrics (Borgatti, 2005) (e.g., betweenness, in and out closeness, etc), building what can be described as a super-process or multiple service orchestration, a distributed orchestration choreography (cf. Section 2.4) (Pedraza and Estublier, 2009).

3.3 Mining Web Services

As discussed in Chapter 2, Service Oriented Architectures play an increasing role in enterprise business process integration. Services are used to implement an organization's functionality that can be utilized by other organizations, enabling service integration, and business process interoperability. Service orchestrations define another abstraction layer in which services are built upon existing services, hence service reutilization and composition. Even though service orchestrations enable service composition and therefore business collaboration, they are centralized services that can be deployed as web services and communicate with other service orchestrations, becoming integration enablers.

Web service mining has been introduced by (Gombotz et al., 1999) with the purpose of analysing web service interaction, according to two different challenges:

- Discovering complex patterns within web services.
- Supervising and monitoring web services.

In (Gombotz et al., 1999; Dustdar et al., 2004; Gombotz and Dustdar, 2005; Dustdar and Gombotz, 2007), a web services interaction mining architecture (WSIM) is proposed, in which three levels of abstraction are presented, and a normative log format, or specification, is introduced for each level:

1. **Web service operational level** Observes only one single web service and its internal behaviour. The proposed log specification for this level contains:
   (a) Activity, meaning operation name.
   (b) Performer, the client web service.
   (c) Status, either start or complete.
   (d) Timestamp, each activity's execution time.

As one can observe, the presented log specification is close related to the MXML log file format (cf. Section 3.1).

2. **Web service interaction level** Observes only one single web service, but with regards to its interactions with other services. In these interactions, services are considered to act either as a consumer or a provider. To achieve this step, interactions must be logged. Types of interaction are presented as being; one-way; request-response; solicit-response and notification.

3. **Web services workflow level** Observes the overall interactions and collaboration between web services (Novatnack and Koehler, 2004). It tries to identify workflows, or business processes implemented using web services. The problem is that in order to successfully mine exact distributed workflow models, workflow log information must be present in service-oriented systems. In (Gombotz and Dustdar, 2005; Dustdar and Gombotz, 2007), a web services workflow mining approach is presented, along with a classification of service-oriented systems according to the richness of the log data they provide. However, no service orchestration or BPEL[6] environments are used. (Gaaloul et al., 2008) presents a patterns mining algorithm that aims to discover the implicit orchestration process based on analysing individual patterns. A statistical technique is used to discover patterns in execution logs from a set of web services that interact with each other. Using this distributed interaction logic, since there is no orchestration process, the algorithm formalises the orchestration logic and explores if it can be mapped to an explicit BPEL language.

The presented mining approaches are never related to explicit service orchestration, instead, they relate to implicit orchestrated processes, in a sense that each web service's behaviour and patterns are analysed so as to build the overall and distributed system behaviour. According to (Gombotz and Dustdar, 2005; Dustdar and Gombotz, 2007), the web services interaction mining architecture (WSIM) is one step behind web service orchestration, since it deals with service-oriented systems that do not apply web service orchestration tools. The former web services workflow level is indeed more related to service choreographies, whereas the web services operational level and the interaction level together with service workflow and orchestration logic can be considered as a service orchestrations' mining approach. Both levels relate to one single web service and its interactions with other services, and so, introducing workflow or orchestration logic, they become service orchestrations.

3.4 ProM: a Process Mining Framework

Although there are currently some academic and commercial tools self-proclaimed as Business Process Analysis (BPA), Business Activity Monitoring (BAM), Business Operations Management (BOM) or Business Process Intelligence (BPI) applications, such as IBM's Cognos[7], or SAP's BusinessObjects[8], the truth is that many of them are limited to performance analysis, not enabling real process discovery, the control-flow perspective (Dongen et al., 2005; Aalst et al., 2007).

ProM is a "pluggable" environment framework designed for process mining. Since there are many information systems, each of them with its own log file format, ProM has developed a generic

[6]BPEL, namely WS-BPEL, was mentioned in Section 2.3.
[7]IBM Cognos, http://www-01.ibm.com/software/data/cognos/
[8]SAP BusinessObjects, http://www.sap.com/solutions/sapbusinessobjects/index.epx

XML format to store a log in, named MXML (cf. Section 3.1). After mapping the event log to the MXML format, ProM can be used to start process mining analysis, from every perspective, and several mining plugins can be applied. It is also possible to add new plugins and export/import results to/from other formats. Figure 3.3 presents an overview of the ProM framework, depicting the relations between the framework, the process log format MXML, and the several plugin types.

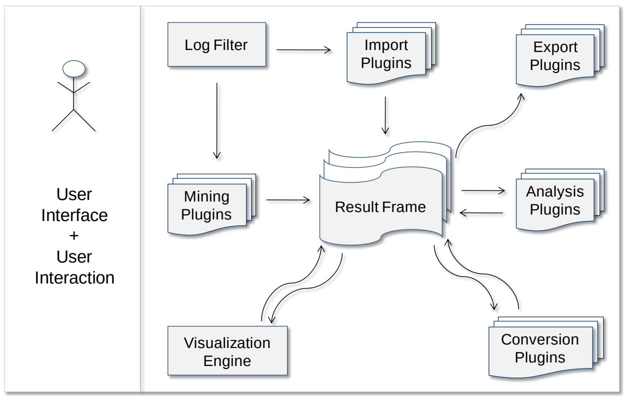


Figure 3.3: ProM framework (Dongen et al., 2005)

There are three main types of mining plugins available (cf. Figure 3.1) (Aalst and Verbeek, 2008; Aalst et al., 2009), namely:

- Discovery plugins: Based only on data retrieved from the event log.
- Conformance plugins: Based on an event log and deployed models. These plugins try to find matching behaviour between mined event logs and the process model, hence if real process execution conforms with the designed model.
- Extension plugins: Try to improve and extend deployed models through process mining.

In the context of service orchestrations, discovery and extension mining plugin types are useful and can be applied. Conformance plugins can only be applied if parsed service orchestration models are exported. Although ProM can be used to analyse service orchestrations and therefore help organizations to answer some of the questions stated before (cf. Section 3.1), there are still some points that actual process mining paradigms cannot completely deal with, namely questions 2, 7 and 8. Therefore, one solution is presented in the next chapters, based on a process mining methodology applied to service orchestrations, and a software application that implements this methodology and unveils these questions.

3.5 Conclusion

This chapter has discussed process mining, presenting its major goals, challenges, research areas and useful techniques concerning Service Oriented Architectures, and, more specifically, service orchestrations. Some of the relevant questions and problems concerning service orchestrations were presented, such as incompleteness and parallelism, along with mining techniques that may be applied to deliver solutions. ProM, a process mining framework, was also presented, and its relevance concerning the analysis and discovery of service orchestrations discussed. In the following chapter, a proposed solution based on a process mining methodology, with the purpose of analysing event logs from deployed service orchestrations and discovering the orchestration's run-time behaviour, unveiling process-aware and performance information, will be presented.

Chapter 4

Mining Service Orchestrations

The proposed solution is based on gathering information from the deployed orchestration's event log, and using process mining techniques so as to deliver useful information, such as run-time behaviour and other process-aware and performance information. A service orchestrations' mining methodology will be introduced in this chapter, where every phase is discussed and explained with more detail. The proposed solution will be demonstrated using Microsoft Biztalk integration platform, and orchestration from Section 2.3 used as an example to verify delivered results.

4.1 Methodology

This solution is focused mainly on the control-flow, or process perspective, since the main concern is in gathering the service orchestration's design model. Besides the logical model, some performance issues are also of interest. Figure 4.1 illustrates the main steps involving the proposed ADSO process mining methodology, where ADSO stands for "Analysis and Discovery of Service Orchestrations".

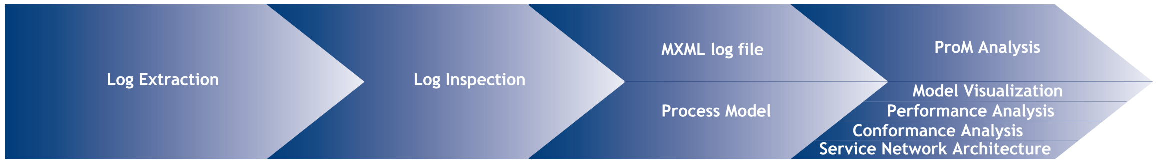


Figure 4.1: Analysis and discovery of service orchestrations (ADSO), the mining methodology

To perform the depicted ADSO mining methodology, the following phases are presented (Aalst et al., 2009; Bozkaya et al., 2009):

1. Log extraction: Identify every relevant information and where it is stored by the integration platform. It is important to notice that since the objective is to obtain the process model performed by the orchestration service[1], the orchestration's design model, no log information from presently running and still incomplete orchestration instances will be extracted. This way, some of the problems related to the following inspection step (cf. Section 3.1) are avoided and may be solved *"a priori"* (e.g., incompleteness).

[1]Referentiates the orchestration itself as an available service or process, and not the external services invoked by the orchestration. To avoid misinterpretation, whenever an external service is considered, it will be referenced explicitly.

2. Log inspection: Event logs are retrieved and pre-processed. Errors or some inconsistencies are checked and unnecessary information is removed. This step is also useful when applying filters, like choosing process instances by date or by deployment version.

3. Log aggregation: The ordering of activities is retrieved, and the executed logical flow modelled. There are two possible ways here, namely:

   (a) MXML log file: The pre-processed log file is mapped to MXML format (cf. Section 3.1). *WorkflowModelElement*, *EventType*, *Timestamp* and *Originator* elements are used. Considering the event types found in the logs, only two types from the transactional model (Dongen et al., 2005) were chosen, start and complete. The MXML log file can therefore be used by another system, such as the ProM framework, to retrieve the process' logical flow.

   (b) Process model: Using a control-flow perspective, data is mined and the orchestration's design model is created. A dependency frequency table (Weijters and van der Aalst, 2002, 2003) is built to achieve this step.

4. ProM analysis: Process mining analysis can be performed by the ProM framework, using the previous MXML log file.

5. Model Visualization: The orchestration's design model can be visualized, by means of a dependency graph.

6. Performance analysis: Activity and process performance can be analysed, regarding execution times and path probabilities.

7. Conformance analysis: The execution or run-time behaviour of the mined orchestration can be compared with the idealized and initially designed model. Both models can be matched together, showing any existing discrepancies between the initial process model and the deployed process execution (Rozinat and van der Aalst, 2008).

8. Service network architecture: External communications are analysed and reported, which can be used to discover the wider service architecture (cf. Figure 2.6).

All phases involved in the ADSO methodology can be seen as the *breadcrumbs* to discover process-aware information about service orchestrations. Each step will be discussed in detail in the following sections.

4.2 Log Extraction

Deployed orchestrations leave a trace of their executed activities and other related information in a history log. In the case of the deployed orchestration presented in Section 2.3, and considering again Microsoft Biztalk integration platform, this data is stored in a Microsoft SQL Server database. The retrieved process log, the orchestration's event log presented in Table 3.1, shows only a few of the myriad of data found in such execution generated logs. The purpose of log extraction is to identify and correlate every relevant data found about orchestration instances, namely:

- Process or service instance id itself, which refers to every orchestration's case identification. Service instance ids are unique identifiers (UUID).
- Activity or instruction ids, which are unique identifiers (UUID) and specific to the integration platform. Instruction ids will not be present in the process' event log, they are mainly used to relate instruction names and types.
- Activity or instruction name, a text identifying every task performed by the orchestration.
- Activity or instruction type, which indicates the specific BPEL[2] activity type. This information is not relevant to the log aggregation phase, however, it will be very useful when performing phase 2, log inspection, and phase 5, visualization, designing the process model.
- Event type, that is to say the transactional model of each event. The event types used are only start and complete event types. As referred before, this information is very important when dealing with concurrency and parallelism, since it helps to understand every activity's precedence.
- Originator, defines who has started or executed the activity. Considering service orchestrations, all activities are performed by the system, even when human interaction is present. Therefore, it is considered that the server or node name is indeed the activity performer. This conclusion is also of utmost interest when defining service interoperability, as it enables gathering each activity's service placement information in a distributed orchestration network (cf. Section 2.4), a choreography.
- Timestamp, the start and end time of each orchestration's instance and execution time of every inherent activity. As presented before in Table 3.1, activity timestamps can sometimes diverge from milliseconds. Orchestration instances and their activities are ordered according to their corresponding timestamps.
- Orchestration version. Version ids, which are also unique identifiers (UUID), and their deployment timestamps are retrieved. Version information is very useful, as it enables orchestration growth analysis, which helps to understand how the orchestration was developed along time, and also improves designed model understandability.
- Communication ports. Port name, direction[3] and timestamp information are retrieved. Even though port retrieval and perception is not specific process-aware information, it will become helpful when defining the service network architecture and interoperability.

Instance information in Microsoft Biztalk can be found using tables from the simplified database model depicted in Figure 4.2. Table *dba_ServiceFacts* keeps information about each orchestration's name, state, either complete or terminated, start and end time, version, and instance identification (UUID), amongst others. Table *dta_ServiceInstances* stores information about each orchestration's instance execution mappings, such as *serviceID*, which indicates the orchestration's version, start and end times, and other execution related data, like error information. Table *dta_DebugTrace* keeps information about each instance's activity sequence and execution time, stored as begin and end timestamps.

[2]BPEL, namely WS-BPEL, was mentioned in Section 2.3.
[3]Communication ports are mainly send and receive direction ports.

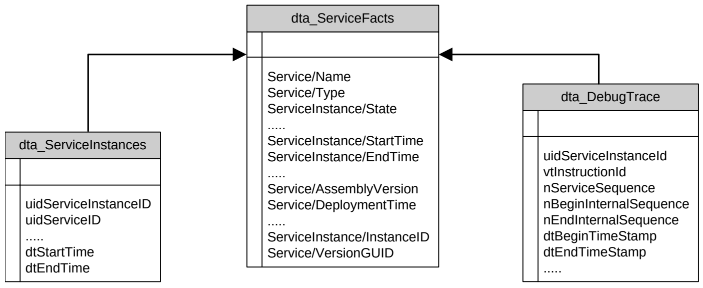


Figure 4.2: Microsoft Biztalk simplified database model with orchestration instance information

Only information about finished orchestration instances is gathered, either completed or terminated instances[4]. A terminated instance represents an orchestration's execution that did not complete its entire flow and was terminated either by the user, or system, or by an error. Completing its entire flow, means that the orchestration's instance starts in one start activity, and ends in one predicted end activity[5]. The purpose of retrieving only completed or terminated instances information, is to solve some problems about incompleteness that could arise if information about instances in execution were to be gathered.

4.3 Log Inspection

After identifying and correlating every relevant data about deployed orchestration instances, event logs are retrieved and pre-processed, errors and inconsistencies checked and unnecessary information removed. Needless or problematic data concerning any specific integration platform must be distinguished and dealt with, such as:

- Reserved activity or instruction ids. Some activity ids may be reserved by the integration platform. Unnecessary activity ids are removed.
- Reserved activity or instruction names. Since instruction ids relate to instruction names, there may exist reserved activity names that must be recognized.
- Consistent activity or instruction names. Different activities must not have the same name. However, the same activity may appear several times in the process. Since it is the responsibility of the orchestration's developer to name activities during development, and considering again Microsoft Biztalk integration platform, where activity name checking is not done, this constitutes as a problem. Without name consistency, the control-flow perspective of the mined orchestration is not altered, although the mined process model may be misinterpreted. Nevertheless, other process mining results, like performance issues, are hugely degraded, since different activities are misguided as being similar. One way to deal with this issue is to correlate activity names with their corresponding scope in the orchestration's process, whenever scope type activities are present[6]. If an activity presents itself "inside" a scope type activity, its name is changed to *"Scope Activity Name.Activity Name"*. This solution, however, does

[4]Service orchestration instances can be in execution, completed or terminated states.
[5]Last activity executed by one orchestration instance.
[6]A BPEL scope type activity is a collection of activities, where local variables, messages and correlations may be used.

not solve every problem, since scopes may not be present, and even when they are, it is always up to the developer to decide each activity's name. Another possible solution is to label activities using their unique position in the event log, and therefore in the process, using the previous and/or following activities to deliver name consistency.

- Reserved activity types. There may exist some activity types that are not standard BPEL activities. These cases must be analysed, identified and correctly interpreted. Others may have specific integration platform names, so the correct "translation" to the BPEL standards is useful when designing the process model. Scope type activities, when present, must be identified and their internal activities correlated together, to assure consistent activity naming, as referred before.
- Reserved event types. Since the transactional model used for every event consists only in start and complete event types, others are discarded.
- Timestamp formats can differ, so a standard should be established and used accordingly. The MXML timestamp standard format can be a good choice, since the resulting pre-processed event log will be mapped to this format in the log aggregation phase.

Figure 4.3 shows one SQL query that retrieves orchestration instance information. Once more, the deployed orchestration presented in Section 2.3 is used. As one can observe in the SQL query, the *ServiceInstanceId*, representing the process case identification, the orchestration's instance, has the UUID "2f4157c0-0be0-49e6-a7be-c43d74e05b50", and was used as an example. Applying the presented SQL query, it is possible to gather each instance's activity flow information:

- Activity name "Initialization", presented in Table 3.1, is a reserved activity name and activity type, with the hard-coded id "e211a116-cb8b-44e7-a052-0de295aa0001" (cf. lines 16 and 21). This activity merely indicates the starting and ending of each orchestration's instance.
- *Naction*, is a reserved and specific integration platform code for an event type, where code 1 represents start event types, and code 2 complete event types (cf. lines 3, 9, 17 and 22). *InternalSequence* represents each activity's placement in the instance's activity flow.
- All start event type activities are selected (cf. lines 3 to 8). Each activity's begin timestamp is selected as the activity's timestamp, and *nAction* is hard-coded to 1. Only activities from the selected orchestration's instance, which must not be in execution, are selected (cf. lines 39 and 40).
- All complete event type activities are selected (cf. lines 9 to 14). Each activity's end timestamp is selected as the activity's timestamp, and *nAction* is hard-coded to 2.
- Reserved "Initialization" activity is added with both *nAction* values (cf. lines 16 to 19 and 21 to 23). Orchestration's instance's start time and end time are selected as both timestamps, start and complete's, and last *InternalSequence* is calculated (cf. lines 24 to 38).
- Reserved instruction ids with the UUID "00000000-0000-0000-0000-000000000000" are removed (cf. line 42).
- Activities are ordered according to an internal sequence that complies with their timestamps (cf. line 43).

```
SELECT t.uidServiceInstanceId as sid, t.vtInstructionId as iid, t.nAction as nact, t.dtTimeStamp as ts
  FROM dbo.dtav_ServiceFacts sf WITH (READPAST), (
    SELECT dt.uidServiceInstanceId, dt.vtInstructionId, 1 as nAction, dt.dtBeginTimeStamp as dtTimeStamp,
            dt.nServiceSequence, dt.nBeginInternalSequence as nInternalSequence
      FROM dta_ServiceInstances s JOIN dta_DebugTrace dt WITH (READPAST)
      ON dt.uidServiceInstanceId = s.uidServiceInstanceId
      AND dt.nServiceSequence = 0 WHERE s.uidServiceInstanceId = '2f4157c0-0be0-49e6-a7be-c43d74e05b50'
    UNION ALL
    SELECT dt.uidServiceInstanceId, dt.vtInstructionId, 2 as nAction, dt.dtEndTimeStamp as dtTimeStamp,
            dt.nServiceSequence, dt.nEndInternalSequence as nInternalSequence
      FROM dta_ServiceInstances s JOIN dta_DebugTrace dt WITH (READPAST)
      ON dt.uidServiceInstanceId = s.uidServiceInstanceId
      AND dt.nServiceSequence = 0 AND (dtEndTimeStamp IS NOT NULL)
      WHERE s.uidServiceInstanceId = '2f4157c0-0be0-49e6-a7be-c43d74e05b50'
    UNION ALL
      SELECT TOP 1 s.uidServiceInstanceId, 'e211a116-cb8b-44e7-a052-0de295aa0001' as vtInstructionId,
                1 as nAction, s.dtStartTime as dtTimeStamp, 1 as nServiceSequence, 1 as nInternalSequence
        FROM dta_ServiceInstances s
        WHERE s.uidServiceInstanceId = '2f4157c0-0be0-49e6-a7be-c43d74e05b50'
    UNION ALL
    SELECT TOP 1 s.uidServiceInstanceId, 'e211a116-cb8b-44e7-a052-0de295aa0001' as vtInstructionId,
              2 as nAction, s.dtEndTime as dtTimeStamp, 1 as nServiceSequence, t2.nInternalSequence
      FROM dta_ServiceInstances s, (
        SELECT MAX(t.nInternalSequence) + 1 as nInternalSequence,
              '2f4157c0-0be0-49e6-a7be-c43d74e05b50' as uidServiceInstanceId
          FROM (
            SELECT MAX(nBeginInternalSequence) as nInternalSequence
              FROM dta_DebugTrace
              WHERE uidServiceInstanceId = '2f4157c0-0be0-49e6-a7be-c43d74e05b50'
              AND nServiceSequence = 0
            UNION ALL
            SELECT MAX(nEndInternalSequence) as nInternalSequence
              FROM dta_DebugTrace
              WHERE uidServiceInstanceId = '2f4157c0-0be0-49e6-a7be-c43d74e05b50'
              AND nServiceSequence = 0
            UNION ALL
            SELECT 1 as nInternalSequence ) as t
                                    ) as t2
      WHERE s.uidServiceInstanceId = '2f4157c0-0be0-49e6-a7be-c43d74e05b50'
      AND s.dtEndTime IS NOT NULL AND s.uidServiceInstanceId = t2.uidServiceInstanceId ) as t
    WHERE sf.[ServiceInstance/InstanceID] = t.uidServiceInstanceId
    AND t.vtInstructionId != '00000000-0000-0000-0000-000000000000'

ORDER BY [ServiceInstance/StartTime], t.uidServiceInstanceId, t.nInternalSequence
```

Figure 4.3: Service orchestration's instance information retrieval in Microsoft Biztalk integration platform

Table 4.1 shows the orchestration's instance information gathered using the SQL query presented in Figure 4.3.

Instance ID	Instruction ID	Event Type	Timestamp
2F4157C0-0BE0-49E6-A7BE-C43D74E05B50	E211A116-CB8B-44E7-A052-0DE295AA0001	1	2009-10-28 19:40:33.730
2F4157C0-0BE0-49E6-A7BE-C43D74E05B50	DBE654A3-1E90-48E1-B616-50E7B7602ACC	1	2009-10-28 19:40:33.730
2F4157C0-0BE0-49E6-A7BE-C43D74E05B50	DBE654A3-1E90-48E1-B616-50E7B7602ACC	2	2009-10-28 19:40:33.730
2F4157C0-0BE0-49E6-A7BE-C43D74E05B50	977695E2-DF0D-49F0-8B7B-E39885805505	1	2009-10-28 19:40:33.730
2F4157C0-0BE0-49E6-A7BE-C43D74E05B50	C7575D71-7E30-462B-8E10-87EC51E74D9D	1	2009-10-28 19:40:33.730
2F4157C0-0BE0-49E6-A7BE-C43D74E05B50	C7575D71-7E30-462B-8E10-87EC51E74D9D	2	2009-10-28 19:40:34.043
2F4157C0-0BE0-49E6-A7BE-C43D74E05B50	64E9C10C-B1AF-4573-A2F4-8E41C69A7777	1	2009-10-28 19:40:34.043
2F4157C0-0BE0-49E6-A7BE-C43D74E05B50	64E9C10C-B1AF-4573-A2F4-8E41C69A7777	2	2009-10-28 19:40:34.090
2F4157C0-0BE0-49E6-A7BE-C43D74E05B50	977695E2-DF0D-49F0-8B7B-E39885805505	2	2009-10-28 19:40:34.090
2F4157C0-0BE0-49E6-A7BE-C43D74E05B50	E211A116-CB8B-44E7-A052-0DE295AA0001	2	2009-10-28 19:40:34.090

Table 4.1: Service orchestration's instance information in Microsoft Biztalk integration platform

Instruction ids are present with their specific event type codes and timestamps. As one should expect, instance ids are identical. Subsequently, the log inspection phase is responsible to map every instruction id with their corresponding activity names and types, translate specific event type codes to standard transactional model labels, start and complete event types, and apply a standard timestamp format.

Corresponding activity names and types are found using instance's id and start time timestamp, and version's id and deployment timestamp information. Since each instance only belongs to one orchestration version, using timestamps, both instance's and version's, will deliver the necessary search granularity needed to discover related activity data. Figure 4.4 presents the SQL query used to discover the corresponding activity names and types. *UidServiceId* identifies the orchestration's

```
1.  SELECT TOP 1 txtSymbol as os
2.    FROM dta_ServiceSymbols sym, dta_ServiceInstances inst
3.    WHERE sym.uidServiceId = '3acc982d-b315-483b-111b-9cc65ce842ea'
4.    AND inst.uidServiceInstanceId = '2f4157c0-0be0-49e6-a7be-c43d74e05b50'
5.    AND DATEDIFF(ss, sym.dtDeploymentTime, inst.dtStartTime) > 0
6.  ORDER BY DATEDIFF(ss, sym.dtDeploymentTime, inst.dtStartTime) ASC
```

Figure 4.4: Service orchestration's activity information discovery in Microsoft Biztalk integration platform

version (cf. line 3); *UidServiceInstanceId* represents the same former orchestration's instance id (cf. line 4); Line 5 shows the mentioned search granularity, in the order of seconds.

A XML type document that includes all information regarding each activity name and type is retrieved, as can be seen in Figure 4.5. Translation between instruction id, which is presented in the document as *ShapeID*, and the corresponding activity name, *shapeText*, and activity type, *shapeType*, is now easily accomplished. As one can observe, the document presents not only every instruction id from Table 4.1, but also all activities that comprise the mined process model. Considering the orchestration's activity information gathered, one can find that there is one activity, "SendReqToERP", that was not executed by the orchestration's instance.

At this point, having discovered and mapped activity information, the log is finally human-readable, with every essential information available. Table 4.2 shows the resulting pre-processed event log after inspection.

Instance ID	Activity Name	Event Type	Originator	Timestamp
3	Initialization	start	Eclipse	2009-10-28T19:40:33.730
3	ReceiveRequest	start	Eclipse	2009-10-28T19:40:33.730
3	ReceiveRequest	complete	Eclipse	2009-10-28T19:40:33.730
3	CheckQuantity	start	Eclipse	2009-10-28T19:40:33.730
3	ConstructRequestDenied	start	Eclipse	2009-10-28T19:40:33.730
3	ConstructRequestDenied	complete	Eclipse	2009-10-28T19:40:34.43
3	SendReqDenied	start	Eclipse	2009-10-28T19:40:34.43
3	SendReqDenied	complete	Eclipse	2009-10-28T19:40:34.90
3	CheckQuantity	complete	Eclipse	2009-10-28T19:40:34.90
3	Initialization	complete	Eclipse	2009-10-28T19:40:34.90

Table 4.2: Pre-processed event log from the log inspection phase

When log inspection is successfully completed, event logs are built and ready to be aggregated. Every event log must be "clean", having only the relevant information identified in the log extraction phase. This way, it will be possible to mine this data and also apply specific mining filters in the following phase, log aggregation.

```
<XsymFile>
  <ProcessFlow xmlns:om="http://schemas.microsoft.com/BizTalk/2003/DesignerData">
.......
    <ShapeInfo>
      <shapeType>ReceiveShape</shapeType>
      <ShapeID>dbe654a3-1e90-48e1-b616-50e7b7602acc</ShapeID>
      <ParentLink>ServiceBody__Statement</ParentLink>
      <shapeText>ReceiveRequest</shapeText>
      </children>
    </ShapeInfo>
.......
    <ShapeInfo>
      <shapeType>DecisionShape</shapeType>
      <ShapeID>977695e2-df0d-49f0-8b7b-e39885805505</ShapeID>
      <ParentLink>ServiceBody__Statement</ParentLink>
      <shapeText>CheckQuantity</shapeText>
      <children>
.......
        <ShapeInfo>
          <shapeType>ConstructShape</shapeType>
          <ShapeID>c7575d71-7e30-462b-8e10-87ec51e74d9d</ShapeID>
          <ParentLink>ComplexStatement__Statement</ParentLink>
          <shapeText>ConstructRequestDenied</shapeText>
          <children>
.......
          </children>
        </ShapeInfo>
        <ShapeInfo>
          <shapeType>SendShape</shapeType>
          <ShapeID>64e9c10c-b1af-4573-a2f4-8e41c69a7777</ShapeID>
          <ParentLink>ComplexStatement__Statement</ParentLink>
          <shapeText>SendReqDenied</shapeText>
          </children>
        </ShapeInfo>
.......
        <ShapeInfo>
          <shapeType>SendShape</shapeType>
          <ShapeID>ce56e77d-119e-4693-b7dc-889ecc98d811</ShapeID>
          <ParentLink>ComplexStatement__Statement</ParentLink>
          <shapeText>SendReqToERP</shapeText>
          <children />
        </ShapeInfo>
      </children>
      </ShapeInfo>
.......
  </ProcessFlow>
.......
</XsymFile>
```

Figure 4.5: Service orchestration's activity information in Microsoft Biztalk integration platform

4.4 Log Aggregation

With the pre-processed orchestration's event log at hand, having all process instances and inherent activities ordered, it will be easier to retrieve the process' ordering of tasks, and model the deployed and executed logical flow. Log aggregation enables the discovery of the orchestration's run-time behaviour design model. Other process mining results can be achieved after this phase, like performance issues, conformance analysis, or model extensions.

The main ideas beneath the log aggregation phase are to bring together each instance's activity and communication information, either from every instance found in the previous phase or by filtering chosen instances, and to build a dependency frequency table that models the mined orchestration's control-flow perspective. Aggregated instance lists may be refined using three different filters, namely:

1. Orchestration version.

2. Instance timestamp.

3. Instance id.

Instance activity ordering information retrieved from the pre-processed event log can then be mapped and placed in a MXML log file, or mined, so as to build a dependency frequency table that describes the orchestration's model.

4.4.1 MXML Log File

The purpose of MXML log files is to create a common log format for process mining techniques, which enables information and log integration from heterogeneous sources. As mentioned before, MXML is a standard XML format that is used to import or export event logs from or to several distinct systems, like the ProM framework. In Section 3.1, the MXML format was presented and some of its element mappings explained. Considering every relevant data found about orchestrations instances presented in the log extraction phase, MXML log files can be produced using the following standard mappings:

- *Process id*, which indicates the orchestration's identification, or name. It is possible to place several service orchestrations in one MXML log file.
- *ProcessInstance id*, representing each orchestration's instance.
- *WorkflowModelElement*, which maps the activity name.
- *EventType*, maps the transactional model of each event. As referred before, only start and complete event types are used.
- *Timestamp*, represents the execution time of each activity in every instance.
- *Originator*, the activity performer. The server name is used, as mentioned before, since it is important to know, in an interoperability perspective, which server node executed each service and activity.

Figure 4.6 shows the MXML log file generated using the pre-processed event log from Table 4.2. For simplification reasons only, the presented log file only has one instance from the service orchestration introduced in Section 2.3. Although the MXML log file format was initially designed to store "standard" process mining information, it can easily accommodate more specific data. The *Data* element presented in the standard MXML log format may be used to achieve this goal. Considering service orchestrations, other important information should also be stored in such a log file, even considering that for the moment, process mining frameworks, like ProM, are not able to understand such orchestration's specific knowledge, namely:

- Activity type, or BPEL activity type, important when performing phase 5, visualization;
- Orchestration version, enabling process growth analysis and improved design model comprehension; and
- Communication ports, helpful when defining the service network architecture and interoperability,

```
<?xml version="1.0" encoding="UTF-8"?>
<!–MXML version 1.0–>
<!–This is a process enactment event log created by OrchInsider to be analysed by ProM.–>
<!–ProM is the process mining framework. It can be freely obtained at http://www.processmining.org/.–>
<WorkflowLog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://is.tm.tue.nl/research/processmining/WorkflowLog.xsd"
description="OrchInsider log">
  <Source program="OrchInsider" />
  <Process id="EApp1Orchestrations.EApp1Process" description="EApp1Orchestrations.EApp1Process">
    <ProcessInstance id="2f4157c0-0be0-49e6-a7be-c43d74e05b50" description="Instructions number: 10">
      <AuditTrailEntry>
        <WorkflowModelElement>Initialization</WorkflowModelElement>
        <EventType>start</EventType>
        <Timestamp>2009-10-28T19:40:33.730</Timestamp>
        <Originator>Eclipse</Originator>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>ReceiveRequest</WorkflowModelElement>
        <EventType>start</EventType>
        <Timestamp>2009-10-28T19:40:33.730</Timestamp>
        <Originator>Eclipse</Originator>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>ReceiveRequest</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2009-10-28T19:40:33.730</Timestamp>
        <Originator>Eclipse</Originator>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>CheckQuantity</WorkflowModelElement>
        <EventType>start</EventType>
        <Timestamp>2009-10-28T19:40:33.730</Timestamp>
        <Originator>Eclipse</Originator>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>ConstructRequestDenied</WorkflowModelElement>
        <EventType>start</EventType>
        <Timestamp>2009-10-28T19:40:33.730</Timestamp>
        <Originator>Eclipse</Originator>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>ConstructRequestDenied</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2009-10-28T19:40:34.43</Timestamp>
        <Originator>Eclipse</Originator>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>SendReqDenied</WorkflowModelElement>
        <EventType>start</EventType>
        <Timestamp>2009-10-28T19:40:34.43</Timestamp>
        <Originator>Eclipse</Originator>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>SendReqDenied</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2009-10-28T19:40:34.90</Timestamp>
        <Originator>Eclipse</Originator>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>CheckQuantity</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2009-10-28T19:40:34.90</Timestamp>
        <Originator>Eclipse</Originator>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>Initialization</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2009-10-28T19:40:34.90</Timestamp>
        <Originator>Eclipse</Originator>
      </AuditTrailEntry>
    </ProcessInstance>
  </Process>
</WorkflowLog>
```

Figure 4.6: Log aggregation's generated MXML log file

4.4.2 Process Model

Although creating the MXML log file is very important, because it allows for pre-processed event log file storage and further analysis in distinct process mining platforms, the main goal of the log aggregation phase is to extract the design model of the mined orchestration's control-flow perspective. This is accomplished by mining every orchestration's instance's activity ordering information retrieved from the pre-processed event log, and building an activities' dependency frequency table that represents the ordering of tasks, the process' logical flow.

Table 4.3 shows the dependency frequency table discovered from the deployed orchestration presented in Section 2.3, where each row represents an edge or transition, connecting one activity to the other[7]. One interesting thing to notice is that the event type information is also present.

Activity Name A	Event Type A	Activity Name B	Event Type B	Frequency
Initialization	Start	ReceiveRequest	Start	3
ReceiveRequest	Start	ReceiveRequest	Complete	3
ReceiveRequest	Complete	CheckQuantity	Start	3
CheckQuantity	Start	SendReqToERP	Start	2
CheckQuantity	Start	ConstructRequestDenied	Start	1
SendReqToERP	Start	SendReqToERP	Complete	2
SendReqToERP	Complete	CheckQuantity	Complete	2
CheckQuantity	Complete	Initialization	Complete	3
ConstructRequestDenied	Start	ConstructRequestDenied	Complete	1
ConstructRequestDenied	Complete	SendReqDenied	Start	1
SendReqDenied	Start	SendReqDenied	Complete	1
SendReqDenied	Complete	Intitialization	Complete	1

Table 4.3: Log aggregation's dependency frequency table

Such knowledge is very important, not only it allows for concurrency or parallelism detection, when one activity is executed before another one ends; this is the case of the "CheckQuantity" and "SendReqToERP" activities; "SendReqToERP" is executed after "CheckQuantity" starts and before the latter ends; but also because with start and complete event types it is also possible to gather each activity's execution performance data. Finally, there is a frequency column that gives the cardinality of each transition, namely the number of times an edge was traversed. The former transition, connecting activities "CheckQuantity" and "SendReqToERP", has a frequency value of two, meaning that after mining all executed instances from the deployed orchestration, this transition was fired or triggered two times.

The dependency frequency table has relevant information to describe and model the mined orchestration's control-flow perspective, and is the starting point for some of the other phases involving the analysis and discovery of service orchestrations, such as model visualization, performance analysis and conformance analysis.

4.5 ProM Analysis

Using the MXML log file created in the former log aggregation phase, ProM can be used to apply process mining techniques to the orchestration's event log. However, as mentioned before in

[7]Activity A is executed and followed by activity B.

Sections 3.4 and 4.4.1, it will not be possible to answer some of the questions concerning service orchestrations, namely:

2. How was the service orchestration built? How many versions does it have? Can one see its growth along time?

6. How does the service orchestration's run-time behaviour conform with the previously designed model?

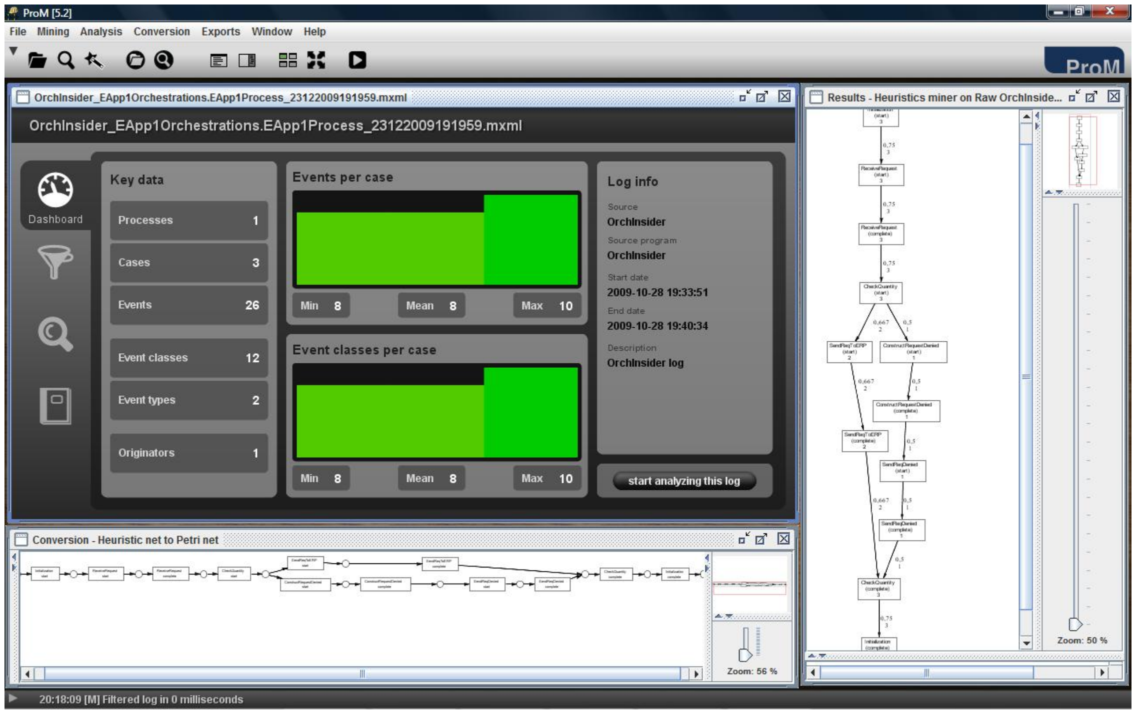
7. What are the external service communications?

8. What is the service architecture in the organization?

The MXML standard format does not accommodate more specific knowledge about service orchestrations, like:

- Activity type.
- Orchestration version.
- Communication ports.

Even if this information was present in the MXML log file, either in the *Data* element or in a proposed augment of the format, ProM's mining techniques would not be able to deal with such knowledge. Nevertheless, the orchestration's event log placed in the MXML log file is seen by the ProM application as a process, and, for that matter, ProM can be used to analyse service orchestrations, from every perspective. Figure 4.7 shows ProM's control-flow analysis of the deployed orchestration presented in Section 2.3. The discovery heuristic miner plugin was utilized and the orchestration's process model is presented. Also, a conversion from the heuristic net to a Petri net is shown, using a conversion plugin.



OrchInsider_EApp1Orchestrations.EApp1Process_23122009191959.mxml

Key data	
Processes	1
Cases	3
Events	26
Event classes	12
Event types	2
Originators	1

Events per case: Min 8 Mean 8 Max 10

Event classes per case: Min 8 Mean 8 Max 10

Log info
Source: OrchInsider
Source program: OrchInsider
Start date: 2009-10-28 19:33:51
End date: 2009-10-28 19:40:34
Description: OrchInsider log

20:18:09 [M] Filtered log in 0 milliseconds

Figure 4.7: ProM's control-flow analysis of a service orchestration's MXML log file

Although some of the advantages of the model visualization phase introduced in the next section are not present using the ProM framework, due to the specificity of service orchestrations, ProM can still be used to discover the process model and other performance issues.

4.6 Model Visualization

The next phase in the proposed ADSO mining methodology aims to visualize and interpret results, namely the complete service orchestration's control-flow. Model visualization presents process mining results so it is possible to gain clear insight of the process model, thus helping to answer questions:

1. How is the service orchestration modelled? What are the business rules? Are they coherent and executed correctly?

2. How was the service orchestration developed? How many versions does it have? Can one see its growth along time?

To fulfil these needs, the mined orchestration's model is visualized by means of a dependency graph. With the dependency frequency table from the log aggregation phase, this task can easily be accomplished, since every activity and edge is present.

Figure 4.8 shows the orchestration's dependency graph built using the previous dependency frequency table (cf. Table 4.3). Each edge's frequency is displayed and non-concurrent[8] activities are merged into one single activity. For instance, the "ReceiveRequest" start event activity is immediately followed by the "ReceiveRequest" complete event activity. In such cases, for a better understanding of the model, improving model visualization, these activities are merged into a single activity and the distinction between start and complete event types is no longer present. As one can easily observe, "ReceiveRequest", "SendReqToERP", "ConstructRequestDenied" and "SendReqDenied" become single activities whereas the "CheckQuantity" activity remains with start and complete event types distinction.

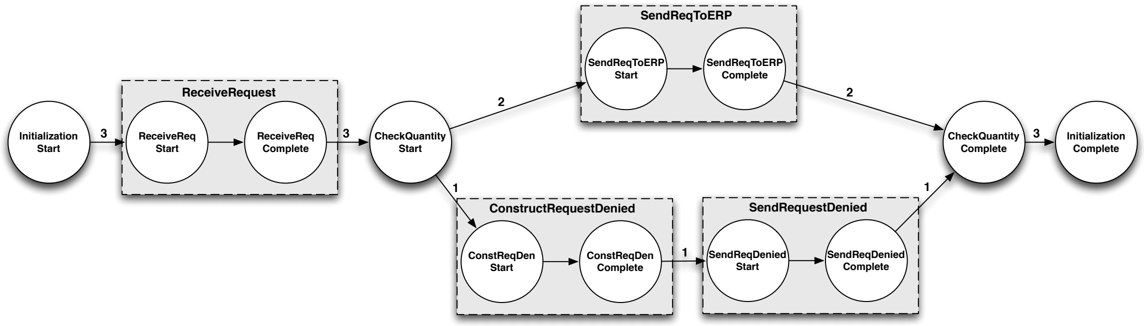


Figure 4.8: Model dependency graph

Although the dependency graph models the orchestration's control-flow, it is also important to enrich such model visualization with other orchestration-oriented information, in order to provide a more complete view of the orchestration's business logic.

[8]Activities that start and complete their execution without any other activity being started in between.

As mentioned in the log extraction phase, activity type and version information are very useful when designing the process model, as they enable improved design model understandability. Table 4.4 shows this information, retrieved from every activity present in the log aggregation's dependency frequency table from Section 4.4.2. One thing to notice, however, is that, in this case, all activities belong to the same orchestration version, since there is only one version available.

Activity Name	Activity Type	Frequency	Scope	Version/s
Initialization	Orchestration	3	-	1
ReceiveRequest	ReceiveShape	3	-	1
CheckQuantity	DecisionShape	3	-	1
SendReqToERP	SendShape	2	-	1
ConstructRequestDenied	ConstructShape	1	-	1
SendReqDenied	SendShape	1	-	1

Table 4.4: Orchestration-oriented activity information

1. **Activity type**

   (a) Identifying specific BPEL standard activities is helpful when designing the process model. BPEL activities like the <if> element, which can be found in Table 4.4 as the *Decision-Shape* type activity, whenever present, can be drawn in a similar BPEL standard fashion enabling better comprehension of the model.

   (b) One relevant activity type is the scope type activity. When present, not only it helps assuring activity naming consistency, as mentioned before, but is also a powerful tool that can help to improve model visualization and comprehension. Since scopes are a collection of activities, each scope can be seen as a macro activity. Identifying each activity's scope and placing scope information in the design model, can also enhance model readability.

2. **Version information** The development of the orchestration itself can be analysed using version information. Each activity can belong to one or several versions, so it is of utmost importance to keep track of this information. When the orchestration's model is designed and developed during the course of time, changes can be made and made over, some activities may even disappear and new ones introduced, so, with this information at hand, it will be possible to visualize and compare different process model versions, and understand how the orchestration was built along time.

Figure 4.9 presents the model visualization phase enriched with orchestration-oriented information. Since the orchestration modelled is the one from Section 2.3, it is easy to find some similarities when comparing Figure 2.5 with Figure 4.9. Activities are pictured as rounded boxes and business decision rules as diamond boxes. Analysing Table 4.4, one can see that the "CheckQuantity" activity is a *DecisionShape* type activity, and therefore a decision rule, or an <if> BPEL activity. This model depicts every behaviour "learned" from the event logs shown in Tables 3.1 and 4.2.

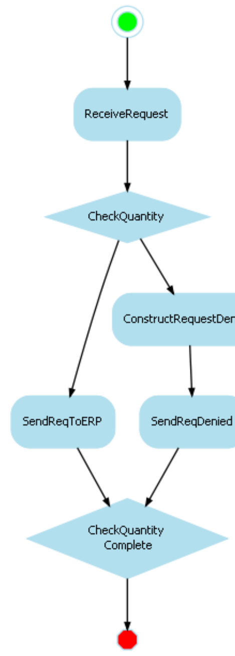


Figure 4.9: Model visualization phase with orchestration-oriented information

The visualization phase presents not only the orchestration's design model, as an activity dependency graph, but can also be enriched with more specific orchestration-oriented information, like standard BPEL activity design, scope "macro" activities, version analysis, and other information regarding performance analysis, such as activity and edge frequency.

4.7 Performance Analysis

Model visualization gives an overall idea and a clear insight of the executed orchestration's design model, the orchestration's control-flow, but this knowledge must be enhanced with orthogonal process mining results, like performance analysis. Such results may even enrich the orchestration's design model, allowing for easier and swift understandability of the process' performance, but they are also important as they help to seize control over the executed process, delivering answers to the following questions:

3. What are the most frequent paths and their execution probabilities along the service orchestration?

4. What is the average/maximum/minimum performance time for each activity and for the whole orchestration? And for external services, whenever present?

5. Is the process model efficient and adequate for its purpose? Are there any bottlenecks? What are the critical paths?

These are important questions that must be answered, and performance analysis attains all the required information to grasp this knowledge.

As mentioned in the log extraction phase, instance and activity timestamps are retrieved from the history log, and afterwards mapped to a standard format in the log inspection phase. Having each activity's timestamps from every orchestration's instance, and event type information, start and complete's, one can calculate each activity's maximum, minimum, and average execution times, as presented in Table 4.5. Also, activity frequency is retrieved, indicating the number of times each activity was performed.

Moreover, one can observe that activity execution times are quite fast, in the order of milliseconds, as mentioned before in Section 3.1. This relates to the fact that in service orchestrations, activities are mainly process logic execution and service invocation activities. However, when external services are invoked within the orchestration, activities that relate to service invocation may have higher execution times, since they are waiting for an answer from the external service.

Activity Name	Activity Type	Frequency	Max. Time	Min. Time	Avg. Time
Initialization	Orchestration	3	00:00:00.3600000	00:00:00	00:00:00.1800000
ReceiveRequest	ReceiveShape	3	00:00:00.2160000	00:00:00	00:00:00.1080000
CheckQuantity	DecisionShape	3	00:00:00.3600000	00:00:00	00:00:00.1800000
SendReqToERP	SendShape	2	00:00:00.0300000	00:00:00	00:00:00.0150000
ConstructRequestDenied	ConstructShape	1	00:00:00.3130000	00:00:00.3130000	00:00:00.3130000
SendReqDenied	SendShape	1	00:00:00.0470000	00:00:00.0470000	00:00:00.0470000

Table 4.5: Service orchestration's activity performance information

With activity performance information, knowing each activity's execution time performance and the whole orchestration's, which is betokened by the "Initialization" activity, one can educe several performance indicators:

- **Internal activities performance indicators**, such as maximum, minimum and average internal execution times.
- **External services performance indicators**, such as maximum, minimum and average external execution times.
- **Bottleneck analysis**, whenever an activity performance indicator, internal or external, surpasses a given threshold.

Since orchestrations are recursive compositions of services, there is a strong possibility that some external services are invoked. When a service orchestration calls an external service and expects an answer[9], some of the orchestration's instance's execution time is consumed "outside" the orchestration. For that matter, the whole internal execution time may not agree with the sum of each activity's execution time, either maximum, minimum or average. This occurs when *ReceiveShape*[10] type activities are present. In such cases, since these activities are waiting to receive an answer from an external service, the orchestration's execution time is being added with external service execution time. However, there is one exception that must be taken into account, which is the first activity, other than the "initialization" activity, present in the orchestration. Such first activity is a *ReceiveShape* type activity that instantiates and triggers the orchestration's execution upon message receival. When this is the case, it is clearly not an external service answer, since no service was invoked. For that matter, no external service execution time is considered. So, to really apprehend

[9]Request-response communication ports.
[10]Equivalent to a <receive> BPEL activity.

performance indicators, it is important to grasp, not only the orchestration's internal performance indicators, but also the orchestration's external performance indicators.

Other than time performance, edge and activity frequency information is also of use. Activity frequency values presented in Table 4.5, and edge frequency values displayed in Table 4.3, in conjunction with the internal activities performance indicators, support:

- **Critical paths** discovery, when high frequency paths containing bottleneck activities occur.
- **Path analysis**, concerning:
  - Path frequency, either path cardinality or probabilistic analysis.
  - Start and end activities, which determine every first executed activity from the orchestration, and every activity that is lastly executed. Considering again Microsoft Biztalk integration platform, it is enough to simply retrieve every edge that contains the "Initialization" activity, since it clearly indicates the starting and ending of each orchestration's instance. Table 4.6 presents start and end activities attained from the dependency frequency table (cf. Table 4.3). This analysis helps to determine the status of orchestration instances, namely completed or terminated instances. When an orchestration instance ends with a non-predicted end activity, this means that such instance was terminated and did not complete its entire flow.

Start Activities	Frequency	End Activities	Frequency
ReceiveRequest	3	CheckQuantity	3

Table 4.6: Service orchestration's start and end activities

  - Paths number. Path evaluation calculates the number of paths between any two given activities. This is achieved through a breadth-first search (BFS)[11] with loop detection, which explores the dependency graph (cf. Figure 4.8).

With performance analysis it is possible to unveil "hidden" knowledge about deployed orchestrations and their run-time behaviours, namely performance issues. This information may be used to enrich the design model visualization, and achieve higher control over the orchestration's process, really understanding how, when and why the orchestration is executed the way it is.

4.8 Conformance analysis

Having discovered the service orchestration's execution model and other performance issues, it is now possible to compare such retrieved model with the initially designed and modelled process flow, answering question:

6. How does the service orchestration's run-time behaviour conform with the previously designed model?

[11] In graph theory, a breadth-first search (BFS) is a graph search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those neighboring nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal.

Conformance checking, also referred to as conformance analysis, aims to detect and quantify any inconsistencies between the orchestration's projected design model and the learned run-time behaviour of the deployed orchestration. One important prerequisite for conformance analysis is that every activity in the design model must be associated, or mapped, with the equivalent activity, if it exists, in the event log. There must be a simple one to one mapping, where each process model activity is associated with exactly only one event log activity, meaning that no other activity in the model can match the same activity in the event log. When mapping equivalent activities, the following concerns may occur:

1. **Duplicate activities** Activities may appear more than once in the model. Even when name consistency is granted (cf. Section 4.3), "duplicate" activities are present. As one can observe, in the model dependency graph from Figure 4.8, there are two activities with the same name, "CheckQuantity". The only information that can be used to differentiate them is their corresponding event types. For that reason, activity mapping must include event type information.

2. **Invisible activities** Some activities were simply not executed, and therefore are not present in the event log. Hence, certain steps in the orchestration's process model may not be observable.

3. **Orchestration model version** Activities may be removed, changed from position, or renamed from time to time. It is crucial to grant that orchestration versioning is correctly performed, so event logs and their activities are mapped correctly with their correspondent orchestration version.

Considering once more Microsoft Biztalk integration platform, each designed orchestration creates an ".odx" file that contains all the orchestration's operations, artefacts, and workflow logic. With such *odx* file, it is possible to parse the entire designed process model (cf. Section 4.4.2), although in this case, instance aggregation is not required. The dependency frequency table is established, even if every edge has a frequency value of one. One important aspect when parsing the orchestration's model from the *odx* file, is the name consistency issue, which relates with activity mapping. Activity name consistency must be granted before activity mapping, hence both models, either mined or parsed models[12], must be built using the same activity labelling option, as discussed in the log inspection phase. Afterwards, the activity information table (cf. Table 4.4) is created, so it is possible to visualize and differentiate both models, the mined and the parsed model.

Conformance can be analysed from two different ways:

- Instance conformance: measures fitness between event logs and parsed models.

- Model conformance: measures and quantifies how much behaviour allowed by the designed or parsed model is never used by the mined model, thus behavioural appropriateness.

[12]The initially designed and modelled process is referred to as parsed model.

4.8.1 Instance Conformance

The idea beneath instance conformance, or fitness measurement, is to replay every edge from every instance's event log in the parsed model, and measure mismatches, and correctly triggered transitions. The following instance conformance metric is adapted from the fitness metric introduced in (Rozinat and van der Aalst, 2008).

Fitness Let $\#E$ be the number of edges present in each instance. Let $\#C$ be the number of correctly triggered transitions. Let $\#M$ be the number of mismatches. Let $\#I$ be the number of instances. For each instance ($1 \leq i \leq \#I$), the instance fitness is equal to $\frac{\#C}{\#E}$ and $\#M$ is equal to $\#E$ - $\#C$. So, the event log fitness metric f is defined as follows:

$$f = \frac{1}{\#I} \sum_{i=1}^{\#I} \frac{\#C}{\#E}$$

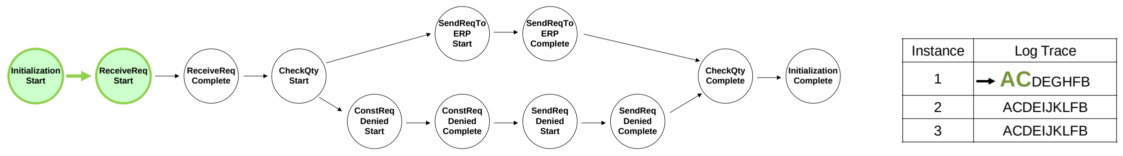
The replay of each instance's event log starts with the first transition, or edge, and afterwards, all transitions belonging to the event log are triggered one after another. During the replay, correctly triggered edges are counted until there are no more transitions in the event log.

To illustrate this rationale, Figure 4.10 depicts all transitions triggered during event log replay using instance number one from Table 3.1 and the previous dependency graph (cf. Figure 4.8). For easier understandability of the process, all activities, including their corresponding event types, were mapped to letters A, B, C and so forth, as shown in Table 4.7.

Activity Name	Event Type	Label
Initialization	start	A
Initialization	complete	B
ReceiveRequest	start	C
ReceiveRequest	complete	D
CheckQuantity	start	E
CheckQuantity	complete	F
SendReqToERP	start	G
SendReqToER	complete	H
ConstructRequestDenied	start	I
ConstructRequestDenied	complete	J
SendRequestDenied	start	K
SendRequestDenied	complete	L

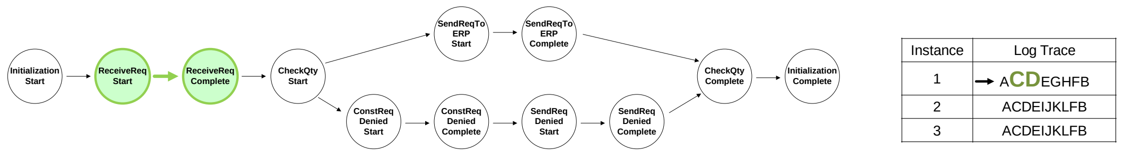
Table 4.7: Activity mapping using event type information

All transitions were triggered correctly, as one should expect, yet, in such cases as orchestrations that did not complete their entire flow, either by an error, or because they were terminated by the user, the fitness metric presents itself as a useful detection tool.



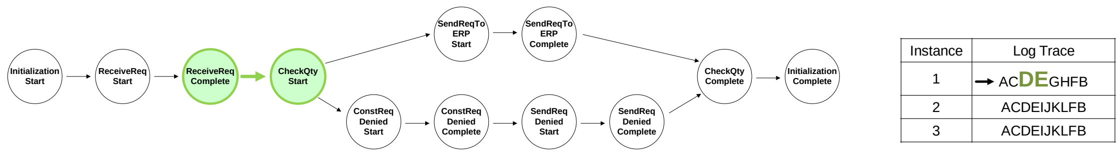
Instance	Log Trace
1	→ ACDEGHFB
2	ACDEIJKLFB
3	ACDEIJKLFB

(a) Initialization (Start) > ReceiveRequest (Start)



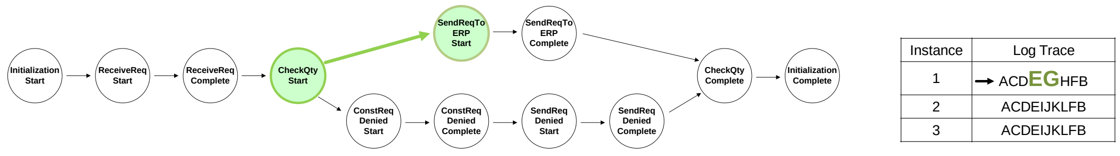
Instance	Log Trace
1	→ ACDEGHFB
2	ACDEIJKLFB
3	ACDEIJKLFB

(b) ReceiveRequest (Start) > ReceiveRequest (Complete)



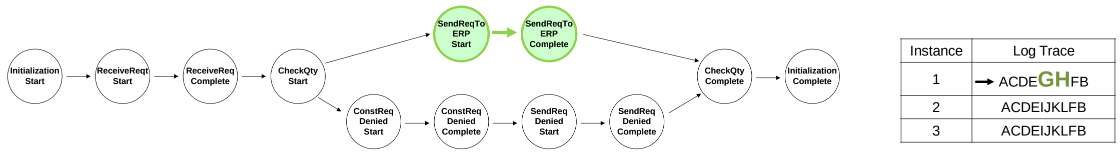
Instance	Log Trace
1	→ ACDEGHFB
2	ACDEIJKLFB
3	ACDEIJKLFB

(c) ReceiveRequest (Complete) > CheckQuantity (Start)



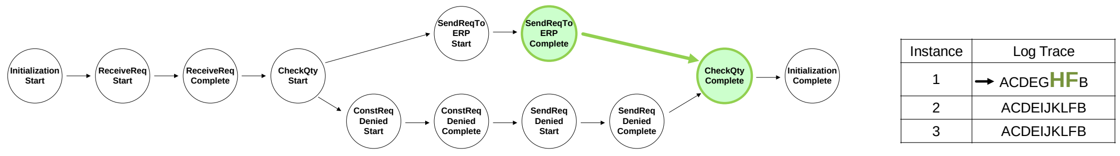
Instance	Log Trace
1	→ ACDEGHFB
2	ACDEIJKLFB
3	ACDEIJKLFB

(d) CheckQuantity (Start) > SendReqToERP (Start)



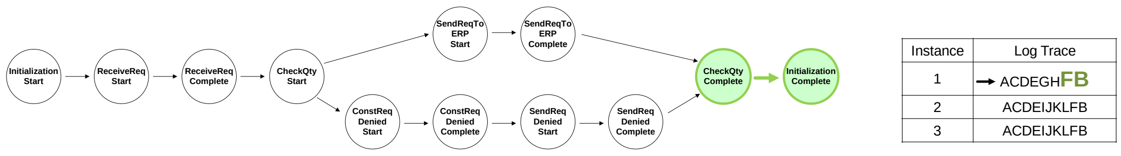
Instance	Log Trace
1	→ ACDEGHFB
2	ACDEIJKLFB
3	ACDEIJKLFB

(e) SendReqToERP (Start) > SendReqToERP (Complete)



Instance	Log Trace
1	→ ACDEGHFB
2	ACDEIJKLFB
3	ACDEIJKLFB

(f) SendReqToERP (Complete) > CheckQuantity (Complete)



Instance	Log Trace
1	→ ACDEGHFB
2	ACDEIJKLFB
3	ACDEIJKLFB

(g) CheckQuantity (Complete) > Initialization (Complete)

Figure 4.10: Log replay during instance conformance checking

4.8.2 Model Conformance

Instance conformance does not use aggregated event logs. Instead, it works with every orchestration's instance independently and calculates the overall fitness metric. On the other hand, model conformance works with both orchestration models, the mined process model and the parsed model (cf. Section 4.4.2). Model conformance is interested in the complete workflow, or orchestration's behaviour, and tries to quantify the behavioural appropriateness, how much of the behaviour present in the parsed orchestration's model is availed by the mined orchestration's model. The procedure behind model conformance is to traverse every edge from the parsed orchestration's model, and replay it in the mined orchestration's model, measuring correctly triggered transitions and mismatches. The following model conformance metric is adapted from the behavioural appropriateness metric introduced in (Rozinat and van der Aalst, 2008).

Behavioural appropriateness Let $\#E$ be the number of edges present in the parsed model. Let $\#C$ be the number of correctly traversed edges. Let $\#B$ be the number of edges learned in the mined model. For each edge $(1 \leq i \leq \#B)$, the behavioural metric B_i is either one, if the edge is present in the parsed model, or zero, otherwise. The model behavioural appropriateness metric b is defined by the following equation:

$$b = \frac{1}{\#E} \sum_{i=1}^{\#E} B_i$$

Even when instances are fully compliant, the whole process model may not be. One should always design a process as precisely as possible, but, on the other hand, transitions that do not occur in the mined process model may be due to log incompleteness (cf. Section 3.1). Such behaviour may even occur in the future, but is not currently recorded in the event log. Furthermore, the designed model may be too generic and this way allowing for unwanted and unrealistic behaviour. It is always up to the developer to analyse and differentiate such situations, in order to accomplish a compromise between both realities.

Although conformance analysis was presented as being one of the last phases in the ADSO mining methodology, it is an important fact that after the log aggregation phase, conformance analysis can be useful to filter unwanted orchestration instances from evaluation. For instance, if conformance checking is executed before the design model phase, some of the terminated and problematic instances can be identified and, if so desires, the developer or the analyst can select only complete instances to mine the orchestration's design model. Also, conformance checking allows for an overall starting point which enables improved performance analysis, since a global view of the mined orchestration is presented, and mining decisions can be made with some previous knowledge.

4.9 Service Network Architecture

Service orchestrations are recursive composition of services, and enable service and process collaboration. For this reason, it is important to understand which external services are invoked during orchestration execution. The service network architecture phase aims to analyse the orchestration's communication ports, and answer questions:

7. What are the external service communications?
8. What is the service architecture in the organization?

Communication ports are important aspects of orchestrations, as they enable service re-usage and therefore SOA-based services. After the log aggregation phase, each port name, direction and timestamp information retrieved from every orchestration's instance gives enough information to characterize communications. Port direction information, namely send and receive, is useful to detect port types, such as:

- Request-only port types, when ports are only invoked one way, either sending or receiving messages.
- Request-response port types, when ports are invoked both ways, sending and receiving messages.

With this information at hand, it is possible to define the service network architecture, depicting service interoperability. Even so, this network would be centralized in the selected orchestration, since there is only communication information available from such orchestration. A more interesting approach would be to design a distributed orchestration network (cf. Section 2.4), a choreography. It would lead to a new mining perspective (cf. Section 3.2), an interoperability or choreography perspective. This brings forth two new issues;

- How to gather every orchestration's log in a joint event log.
- How to link every instance with every communication message.

The first issue could be solved using some kind of a log integration platform, but even so, every orchestration's log must be aggregated in such platform. The second issue remains as an investigation opportunity. In (Aalst and Verbeek, 2008) this challenge is somewhat introduced, but with the purpose of correlating events with process instances. The presented ADSO methodology solves this problem and brings forward this other similar problem, how to link messages with execution instances. One idea would be to use timestamps, but this brings another problem, time synchronism between execution engines. All execution engines should be synchronized with exactly the same time, or use some kind of a time server. However, correlation seems to be the best way to address this problem.

4.10 Conclusion

A service orchestrations' mining methodology, ADSO, was introduced in this chapter, with all the main steps involving the analysis and discovery of service orchestrations discussed and scrutinized. An orchestration developed with Microsoft Biztalk integration platform was used as an example to substantiate the proposed methodology. In Chapter 6, the ADSO methodology will be validated and tested with more complex service orchestrations. The proposed service orchestrations' mining methodology has delivered some relevant information that has brought into the light some other aspects of orchestration analysis, namely service communications delivering distributed services, or choreographies. These are unsolved issues and present themselves as research opportunities. In the following chapter, OrchInsider, a service orchestrations' mining application that applies the presented solution in the same integration platform, will be introduced.

Chapter 5

OrchInsider: a Service Orchestrations' Mining Application

A software application, henceforth referred to as OrchInsider, was designed to implement the analysis and discovery of service orchestrations (ADSO) mining methodology[1], based on Microsoft Biztalk integration server. Although it was initially conceived to support Biztalk, OrchInsider can easily be adapted to support other integration platforms, only requiring the capability of extracting event logs from other systems. This can be accomplished by adding a software "plugin", or by using the MXML data log format depicted in Figure 3.2. Also, OrchInsider can work together with ProM, since it can export pre-processed event log files to MXML. Nevertheless, some of the analyses performed by OrchInsider are not yet provided by the ProM framework (cf. Sections 4.4.1 and 4.5), such as version analysis, communication analysis or even conformance[2] analysis.

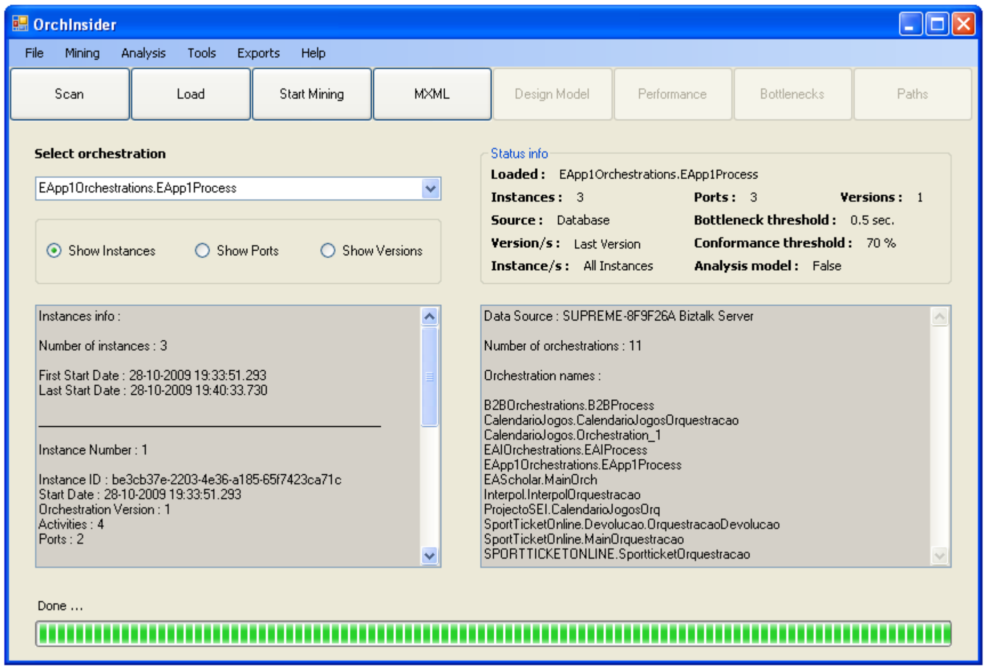
OrchInsider has been developed to analyse and discover the following aspects of deployed service orchestrations:

- Version Analysis.
- Communication Ports.
- Design Model.
- Performance Analysis.
- Bottleneck Analysis.
- Path Analysis.
- Conformance Analysis.

OrchInsider is a context-based application, where every information and command can only be displayed and executed according to the current step in the ADSO mining methodology[3]. This

[1]ADSO - Analysis and Discovery of Service Orchestrations mining methodology, was introduced in Chapter 4.
[2]ProM's conformance plugins can only be applied if parsed service orchestration models are exported.
[3]For instance, design model can only be attained after log aggregation.

is illustrated in Figure 5.1, where instance information from the deployed orchestration presented in Section 2.3 is displayed. Also, it is useful and possible to observe the orchestration's first and



Select orchestration

EApp1Orchestrations.EApp1Process

(● Show Instances) ○ Show Ports ○ Show Versions

Status info

Loaded : EApp1Orchestrations.EApp1Process
Instances : 3 **Ports :** 3 **Versions :** 1
Source : Database **Bottleneck threshold :** 0.5 sec.
Version/s : Last Version **Conformance threshold :** 70 %
Instance/s : All Instances **Analysis model :** False

```
Instances info :

Number of instances : 3

First Start Date : 28-10-2009 19:33:51.293
Last Start Date : 28-10-2009 19:40:33.730

_______________________________________________

Instance Number : 1

Instance ID : be3cb37e-2203-4e36-a185-65f7423ca71c
Start Date : 28-10-2009 19:33:51.293
Orchestration Version : 1
Activities : 4
Ports : 2
```

```
Data Source : SUPREME-8F9F26A Biztalk Server

Number of orchestrations : 11

Orchestration names :

B2BOrchestrations.B2BProcess
CalendarioJogos.CalendarioJogosOrquestracao
CalendarioJogos.Orchestration_1
EAIOrchestrations.EAIProcess
EApp1Orchestrations.EApp1Process
EAScholar.MainOrch
Interpol.InterpolOrquestracao
ProjectoSEI.CalendarioJogosOrq
SportTicketOnline.Devolucao.OrquestracaoDevolucao
SportTicketOnline.MainOrquestracao
SPORTTICKETONLINE.SportticketOrquestracao
```

Done ...

Figure 5.1: Deployed orchestrations and instance information, using OrchInsider

last execution dates, and versions number. When mining service orchestrations, this information comes at hand, since it provides the analyst with appropriate knowledge regarding the usage of filters, improving analysis control. Several options can be selected using OrchInsider. For instance, one could define a specific list of instances to analyse, or select instances between start and end dates, and even target results by specifying which version/s to analyse. The status info displayed in Figure 5.1 shows a summary of relevant information about the loaded orchestration, along with some of the current application options. The following sections will address the utility of these and other options combined with the presented application features with the purpose of seizing control and monitor orchestration execution.

5.1 Versioning

Version analysis is very useful due to the nature of orchestrations (cf. Sections 3.2 and 4.6). Using version analysis, it is possible to observe how the orchestration's logical model was developed and evolved during the course of time, which decisions were made and made over, and use this information to gather further knowledge about the life cycle of service orchestrations. It is also important because it allows building the orchestration's model accordingly, avoiding misinterpretation. If the model is changed during time, some paths or activities may become unused or even "disappear", so it is of utmost importance to deal with this problem. Version analysis can be the key. However, this kind of analysis is always user dependent, since it is the user's or developer's responsibility to control and manage versioning.

OrchInsider displays information about each orchestration's version, specifically every version's last deployment date, and allows for orchestration version analysis, since it can analyse every instance and correlate it with the corresponding version. Using this information, it is possible to design and recognize an orchestration model containing several different versions, therefore immediately observing model differences along time.

Figure 5.2 presents a small part of the version analysis performed using *SportTicket Online's* buying process orchestration[4]. As can be seen, it is possible to easily detect version activities.

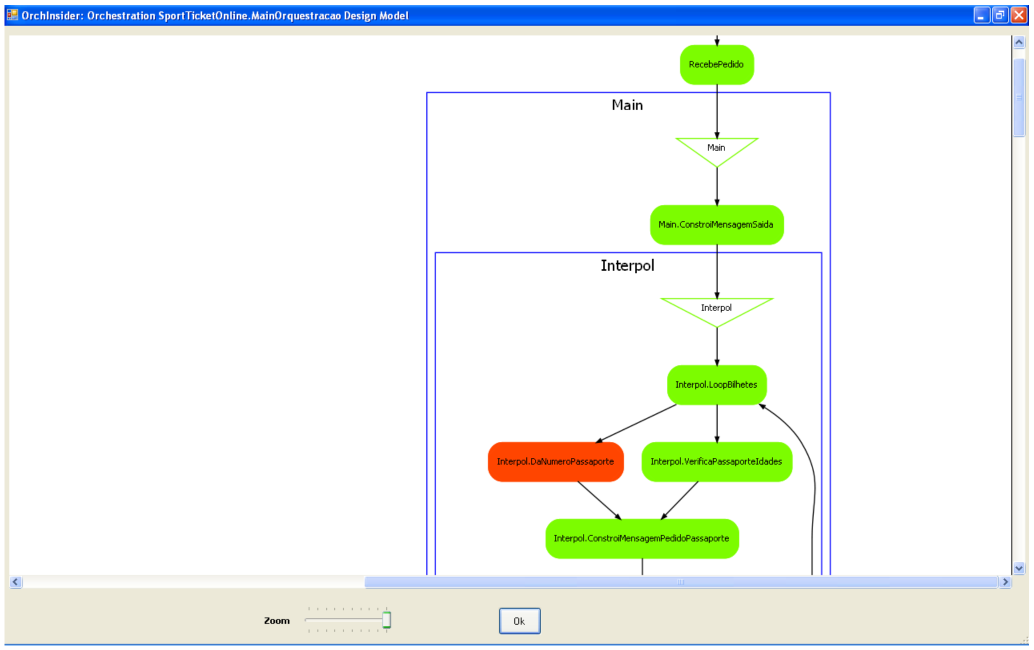

OrchInsider: Orchestration SportTicketOnline.MainOrquestracao Design Model
RecebePedido
Main
Main
Main.ConstroiMensagemSaida
Interpol
Interpol
Interpol.LoopBilhetes
Interpol.DaNumeroPassaporte
Interpol.VerificaPassaporteIdades
Interpol.ConstroiMensagemPedidoPassaporte
Zoom
Ok

Figure 5.2: Version differences analysis, using OrchInsider

Latter version activities are marked green whereas former version activities appear in red. In this case, one can observe that the activity "*DaNumeroPassaporte*" is somewhere in time discarded, and presumably replaced by the new activity "*VerificaPassaporteIdades*".

5.2 Communication Ports

OrchInsider delivers information about service communications, where each instance's ports are listed, along with;

- port name;
- port direction, send or receive; and
- port timestamps;

If a communication port is invoked several times during orchestration execution, such port will be listed with different timestamps. With port direction and timestamp information, it is possible to understand the orchestration's communication behaviour, which is displayed in Figure 5.3, and depict the service network architecture.

[4] *SportTicket Online's* buying process orchestration will be discussed in the case study presented in Chapter 6.

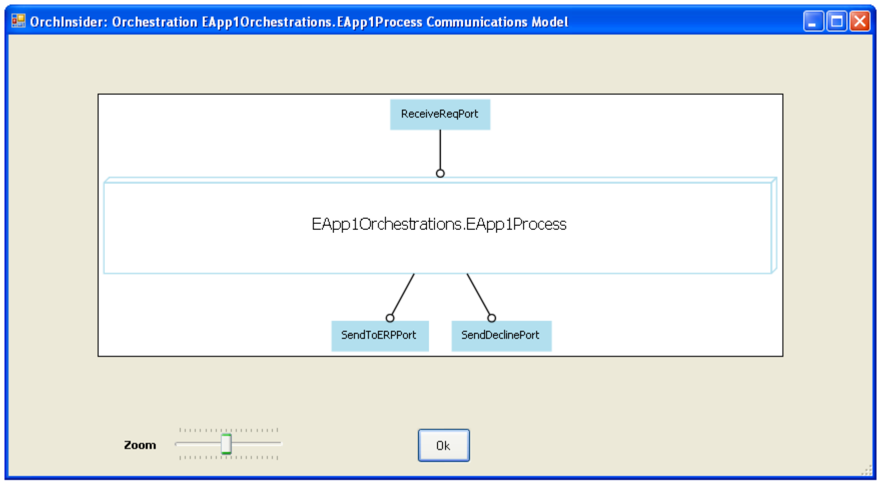

OrchInsider: Orchestration EApp1Orchestrations.EApp1Process Communications Model
ReceiveReqPort
EApp1Orchestrations.EApp1Process
SendToERPPort
SendDeclinePort
Zoom
Ok

Figure 5.3: Service orchestration's external communications, using OrchInsider

Figure 5.3 shows the external service communications from the orchestration introduced in Section 2.3, where it is possible to view each port with corresponding direction information[5].

5.3 Design Model

Figure 4.9 already presented the mined design model from the selected orchestration. However, as discussed in Sections 4.6 and 4.7, the process model can be enriched with other information concerning performance analysis, like path traversing frequency or probability. Critical paths and bottlenecks can also be depicted in a similar fashion. OrchInsider allows for model performance analysis visualization. It is possible to configure and observe in the design model several performance indexes and orchestration-oriented information, such as:

- Bottleneck activity visualization, or bottleneck probabilistic analysis, concerning activity execution time, average, maximum or minimum. Bottleneck threshold value can also be configured[6]. Bottleneck activity visualization highlights every activity whose execution time surpasses the configured threshold[7]. Bottleneck probabilistic check is somewhat different, since every activity is drawn with a gray tonality according to its execution time and related to the orchestration's execution time.

- Activity frequency probabilistic analysis, concerning the number of times each activity was executed. Once more, activities are drawn with a gray tonality according to their execution frequency, and related to the maximum activity execution frequency in the orchestration.

- Path frequency visualization, and path probabilistic analysis, concerning the number of times each edge was traversed. Path frequency visualization displays every edge's frequency number. Path frequency probabilistic analysis relates to high traversed edges, since each edge's width is correlated to the edge's frequency and the maximum edge frequency found in the orchestration.

[5]Port direction information is displayed using small non-filled dots that represent receive direction ports.
[6]Bottleneck threshold represents activity duration, either average, maximum or minimum, in seconds.
[7]Bottleneck activities appear in red.

If a bottleneck activity is present in a high frequency path, then the path may be considered as a critical path.

- Standard BPEL[8] activity design, where activities are drawn in a similar BPEL standard fashion.
- Scope "macro" activity visualization, where scope information is depicted in the design model. Every activity belonging to a specific scope is placed inside a blue square, representing the scope area. Scope areas can be labeled with proper name identification.

Figure 5.4 displays the mined design model, enhanced with performance information, for the orchestration presented in Section 2.3. The retrieved and revealed model includes bottleneck probabilistic

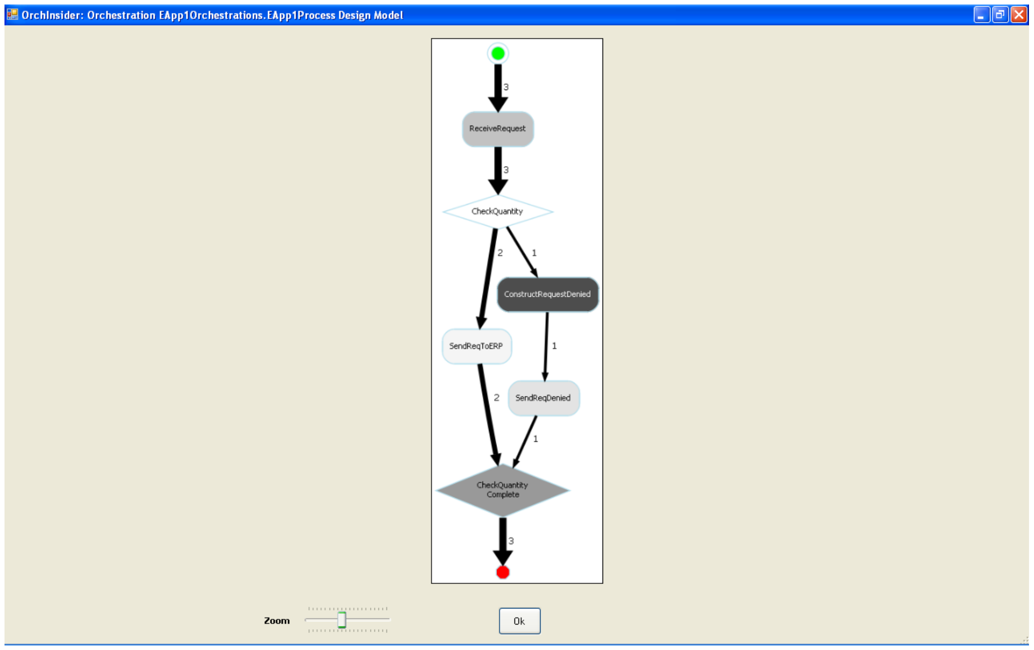


Figure 5.4: Design model enhanced with performance analysis, using OrchInsider

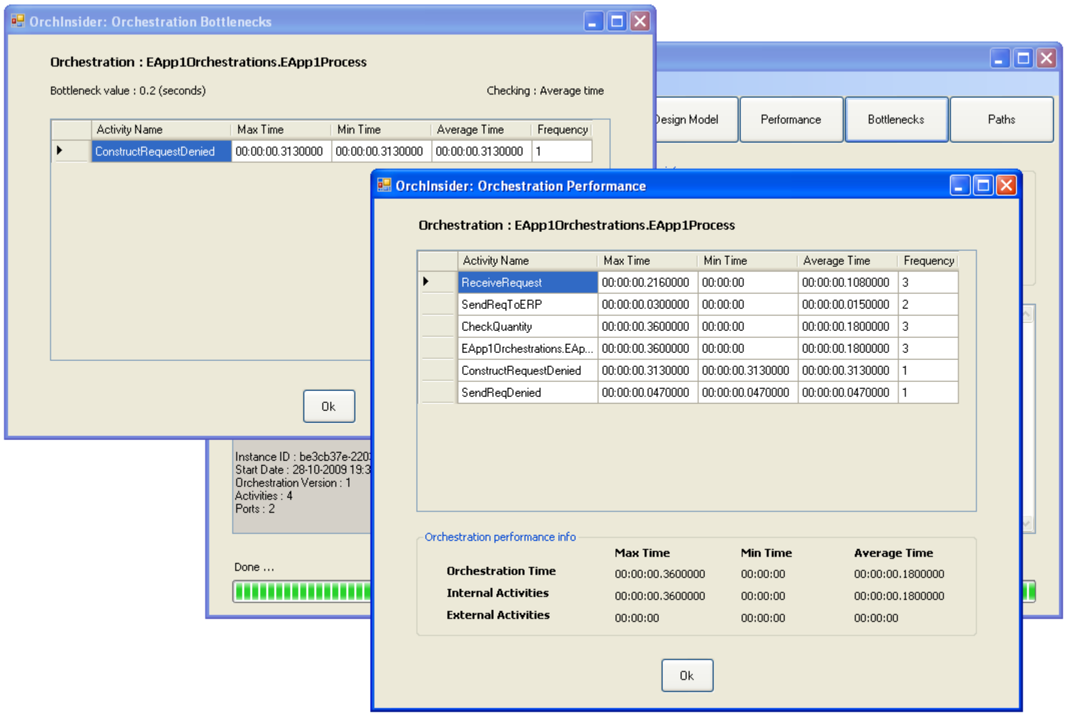
analysis, path frequency visualization, and path probabilistic analysis. As one can easily observe, a glance is almost enough to immediately perceive critical paths and bottlenecks, and most frequent paths.

5.4 Performance Analysis

Using OrchInsider's performance analysis features provides further knowledge about the mined process behaviour. However, to properly adjust performance features with run-time performance analysis, and due to the granularity of such features, a previous understanding of the modelled process should exist, so even better results can be accomplished. Considering again the selected orchestration from Section 2.3, which will be used as an example in the following sections, to properly analyse bottleneck activities and critical paths, such previous process awareness empowers the user to properly adjust bottleneck thresholds and correctly understand performance indicators.

[8]BPEL, namely WS-BPEL, was mentioned in Section 2.3.

OrchInsider displays activity performance indicators, orchestration performance information and allows for bottleneck configuration and analysis, as can be observed in Figure 5.5.


OrchInsider: Orchestration Bottlenecks
Design Model
Performance
Bottlenecks
Paths
Ok
Instance ID : be3cb37e-220
Start Date : 28-10-2009 19:3
Orchestration Version : 1
Activities : 4
Ports : 2
Done ...
OrchInsider: Orchestration Performance
Ok

Orchestration : EApp1Orchestrations.EApp1Process

Bottleneck value : 0.2 (seconds)

Checking : Average time

Activity Name	Max Time	Min Time	Average Time	Frequency
ConstructRequestDenied	00:00:00.3130000	00:00:00.3130000	00:00:00.3130000	1

Orchestration : EApp1Orchestrations.EApp1Process

Activity Name	Max Time	Min Time	Average Time	Frequency
ReceiveRequest	00:00:00.2160000	00:00:00	00:00:00.1080000	3
SendReqToERP	00:00:00.0300000	00:00:00	00:00:00.0150000	2
CheckQuantity	00:00:00.3600000	00:00:00	00:00:00.1800000	3
EApp1Orchestrations.EAp...	00:00:00.3600000	00:00:00	00:00:00.1800000	3
ConstructRequestDenied	00:00:00.3130000	00:00:00.3130000	00:00:00.3130000	1
SendReqDenied	00:00:00.0470000	00:00:00.0470000	00:00:00.0470000	1

Orchestration performance info

	Max Time	Min Time	Average Time
Orchestration Time	00:00:00.3600000	00:00:00	00:00:00.1800000
Internal Activities	00:00:00.3600000	00:00:00	00:00:00.1800000
External Activities	00:00:00	00:00:00	00:00:00

Figure 5.5: Performance analysis with OrchInsider

Analysing the presented results, one can conclude that:

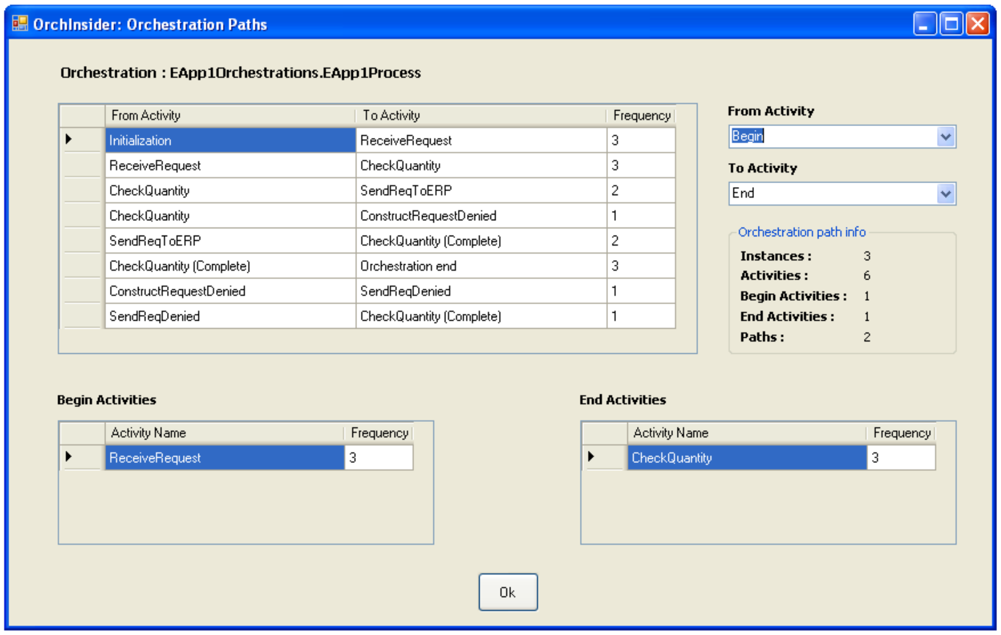
- No execution time was spent outside the orchestration, which can be confirmed with the previous communication analysis, since there are not any request-response communication ports.

- Choosing a bottleneck threshold of 0.2 seconds, which seems to a be a reasonable value considering Table 4.5, and selecting average time as the bottleneck analysis performance indicator (cf. Section 4.7), one can observe that there was one activity, “ConstructRequestDenied”, that surpassed the configured threshold value. This can also be confirmed by observing Figure 5.4, where bottleneck activity probabilistic analysis is displayed.

- Even recognizing that there are few orchestration instances presented, the path containing the “ConstructRequestDenied” activity may be considered a critical path. Figure 5.4 is also useful to reinforce this idea, and path analysis can be used to clarify this issue.

Performance analysis ensures a closer look at the orchestration’s run-time behaviour, allowing for model performance quantification and this way improving process comprehension.

Path analysis, is another feature that can help to clarify and better understand the orchestration’s performance. Using path analysis, it is possible for a developer or an analyst to measure and quantify the orchestration’s behaviour in more detail. Let’s imagine that, when the orchestration was initially developed, it was thought that the leading to the “ConstructRequestDenied” activity path would only be traversed a few times. It is interesting to analyse, after executing the orchestration several times, if this assumption remains true. Path analysis, as presented in Figure 5.6, can help to clarify observed behaviour.



Orchestration : EApp1Orchestrations.EApp1Process

From Activity	To Activity	Frequency
Initialization	ReceiveRequest	3
ReceiveRequest	CheckQuantity	3
CheckQuantity	SendReqToERP	2
CheckQuantity	ConstructRequestDenied	1
SendReqToERP	CheckQuantity (Complete)	2
CheckQuantity (Complete)	Orchestration end	3
ConstructRequestDenied	SendReqDenied	1
SendReqDenied	CheckQuantity (Complete)	1

From Activity: Begin

To Activity: End

Orchestration path info

Instances :	3
Activities :	6
Begin Activities :	1
End Activities :	1
Paths :	2

Begin Activities

Activity Name	Frequency
ReceiveRequest	3

End Activities

Activity Name	Frequency
CheckQuantity	3

Figure 5.6: Path analysis with OrchInsider

OrchInsider's path analysis delivers meaningful information, such as:

- Each edge's frequency, useful to quantify observed behaviour.
- Total number of instances and activities mined.
- Begin and end activities, which help to understand if terminated instances occurred, whenever non-predicted end activities are present (cf. Section 4.7).
- Number of possible paths between any two activities, which can be used to quantify and perceive if specified behaviours, travelled paths between activities, were availed by the mined process.

With the presented features, the developer, or analyst, can "zoom" in the knowledge about the orchestration's process. Observing Figure 5.6, it is possible to apprehend that there were no terminated instances, since each start and end activities are correctly predicted activities[9], and the total number of paths, from the beginning until the end, is two, as initially expected.

5.5 Conformance Analysis

OrchInsider provides useful features concerning the analysis and discovery of deployed orchestrations. First, relevant run-time information, amongst the vast amount of data stored in the event

[9]Activities "ReceiveRequest" and "CheckQuantity" are expected to be start and end activities, respectively, as can be seen in Figures 4.9 and 5.4.

logs, is extracted. Secondly, extracted data is analysed, and the process model along with other performance issues are discovered. Afterwards, the run-time behaviour of the deployed orchestration can be easily scrutinized and quantified. However, developers and analysts may like to go even further, and compare the initially designed and modelled process with the mined and retrieved model. Such conformance analysis is possible with OrchInsider, only requiring that the designed orchestration's ".odx" file is available. The initially modelled process can therefore be parsed from the *odx* file and the parsed model constructed (cf. Sections 4.4.2 and 4.8). Finally, both models, the mined and the parsed model[10], can be compared and inconsistencies detected and quantified. Figure 5.7 depicts the former rationale, which represents the full cycle of the analysis and discovery of service orchestrations (ADSO).

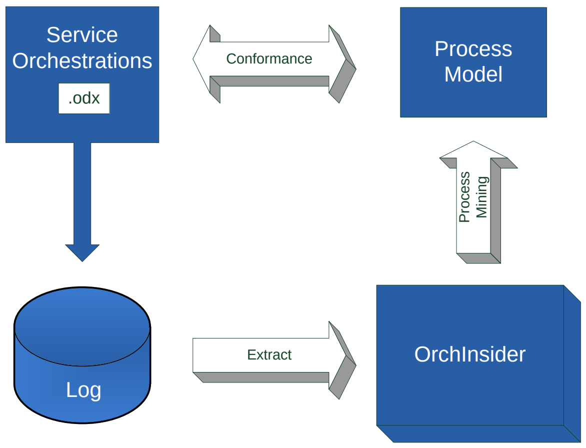


Figure 5.7: Analysis and discovery of service orchestrations (ADSO), the full cycle

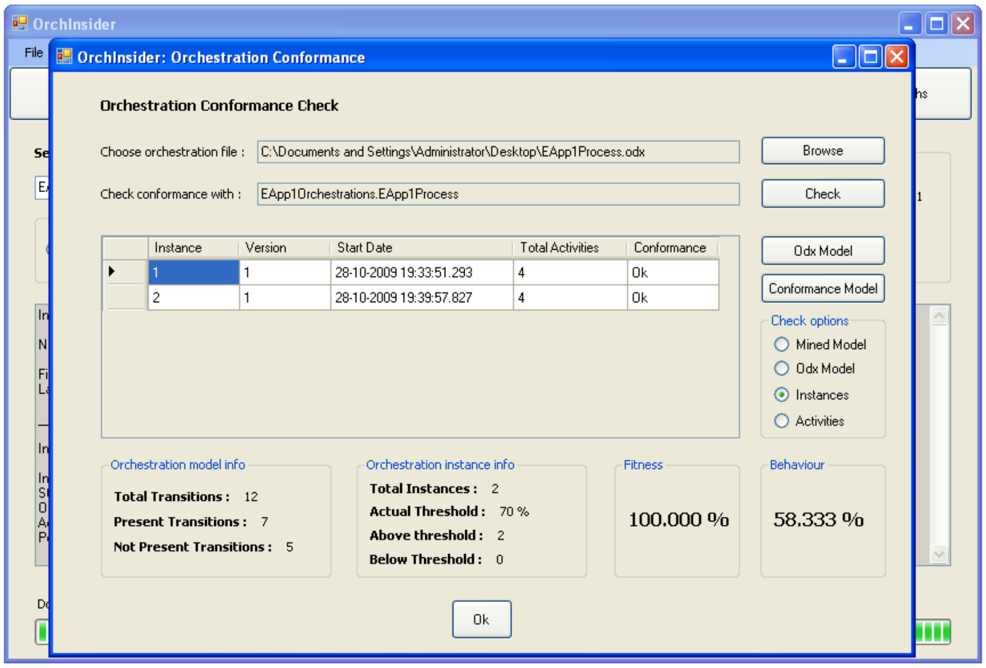
Using OrchInsider, conformance is analysed using two different metrics (cf. Section 4.8):

- Fitness, measuring instance conformance.
- Behaviour, measuring model conformance.

Observing Figure 5.8, where conformance check is made for the selected orchestration, one finds that both instances are above the configure fitness threshold, fitness conformance is 100% and behaviour conformance is around 58%. Since only two instances were chosen and mined, these results appear to be realistic and accurate. As one may see, both instances are *Ok*[11], whereas only seven transitions are present, from the total of twelve transitions available in the parsed model[12]. This means that some behaviour is not performed by the mined orchestration's instances, even if they both conform with the process model. To substantiate this understanding, it is possible to select some options that can improve conformance analysis, such as:

- Mined model: Lists every edge mined from the deployed orchestration, and information about whether each edge is present or not in the parsed model.

[10]The initially designed and modelled process is referred to as parsed model.
[11]*Ok* means 100% conformance.
[12]All transitions present in the dependency frequency table.



Orchestration Conformance Check

Choose orchestration file : C:\Documents and Settings\Administrator\Desktop\EApp1Process.odx

Check conformance with : EApp1Orchestrations.EApp1Process

Instance	Version	Start Date	Total Activities	Conformance
1	1	28-10-2009 19:33:51.293	4	Ok
2	1	28-10-2009 19:39:57.827	4	Ok

Check options: Mined Model / Odx Model / [✓ Instances] / Activities

Orchestration model info
Total Transitions : 12
Present Transitions : 7
Not Present Transitions : 5

Orchestration instance info
Total Instances : 2
Actual Threshold : 70 %
Above threshold : 2
Below Threshold : 0

Fitness
100.000 %

Behaviour
58.333 %

Figure 5.8: Conformance analysis with OrchInsider

- Parsed model: Similar to the previous information, except that now edges from the parsed model are listed, together with information about whether each edge is present or not in the mined model.
- Instances: As one can observe in Figure 5.8, this option displays every mined orchestration's instance, with version, start date, number of activities and fitness conformance information.
- Activities: Displays every activity retrieved from both models, and information about whether each activity is present or not in the mined and parsed models.

OrchInsider enables the possibility to visualize the parsed model (cf. Section 4.6), and also the conformance model. The conformance model, as presented in Figure 5.9, is used to compare mined and parsed behaviours, which is useful to get a clear insight of model conformance. This can be accomplished by using two different approaches:

1. Activity conformance: The purpose is to easily identify conform activities and non-conform activities. Conform activities belong to both models and are marked green, whilst mined-only[13] activities appear in red, and parsed-only[14] activities are marked blue. Mined-only activies, although uncommon and not expected, may occur if model conformance is checked with non-correspondent orchestration versions, or if the orchestration was changed and versioning was not performed accordingly (cf. Section 4.8). Parsed-only activities occur when certain behaviour is not executed by the mined model, and therefore represent activities that are not present in the event log.

[13]Activities that only belong to the mined model.
[14]Activities that only belong to the parsed model.

2. Model differences: Intends to easily detect differences between the parsed model and the mined model. Mined-only activities appear in red, whilst activities present in either the parsed model or in both models appear in blue.

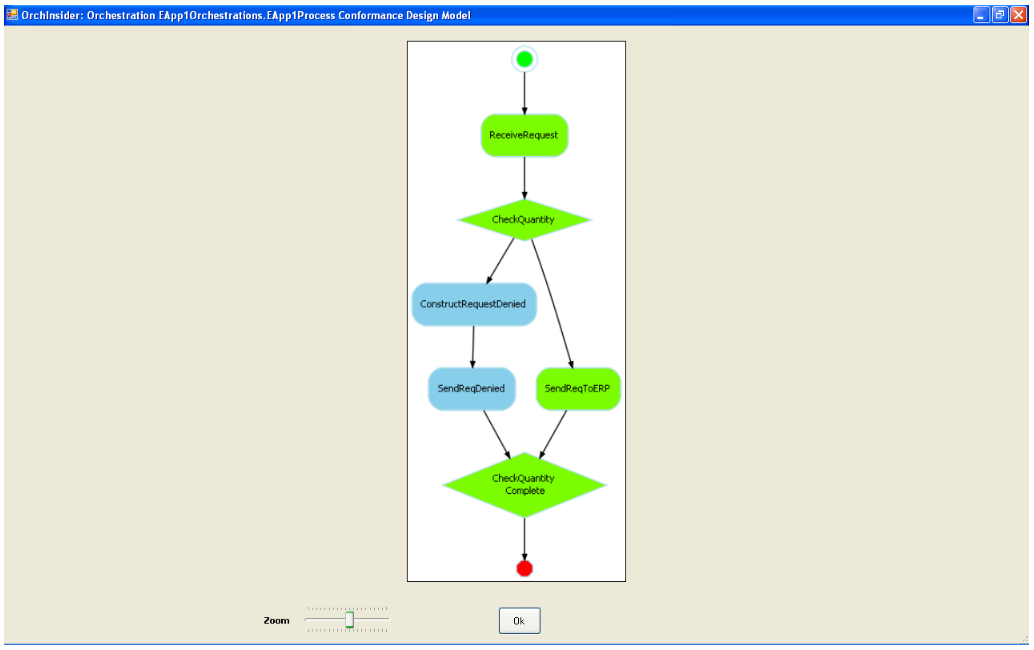


Figure 5.9: Service orchestration's activity conformance model, using OrchInsider

Figure 5.9 displays the conformance model using the activity conformance approach. Observing such conformance model, one can easily detect non present behaviour in the mined model, as the activities marked blue, and present behaviour in the mined model, depicted by green activities.

Service orchestration developers and process analysts can use conformance analysis to perceive if every orchestration instance was executed accordingly and quantify the amount of behaviour executed by the deployed orchestration.

5.6 Conclusion

This chapter has specified some of the functionalities delivered by OrchInsider, a software application that implements the ADSO mining methodology presented in Chapter 4, with the purpose of discovering and analysing service orchestrations. No related work concerning specific orchestration analysis and discovery was found, nor similar applications available nowadays, which reinforces OrchInsider's relevance. Some of its features must be improved, but presently it can analyse and discover some important aspects of deployed orchestrations. In the following chapter, this mining tool and methodology will be validated and tested with further and more complex service orchestrations.

Chapter 6

Case Study: *SportTicket Online*

In order to validate the proposed ADSO mining methodology[1], it is necessary to test the solution with other deployed orchestrations. A simple example, the depicted orchestration from Section 2.3, was already used to introduce Orchinsider and validate its functionalities according to the proposed methodology. Another business scenario should also be tested, using a bigger and more complex service orchestration. The purpose and guideline of this chapter is to properly utilize OrchInsider, applying the service orchestrations' mining methodology, so it is possible to answer some of the questions relative to service orchestrations already presented in Section 3.1, and listed in Table 6.1, and others, concerning the specific business scenarios.

Question	Section/s
How is the service orchestration modelled?	6.2.2
What are the business rules? Are they coherent and executed correctly?	6.2.2; 6.2.3
How was the service orchestration built? How many versions does it have?	6.2.1
What are the most frequent paths and execution probabilities?	6.2.2; 6.2.3
What is the performance time for each activity? and the whole orchestration's?	6.2.3
What is the performance time for external services?	6.2.3
Are there any bottlenecks? And critical paths? Is the process model efficient?	6.2.2; 6.2.3
How does the orchestration's run-time behaviour conform with the designed model?	6.2.1
What are the external service communications?	6.2.4
What is the service network architecture?	6.2.4

Table 6.1: Analysis questions concerning service orchestrations

In the following section, a brief introduction to the business scenario will be given, with some of its peculiarities highlighted. Afterwards, the ADSO mining methodology will be put into practice using *SportTicket Online's* deployed service orchestrations. Finally, results will be reported and analysed.

6.1 *SportTicket Online's* Orchestrations

The business scenario, *SportTicket Online*, was an academic project from the Enterprise Systems Integration course at the Technical University of Lisbon (IST). This project consists of relatively

[1]ADSO - Analysis and Discovery of Service Orchestrations mining methodology, was introduced in Chapter 4.

complex orchestrations that integrate several services to build a ticket reseller for the world cup football championship 2010.

The main purpose of the project is to manage ticket reselling for the world cup, supporting the buying and also the returning processes, both implemented with service orchestrations.

6.1.1 Buying Process

The buying process, orchestration "SportTicketOnline.MainOrquestracao", consists of several interconnected steps, since the game and ticket selection, pricing rules, customer data verification, payment, billing, and ticket emission. The process flow can be summarized as follows:

1. Game selection: Service that gives information about game schedule, namely the teams playing each game, date, time and place.
2. Stadium management: Service that manages each stadium's seats repositioning and ticket information. It reserves and unreserves tickets according to stadium information.
3. Interpol service: Customer data is analysed before emitting the tickets. In some cases, the process can be aborted here, if returned Interpol information so requires.
4. Pricing rules: According to the selected tickets, some business rules may be applied. Tickets may have different categories along the stadium, there may be individual or group tickets, single or multiple tickets, and children tickets, although children must always have one adult companion inside the stadium.
5. Payment service: Executes and verifies payment, sending back transaction status.
6. Billing service: An invoice with game and payment information is generated and stored in a local system.
7. Ticket emission service: Each ticket is stored in a local system, and e-mailed to the customer.

6.1.2 Returning Process

The returning process is the opposite of the buying process, from a business point of view, where payments are returned, tickets unreserved and cancelled. A ticket can only be returned until one hour before the start of the match, and only 90% of the amount payed is restored.

6.2 Mining *SportTicket Online*

In the following sections, the proposed mining methodology will be applied mainly using the buying process orchestration, given that the returning process is a much simpler orchestration that only reverses the buying business process. Nevertheless, the returning process will be of use when mining *SportTicket Online's* full service network architecture, since both orchestrations separately constitute the ticked reselling management service.

6.2.1 Analysis and Discovery

Considering the buying process, and loading this orchestration into OrchInsider, one can check that there were multiple orchestration instances executed, and three orchestration versions are present.

Table 6.2 shows version information from *SportTicket Online's* buying process, where every version's last deployment date is displayed. Section 5.1 already presented a small part of the version analysis using *SportTicket Online's* buying process. Versions three and two were compared in Figure 5.2, where it is possible to detect discrepancies between both versions. Versions one and two were development versions, which means that mostly every instance was executed for testing purposes only, whereas version three is the "final" orchestration version, where buying processes are actually being fully performed, hence, only instances from this version will be utilized to discover the mined process model.

Version Number	Last Deployment Date
1	31-05-2009 19:36:07.643
2	04-06-2009 02:31:25.030
3	16-06-2009 14:07:59.780

Table 6.2: Version information of *SportTicket Online's* buying process

Before discovering the mined process model, conformance analysis can be used to acquire previous knowledge about each deployed orchestration's instance, and this way, helping to better define mining filters according to what is intended, such as specific instances or versions (cf. Section 4.4). For this reason, conformance is checked initially.

One can be interested in analysing only one type of orchestration instances, complete or terminated instances, or both, depending on which type of analysis is desired. Since the goal is to acquire the mined orchestration's model, terminated instances and older versions will be discarded. Of course, other options could be selected, according to the analyst's objectives. If the purpose is to understand when and why such instances were terminated abruptly, then terminated instances should be selected. Mining only last version's completed instances, one can then apply conformance analysis and visualize the activity conformance model (cf. Section 5.5), portrayed in Figures 6.1 and 6.2, which not only illustrate the complexity of the orchestration, but also depict branches that were not executed at run-time, therefore identifying non-present behaviour in the mined orchestration's process model.

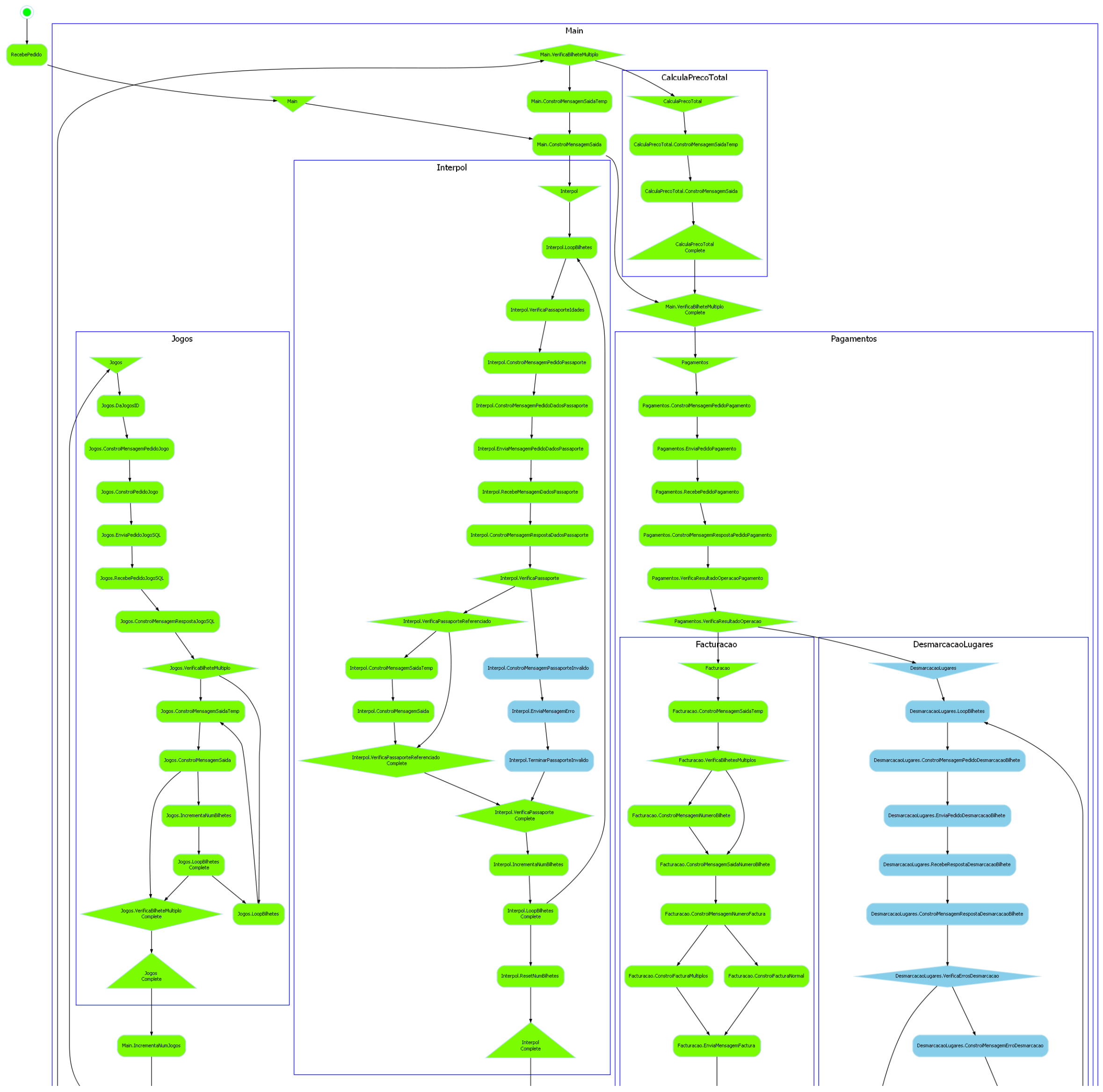

Main
RecebePedido
Main.VerificaBilheteMultiplo
Main
Main.ConstroiMensagemSaidaTemp
Main.ConstroiMensagemSaida
CalculaPrecoTotal
CalculaPrecoTotal
CalculaPrecoTotal.ConstroiMensagemSaidaTemp
CalculaPrecoTotal.ConstroiMensagemSaida
CalculaPrecoTotal Complete
Main.VerificaBilheteMultiplo Complete
Interpol
Interpol
Interpol.LoopBilhetes
Interpol.VerificaPassaporteIdades
Interpol.ConstroiMensagemPedidoPassaporte
Interpol.ConstroiMensagemPedidoDadosPassaporte
Interpol.EnviaMensagemPedidoDadosPassaporte
Interpol.RecebeMensagemDadosPassaporte
Interpol.ConstroiMensagemRespostaDadosPassaporte
Interpol.VerificaPassaporte
Interpol.VerificaPassaporteReferenciado
Interpol.ConstroiMensagemSaidaTemp
Interpol.ConstroiMensagemSaida
Interpol.ConstroiMensagemPassaporteInvalido
Interpol.EnviaMensagemErro
Interpol.TerminarPassaporteInvalido
Interpol.VerificaPassaporteReferenciado Complete
Interpol.VerificaPassaporte Complete
Interpol.IncrementaNumBilhetes
Interpol.LoopBilhetes Complete
Interpol.ResetNumBilhetes
Interpol Complete
Jogos
Jogos
Jogos.DaJogosID
Jogos.ConstroiMensagemPedidoJogo
Jogos.ConstroiPedidoJogo
Jogos.EnviaPedidoJogoSQL
Jogos.RecebePedidoJogoSQL
Jogos.ConstroiMensagemRespostaJogoSQL
Jogos.VerificaBilheteMultiplo
Jogos.ConstroiMensagemSaidaTemp
Jogos.ConstroiMensagemSaida
Jogos.IncrementaNumBilhetes
Jogos.LoopBilhetes Complete
Jogos.VerificaBilheteMultiplo Complete
Jogos.LoopBilhetes
Jogos Complete
Main.IncrementaNumJogos
Pagamentos
Pagamentos
Pagamentos.ConstroiMensagemPedidoPagamento
Pagamentos.EnviaPedidoPagamento
Pagamentos.RecebePedidoPagamento
Pagamentos.ConstroiMensagemRespostaPedidoPagamento
Pagamentos.VerificaResultadoOperacaoPagamento
Pagamentos.VerificaResultadoOperacao
Facturacao
Facturacao
Facturacao.ConstroiMensagemSaidaTemp
Facturacao.VerificaBilhetesMultiplos
Facturacao.ConstroiMensagemNumeroBilhete
Facturacao.ConstroiMensagemSaidaNumeroBilhete
Facturacao.ConstroiMensagemNumeroFactura
Facturacao.ConstroiFacturaMultiplos
Facturacao.ConstroiFacturaNormal
Facturacao.EnviaMensagemFactura
DesmarcacaoLugares
DesmarcacaoLugares
DesmarcacaoLugares.LoopBilhetes
DesmarcacaoLugares.ConstroiMensagemPedidoDesmarcacaoBilhete
DesmarcacaoLugares.EnviaPedidoDesmarcacaoBilhete
DesmarcacaoLugares.RecebeRespostaDesmarcacaoBilhete
DesmarcacaoLugares.ConstroiMensagemRespostaDesmarcacaoBilhete
DesmarcacaoLugares.VerificaErrosDesmarcacao
DesmarcacaoLugares.ConstroiMensagemErroDesmarcacao

Figure 6.1: Process conformance analysis of *SportTicket Online's* buying process (a)

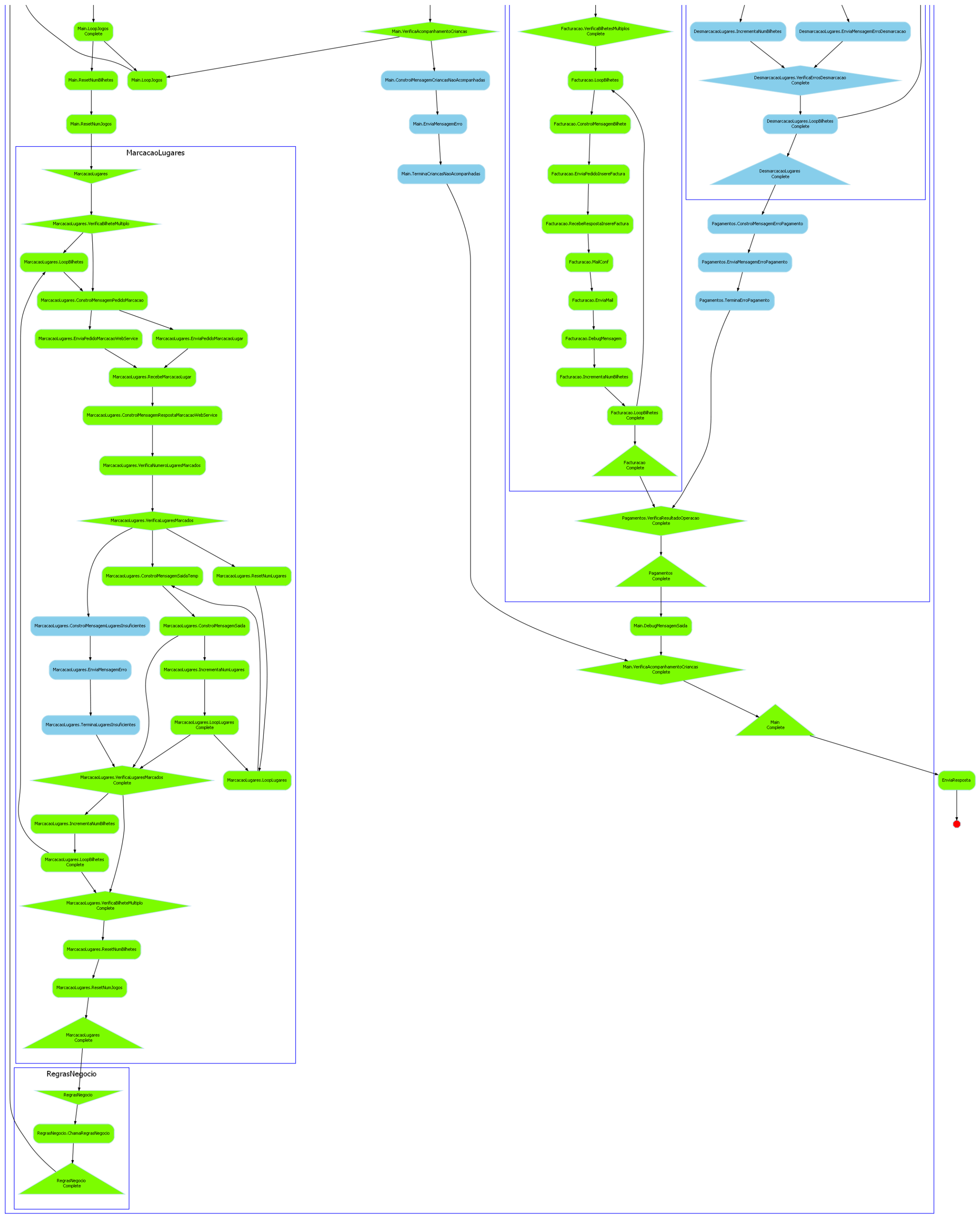


Figure 6.2: Process conformance analysis of *SportTicket Online's* buying process (b)

OrchInsider's conformance analysis quantifies the depicted behavioural conformance as being 79%. From the total of 239 transitions, 50 were not present in the mined process. Activities that belong to both models are conform activities and are marked green, whilst mined-only activities, which are activities that only belong to the mined model, appear in red, and parsed-only activities, representing activities that exist only in the parsed model[2], appear in blue. Since there are not any mined-only activities, one can conclude that model conformance was checked with correspondent orchestration versions, and that versioning was performed accordingly (cf. Section 4.8). Observing the conformance model, depicted in Figures 6.1 and 6.2, it is possible to perceive which behaviour was not performed by the mined buying process model:

1. No buying process was aborted due to “negative” Interpol information. This can be observed inside the Interpol scope (cf. Sections 4.6 and 5.3), since no “*TerminarPassaporteInvalido*” activity was executed in the mined process.

2. No tickets were unmarked due to payment failure, since scope macro activity *“DesmarcacaoLugares”* was never executed.

3. No process was interrupted due to unaccompanied children. Activity *“TerminaCriancasNaoAcompanhadas”*, in the *“Main”* scope, was never executed.

4. Every buying process completed successfully the stadium management service, meaning that the process was never interrupted because of sold out tickets. One can observe that the activity *“TerminaLugaresInsuficientes”*, inside the *“MarcacaoLugares”* scope, is also never executed.

With such previous knowledge of the mined process model and its conform behaviour, one can then visualize and extend the mined orchestration’s design model with further useful information.

6.2.2 Design Model Visualization

After analysing which behaviour was not performed by the mined buying process, one can visualize the orchestration’s design model and enrich such model with more information that can help to characterize the mined orchestration’s executed behaviour. To better understand activity performance and path frequency, the analyst can define a bottleneck threshold[3], according to defined process expectations, and visualize the mined process model with identified bottleneck activities, path frequency and probabilistic analysis (cf. Section 5.3). Selecting average time as the performance indicator (cf. Section 4.7), and one second as the bottleneck threshold, Figures 6.3, 6.4, and 6.5, display the mined orchestration’s model enhanced with performance analysis visualization.

[2]The initially designed and modelled process is referred as parsed model.
[3]Bottleneck threshold represents activity duration, either average, maximum or minimum, in seconds.

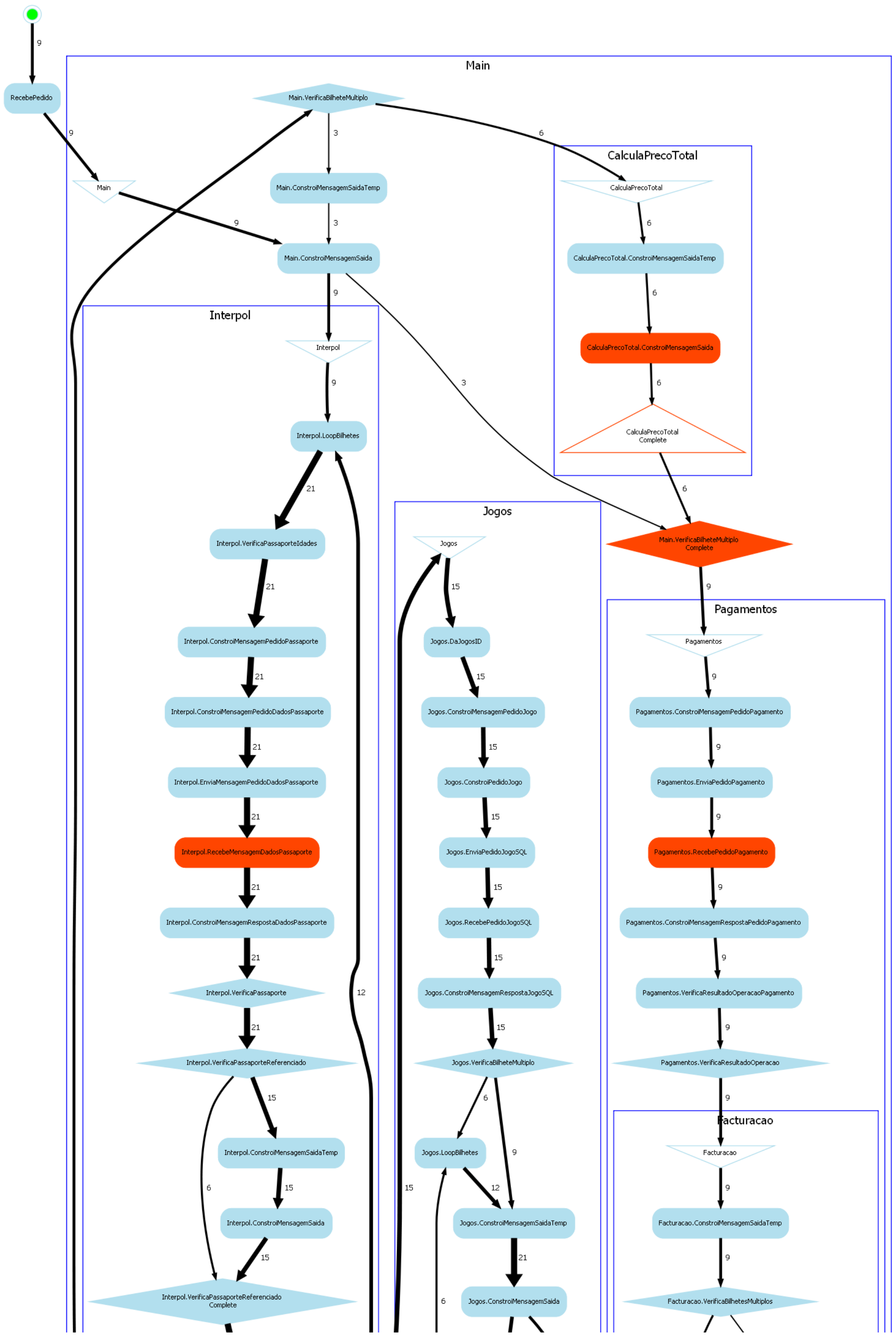

Main
RecebePedido
Main.VerificaBilheteMultiplo
Main
Main.ConstroiMensagemSaidaTemp
Main.ConstroiMensagemSaida
CalculaPrecoTotal
CalculaPrecoTotal.ConstroiMensagemSaidaTemp
CalculaPrecoTotal.ConstroiMensagemSaida
CalculaPrecoTotal Complete
Main.VerificaBilheteMultiplo Complete
Interpol
Interpol.LoopBilhetes
Interpol.VerificaPassaporteIdades
Interpol.ConstroiMensagemPedidoPassaporte
Interpol.ConstroiMensagemPedidoDadosPassaporte
Interpol.EnviaMensagemPedidoDadosPassaporte
Interpol.RecebeMensagemDadosPassaporte
Interpol.ConstroiMensagemRespostaDadosPassaporte
Interpol.VerificaPassaporte
Interpol.VerificaPassaporteReferenciado
Interpol.ConstroiMensagemSaidaTemp
Interpol.ConstroiMensagemSaida
Interpol.VerificaPassaporteReferenciado Complete
Jogos
Jogos.DaJogosID
Jogos.ConstroiMensagemPedidoJogo
Jogos.ConstroiPedidoJogo
Jogos.EnviaPedidoJogoSQL
Jogos.RecebePedidoJogoSQL
Jogos.ConstroiMensagemRespostaJogoSQL
Jogos.VerificaBilheteMultiplo
Jogos.LoopBilhetes
Jogos.ConstroiMensagemSaidaTemp
Jogos.ConstroiMensagemSaida
Pagamentos
Pagamentos.ConstroiMensagemPedidoPagamento
Pagamentos.EnviaPedidoPagamento
Pagamentos.RecebePedidoPagamento
Pagamentos.ConstroiMensagemRespostaPedidoPagamento
Pagamentos.VerificaResultadoOperacaoPagamento
Pagamentos.VerificaResultadoOperacao
Facturacao
Facturacao.ConstroiMensagemSaidaTemp
Facturacao.VerificaBilhetesMultiplos

Figure 6.3: *SportTicket Online's* mined buying process enhanced with performance analysis (a)

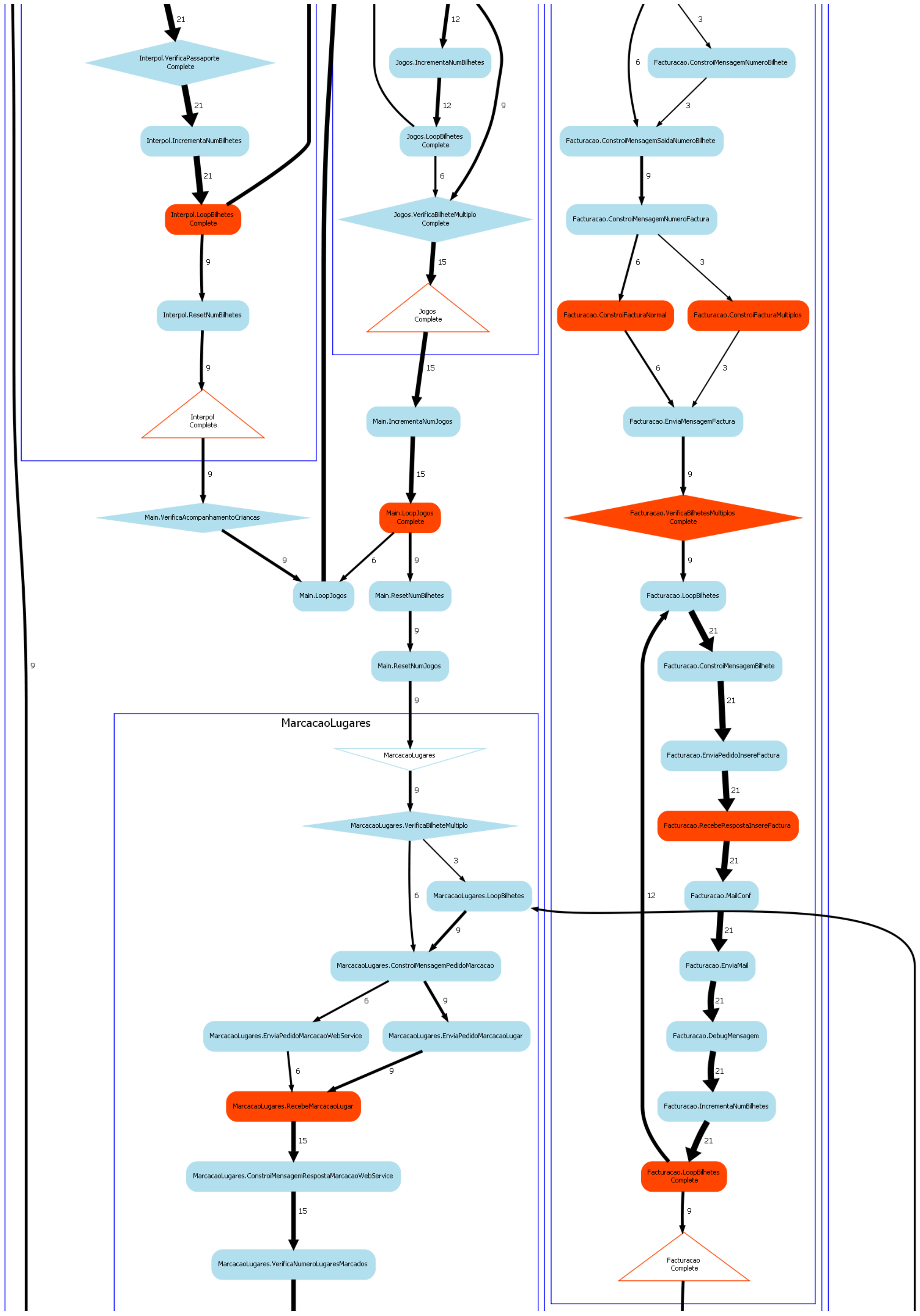

Interpol.VerificaPassaporte Complete
Interpol.IncrementaNumBilhetes
Interpol.LoopBilhetes Complete
Interpol.ResetNumBilhetes
Interpol Complete
Jogos.IncrementaNumBilhetes
Jogos.LoopBilhetes Complete
Jogos.VerificaBilheteMultiplo Complete
Jogos Complete
Main.VerificaAcompanhamentoCriancas
Main.IncrementaNumJogos
Main.LoopJogos Complete
Main.LoopJogos
Main.ResetNumBilhetes
Main.ResetNumJogos
MarcacaoLugares
MarcacaoLugares.VerificaBilheteMultiplo
MarcacaoLugares.LoopBilhetes
MarcacaoLugares.ConstroiMensagemPedidoMarcacao
MarcacaoLugares.EnviaPedidoMarcacaoWebService
MarcacaoLugares.EnviaPedidoMarcacaoLugar
MarcacaoLugares.RecebeMarcacaoLugar
MarcacaoLugares.ConstroiMensagemRespostaMarcacaoWebService
MarcacaoLugares.VerificaNumeroLugaresMarcados
Facturacao.ConstroiMensagemNumeroBilhete
Facturacao.ConstroiMensagemSaidaNumeroBilhete
Facturacao.ConstroiMensagemNumeroFactura
Facturacao.ConstroiFacturaNormal
Facturacao.ConstroiFacturaMultiplos
Facturacao.EnviaMensagemFactura
Facturacao.VerificaBilhetesMultiplos Complete
Facturacao.LoopBilhetes
Facturacao.ConstroiMensagemBilhete
Facturacao.EnviaPedidoInsereFactura
Facturacao.RecebeRespostaInsereFactura
Facturacao.MailConf
Facturacao.EnviaMail
Facturacao.DebugMensagem
Facturacao.IncrementaNumBilhetes
Facturacao.LoopBilhetes Complete
Facturacao Complete

Figure 6.4: *SportTicket Online's* mined buying process enhanced with performance analysis (b)

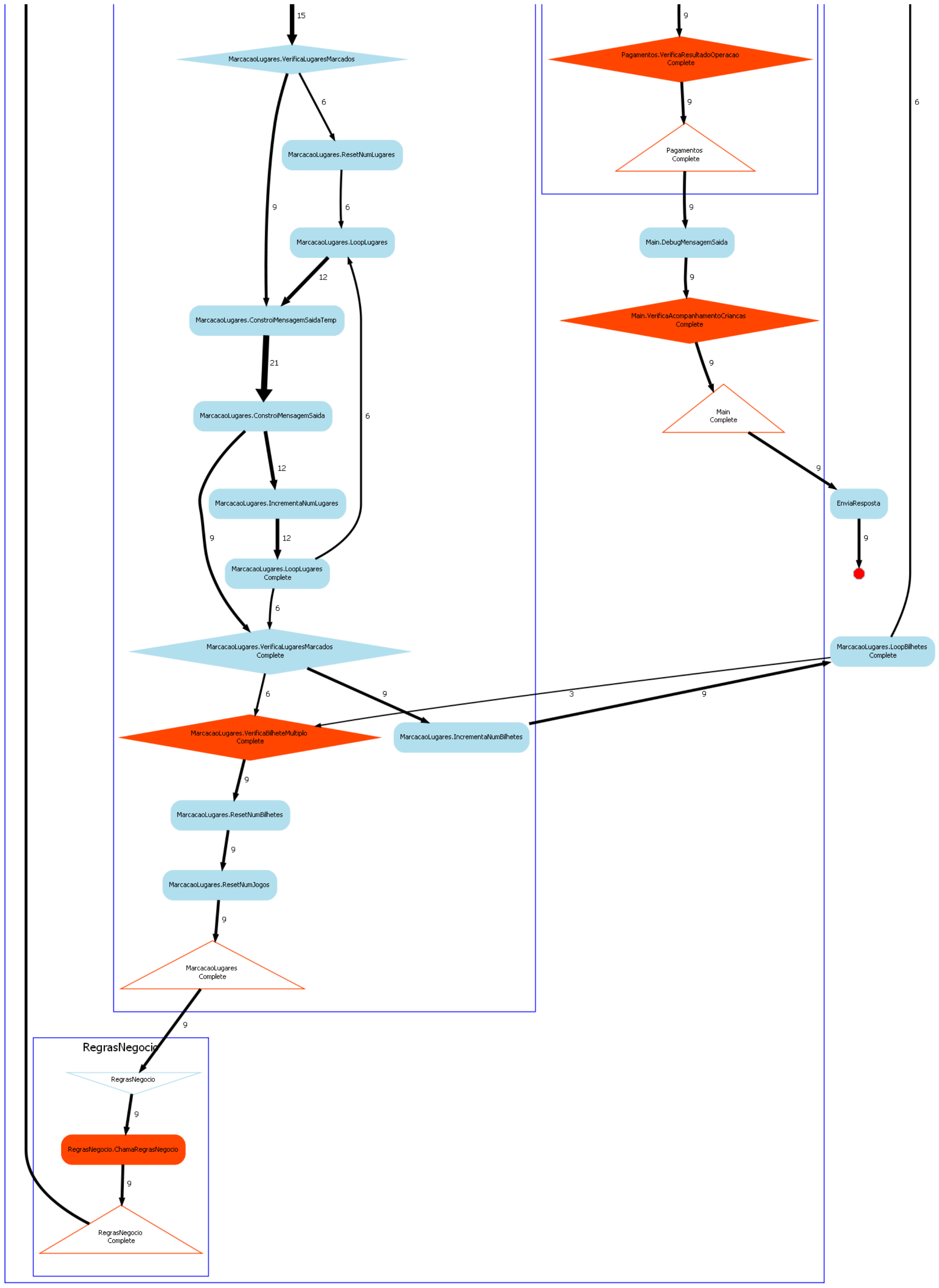

15
MarcacaoLugares.VerificaLugaresMarcados
6
MarcacaoLugares.ResetNumLugares
9
6
MarcacaoLugares.LoopLugares
12
MarcacaoLugares.ConstroiMensagemSaidaTemp
21
MarcacaoLugares.ConstroiMensagemSaida
6
12
MarcacaoLugares.IncrementaNumLugares
9
12
MarcacaoLugares.LoopLugares Complete
6
MarcacaoLugares.VerificaLugaresMarcados Complete
6
9
MarcacaoLugares.VerificaBilheteMultiplo Complete
MarcacaoLugares.IncrementaNumBilhetes
3
9
9
MarcacaoLugares.ResetNumBilhetes
9
MarcacaoLugares.ResetNumJogos
9
MarcacaoLugares Complete
9
RegrasNegocio
RegrasNegocio
9
RegrasNegocio.ChamaRegrasNegocio
9
RegrasNegocio Complete
9
Pagamentos.VerificaResultadoOperacao Complete
9
Pagamentos Complete
9
Main.DebugMensagemSaida
9
Main.VerificaAcompanhamentoCriancas Complete
9
Main Complete
9
EnviaResposta
9
6
MarcacaoLugares.LoopBilhetes Complete

Figure 6.5: *SportTicket Online's* mined buying process enhanced with performance analysis (c)

Observing the mined orchestration's process model presented in Figures 6.3, 6.4 and 6.5, it is possible to perceive that:

1. The buying process was instantiated and executed 9 times, and the maximum path frequency[4] present in the mined model is 21.

2. The Interpol service was invoked 21 times. There is one activity inside the *"Interpol"* scope, *"RecebeMensagemDadosPassaporte"*, that exceeds the configured bottleneck threshold[5]. This may indicate that such path, being a highest frequency path, constitutes a critical path[6].

3. The game selection service, which is invoked inside the *"Jogos"* scope, was used 15 times. This means that 21 users bought tickets for 15 games.

4. The payment service, invoked inside the *"Pagamentos"* scope, was used 9 times. This means that 9 people payed 21 tickets for 15 games. Also, there is one activity inside the payment scope, *"RecebePedidoPagamento"*, that surpasses the bottleneck threshold.

5. The ticket emission service, which is invoked inside the billing scope, *"Facturacao"*, was used 21 times, as expected. There is also one activity, *"RecebeRespostaInsereFactura"*, that surpasses the configure threshold in this path, which might indicate a critical path, since a high frequency path is present with a bottleneck activity.

6. There may also be a critical path when the stadium management service is invoked, which happens inside the *"MarcacaoLugares"* scope. The activity *"RecebeMarcacaoLugar"* exceeds the configured threshold and has a frequency of 15, which is close to the maximum path frequency of 21. Since a high frequency path contains a bottleneck activity, a critical path may be considered.

7. Looking at the stadium management scope, *"MarcacaoLugares"*, it is also possible to understand that, from the 9 buying processes executed, 6 were related to single tickets, while 3 are multiple tickets, representing the other 9 games, from the total of 15. This can also be confirmed observing the billing scope, *"Facturacao"*, where the activity *"ConstroiFacturaNormal"* is executed 6 times, and the activity *"ConstroiFacturaMultiplos"* is executed 3 times. *"ConstroiFacturaNormal"* is the activity that creates single ticket invoices, whereas *"ConstroiFacturaMultiplos"* delivers multiple ticket invoices. Both activities exceed the configured threshold, which might indicate critical paths. Although each edge does not constitute a high frequency edge, the truth is that either one of them include bottleneck activities.

8. Business pricing rules are performed at the *"RegrasNegocio"* scope, by the activity *"ChamaRegrasNegocio"*, which exceeds the configured threshold, and has a frequency of 9, also confirming the total number of buying processes executed.

Other business conclusions could also be drawn, according to the observer's initial understanding of the process. As mentioned before in Section 3.1, process mining results must work side by side with the modelled processes' owners, analysts and developers so as to embrace the highest process

[4]Every edge's width is correlated to the edge's frequency number and the maximum edge frequency found in the service orchestration.

[5]Bottleneck activities appear in red.

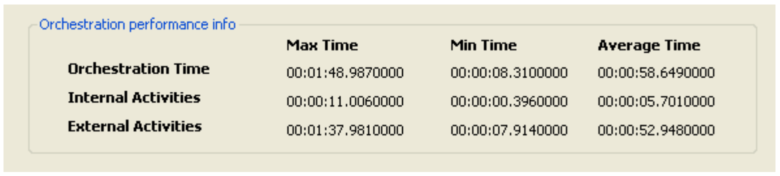
[6]Path frequency visualization displays every edge's frequency number.

comprehension possible. Analysts can then pinpoint and "zoom" in information, in order to relate the acquired business-oriented knowledge with more specific performance information.

6.2.3 Performance Analysis

Analysing performance quantifies and improves the characterization of the mined orchestration's execution. Model visualization helps to gather performed business-oriented knowledge, and performance analysis strengthens such knowledge, clarifying uncertainties.

Figure 6.6 presents *SportTicket Online's* buying process performance information, retrieved from OrchInsider's performance analysis. Considering maximum execution times, one can observe that


Orchestration performance info

	Max Time	Min Time	Average Time
Orchestration Time	00:01:48.9870000	00:00:08.3100000	00:00:58.6490000
Internal Activities	00:00:11.0060000	00:00:00.3960000	00:00:05.7010000
External Activities	00:01:37.9810000	00:00:07.9140000	00:00:52.9480000

Figure 6.6: *SportTicket Online's* buying process' performance information

the orchestration's execution time was approximately 1 minute, and 49 seconds; Internal activities represent 0:11.006 whereas external activities stand for 1:37.981 of the total execution time. So, almost all of the mined orchestration's execution time is consumed "outside" the orchestration, in invoked external services. Since it is also possible to retrieve each activity's performance indicators, it is important to verify former section's detected bottleneck activities, along with some other interesting activities, which are displayed in Table 6.3. Activity frequency is also presented, in the *"#"* row, indicating the number of times each activity was performed.

Activity Name	Scope	Max. Time	Min. Time	Avg. Time	#
"RecebeMensagemDadosPassaporte"	*"Interpol"*	00:28.344	00:01:170	00:14.757	21
"RecebePedidoPagamento"	*"Pagamentos"*	00:01.953	00:00.700	00:01.327	9
"RecebeRespostaInsereFactura"	*"Facturacao"*	01:00.217	00:00.500	00:30.359	21
"RecebeMarcacaoLugar"	*"MarcacaoLugares"*	00:02.860	00:00.687	00:01.774	15
"ConstroiFacturaNormal"	*"Facturacao"*	00:02.504	00:00.000	00:01.252	6
"ConstroiFacturaMultiplos"	*"Facturacao"*	00:01.250	00:00.784	00:01.017	3
"ChamaRegrasNegocio"	*"RegrasNegocio"*	00:02.450	00:00.000	00:01.225	9
"Interpol"	-	00:29.983	00:04.513	00:17.248	9
"Pagamentos"	-	01:08.157	00:02.300	00:35.229	9
"Facturacao"	-	01:06.127	00:01.550	00:33.839	9
"MarcacaoLugares"	-	00:03.250	00:00.750	00:02.000	9
"Jogos"	-	00:01.550	00:00.500	00:01.025	15

Table 6.3: Activity performance indicators *of SportTicket Online's* buying process

- Activity *"RecebeMensagemDadosPassaporte"* is a *ReceiveShape* type activity that receives a response from the Interpol service. Considering the presented performance indicators, it is possible to apprehend that, either the external Interpol service is taking, in some cases, quite a while to respond, or there are communication issues to analyse. Discrepancies between maximum and minimum execution times may reveal that both cases must be considered. Also, due to the high frequency of this activity, the Interpol service could improve its efficiency.

- *"RecebePedidoPagamento"* and *"RecebeMarcacaoLugar"* are also activities that receive responses from external services, the former from the payment service and the latter from the stadium management service. In both cases however, execution times are not overwhelmingly high.

- Considering activity *"RecebeRespostaInsereFactura"*, which is a *ReceiveShape* type activity that waits for an answer from the local ticket emission service, one can perceive that such service might be taking quite a while to perform its operations, namely receive each ticket information and store it in a local system for further analysis and management. Also, due to high discrepancies between maximum and minimum execution times, and high frequency values, the local ticket emission service could be analysed in order to improve its efficiency.

- Analysing scope "macro" activities' performance indicators, such as the *"Interpol"* scope, where the Interpol service is invoked, it is possible to really understand the distribution of execution time performance inside the service orchestration. Considering again maximum execution times, one can observe that *"Pagamentos"*, *"Facturacao"* and *"Interpol"* scopes represent macro activities with the highest execution times, whereas *"MarcacaoLugares",* and *"Jogos"* scopes have the lowest execution times inside the service orchestration.

Inspecting each activity's performance indicator from Table 6.3, where mostly every bottleneck activity is an activity that receives external services responses, it is possible to understand why and where such a big slice of the orchestration's execution time is indeed consumed in external services, which conforms with the presented performance information shown in Figure 6.6.

The purpose so far has been to analyse and discover the mined process model along with performance quantification. Nevertheless, after understanding and characterizing correctly executed and completed orchestration instances, analysts may be interested in terminated[7] instances also, in order to understand what went wrong and why. A terminated instance represents an orchestration's execution that did not complete its entire flow and was terminated abruptly. Completing its entire flow, means that the orchestration's instance starts in one start activity, and ends in one predicted end activity[8]. Path analysis can be utilized to perceive terminated instances, observing end activities.

Mining last version's complete and terminated instances, OrchInsider's path analysis shows that several end activities were found. Since terminated instances are present, a slightly exaggerated and unexpected number of end activities will always appear. Observing non-predicted end activities, which are displayed in Table 6.4, can help to understand where orchestration instances failed their execution.

End Activities	Scope	#
"RecebeMensagemDadosPassaporte"	*"Interpol"*	1
"RecebeMarcacaoLugar"	*"MarcacaoLugares"*	5
"EnviaPedidoMarcacaoWebService"	*"MarcacaoLugares"*	1
"EnviaMensagemFactura"	*"Facturacao"*	12

Table 6.4: Non-predicted end activities retrieved from *SportTicket Online's* buying process

[7]Service orchestration instances can be in execution, completed or terminated states.
[8]Last activity executed by one orchestration instance.

Sporticket Online's buying process was terminated abruptly 19 times, 12 of which were due to failures when calling the billing service, seeing that the activity *"EnviaMensagemFactura"* is responsible to invoke the billing service. Another 6 failures were due to connections with the stadium management service, since both activities, *"EnviaPedidoMarcacaoWebService"* and *"RecebeMarcacaoLugar"* call the stadium management service and receive answers from such service, respectively. One terminated buying process happened while receiving an answer from the Interpol service, which might indicate, either a failure from such service, or a connection problem.

Performance analysis has proven useful to broaden orchestration knowledge, either understanding and characterizing correctly executed orchestration instances and external services, but also inspecting abruptly terminated orchestration instances, and identifying problems concerning external services. Analysts can therefore quantify and correctly evaluate orchestration execution performance.

6.2.4 Service Network Architecture

Once the mined model is captured and characterized, it is interesting to realize *SportTicket Online's* buying process communication behaviour, understanding which external services were invoked during execution. After loading the buying process orchestration into Orchinsider, every instance's port information from every version is displayed. 14 communication ports are retrieved for the buying process orchestration. Considering once again only last version's instances, Figure 6.7 depicts the buying process' service network architecture, where every invoked service communication is displayed[9] (cf. Sections 4.9 and 5.2).

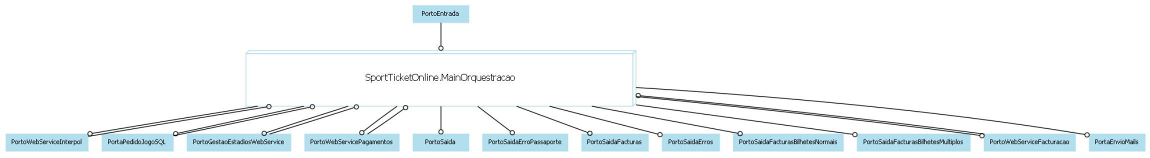

PortoEntrada
SportTicketOnline.MainOrquestracao
PortoWebServiceInterpol
PortaPedidoJogoSQL
PortoGestaoEstadiosWebService
PortoWebServicePagamentos
PortoSaida
PortoSaidaErroPassaporte
PortoSaidaFacturas
PortoSaidaErros
PortoSaidaFacturasBilhetesNormais
PortoSaidaFacturasBilhetesMultiplos
PortoWebServiceFacturacao
PortaEnvioMails

Figure 6.7: Service network architecture of *SportTicket Online's* buying process

Although *SportTicket Online's* returning process was not analysed, it is useful to understand its external service communications, which are depicted in Figure 6.8.

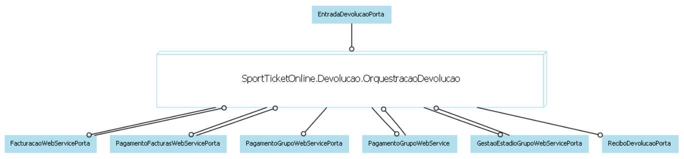

EntradaDevolucaoPorta
SportTicketOnline.Devolucao.OrquestracaoDevolucao
FacturacaoWebServicePorta
PagamentoFacturasWebServicePorta
PagamentoGrupoWebServicePorta
PagamentoGrupoWebService
GestaoEstadioGrupoWebServicePorta
ReciboDevolucaoPorta

Figure 6.8: Service network architecture of *SportTicket Online's* returning process

With communication information from both service orchestrations, it is possible to realize *SportTicket Online's* full service network architecture.

[9]Port direction information is displayed using small non-filled dots that represent receive direction ports.

6.3 Discussion

After applying the proposed methodology, using Orchinsider, some of the analysis questions listed in Table 6.1, along with others, concerning the specific business scenarios, could be answered, and understood. *SportTicket Online's* buying process was analysed and its deployed orchestration discovered, enabling developers and process analysts to:

1. Comprehend executed behaviour and compare it with the initially designed model.
2. Quantify mined and designed model differences.
3. Perceive process development along time.
4. Visualize the mined buying process orchestration.
5. Identify, quantify and correctly evaluate existing problems and inefficiencies, bottlenecks and critical paths.
6. Measure performance and understand where possible improvements can be accommodated.
7. Inspect failures and understand why and when then occurred.
8. Understand external service communications and depict the orchestration's service network architecture.

With such delivered insights, organization analysts and developers can achieve a higher comprehension of the process execution and development, and this way, correctly measure, evaluate and reengineer business processes.

OrchInsider and the proposed ADSO mining methodology can be a valuable asset to organizations, delivering information and knowledge that can help top managers to monitor and seize control over business processes, implemented under the form of service orchestrations, firmly understanding how processes are executed and work in reality.

Chapter 7

Conclusion

Organizations need to seize control over their business processes, in order to keep up with globalization and interoperability concerns, but also with new legislation and regulation acts. BPM and SOA have emerged and enabled Enterprise Application Integration in the last years, allowing service reutilization and cross-organizational business processes, reflecting real-world processes and business relationships more closely. BPM extends typical workflow systems with process observation and analysis, aiming to achieve higher process awareness. Using SOA, an organization can orchestrate several distributed services, and deliver functionalities that can be utilized by other organizations, creating distributed business processes that span several organizations, thus choreographing a distributed service architecture.

Service orchestrations define another abstraction layer in which services are built upon existing services, allowing for service reutilization and composition, business collaboration. Although they enable service composition and therefore business collaboration, service orchestrations are centralized processes that can also be deployed as services and communicate with other services or orchestrations, this way becoming integration enablers and SOA-based services.

7.1 Contributions

This work has presented an overview of integration platforms and their major research areas, with special concern about integration solutions under the form of service orchestrations. Also, it has introduced process mining as an extension to workflow systems, with the purpose of extracting useful information from deployed orchestrations' execution event logs. Using process mining techniques, an orchestration mining methodology was proposed and developed, with the purpose of enabling organizations to analyse the run-time behaviour of service orchestrations, extract process models and compare them with their original models in order to improve or redesign business processes. The ADSO mining methodology was applied by a software application, OrchInsider, that was developed to analyse and discover service orchestrations using Microsoft Biztalk integration server. The proposed solution has delivered some relevant information and brought into the light other aspects of service orchestration analysis, such as distributed services, or choreographies. At present times, there is no related work concerning specific service orchestration analysis and discovery, nor similar applications available, which reinforces the relevance of this work.

Applying process mining techniques to deployed orchestrations using their run-time generated event logs has delivered useful insights that can be used by organizations to seize knowledge control over their business processes.

- Understand executed behaviours and compare discovered models with previously idealized and designed processes.
- Visualize the process model, and perceive its development and growth along time.
- Measure service performance, internal and external, and understand possible improvements.
- Identify and correctly evaluate existing problems, failures and inefficiencies, and understand why and when then occurred.
- Realize the orchestration's external communications and network architecture.

Organizations are continuously pressured to monitor their business processes, either by performance and adaptability issues, or by legislation and regulation acts. Higher governance maturity concerns are increasing inside organizations, so, more internal control requirements, methods and skills are in fact, crucial to an organization's survival. Understanding how organizations work is better achieved and realized by correctly managing the organization's business processes, and process mining has proven its usefulness and importance, by analysing and discovering business processes developed, deployed and executed under the form of service orchestrations.

7.2 Future Work

Service orchestrations are centralized services, and even though other services are invoked and executed, the proposed ADSO methodology and solution only retrieves information from the deployed and "central hub" orchestration. Service network is centralized in the orchestration, since there is only communication information available from such orchestration, as mentioned in Section 4.9. The analysis and discovery of the control-flow, or process perspective, is only related to a single deployed orchestration. This centralized approach could lead to building a new mining perspective, an interoperability, or choreography, where every orchestration is seen as a single service node, each orchestration's model and performance information is retrieved, and communication information from every node is correlated, so a distributed orchestration network, a choreography, is discovered. Section 4.9 has somewhat introduced these challenges and revealed some of the steps and issues that could lead to a possible solution:

- Gather every orchestration's log in a joint event log.
- Link every orchestration's instance with every communication message.

The first issue could be solved using some kind of a log integration platform, but even so, every orchestration's log must be aggregated in such platform. The second issue seems to pose a different challenge. One idea would be to use timestamps, but this brings another problem, time synchronization. All nodes should be synchronized with exactly the same time, or use some kind of a time server. Nevertheless, orchestrations' communications delivering distributed services, or choreographies, are unsolved issues and present themselves as investigation opportunities.

Bibliography

W. M. P. van der Aalst. Don't go with the flow: web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, 2003.

W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters. Genetic process mining. In *26th International Conference on Applications and Theory of Petri Nets (ICATPN)*, pages 48–69, 2005.

W.M.P. van der Aalst and Kees Van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, 2002.

W.M.P. van der Aalst, A. H. M. Ter Hofstede, and M. Weske. Business process management: A survey. In *Proceedings of the 1st International Conference on Business Process Management, volume 2678 of LNCS*, pages 1–12. Springer, 2003a.

W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business process mining: An industrial application. *Information Systems*, 32:713–732, 2007.

W.M.P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003b.

W.M.P. van der Aalst, B.F. van Dongen, C. Günther, A. Rozinat, H. M. W. Verbeek, and A. J. M. M. Weijters. ProM: The process mining toolkit. In *BPM 2009 Demonstration Track, Volume 489 of CEUR-WS.org, Ulm, Germany, September 8*, 2009.

W.M.P. van der Aalst and H.M.W. Verbeek. Process mining in web services: The websphere case. *IEEE Data Engineering Bulletin*, 31:45–48, 2008.

W.M.P. van der Aalst and A.J.M.M. Weijters. Process mining: A research agenda. *Computers and Industry*, 53:231–244, 2004.

W.M.P. van der Aalst and A.J.M.M. Weijters. *Process-Aware Information Systems: Bridging People and Software through Process Technology*, chapter Process Mining, pages 235–255. Wiley & Sons, 2005.

W.M.P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16:1128–1142, 2004.

Stephen P. Borgatti. Centrality and network flow. *Social Networks*, 27(1):55–71, January 2005.

Melike Bozkaya, Joost Gabriels, and Jan Martijn van der Werf. Process diagnostics: A method based on process mining. *International Conference on Information, Process, and Knowledge Management (eKNOW)*, 0:22–27, 2009.

Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration: a synergic approach for system design. In *In Proc. of 4th International Conference on Service Oriented Computing (ICSOC)*, pages 228–240. Springer, 2005.

Carine Courbis and Martlesham Heath. Weaving aspects into web service orchestrations. In *Proceedings of the 3rd IEEE International Conference on Web Services (ICWS 2005)*, pages 69–77, July 11-15 2005.

Gero Decker, Frank Puhlmann, and Mathias Weske. *Business Process Management*, volume 4102/2006, chapter Formalizing Service Interactions, pages 414–419. Springer Berlin, 2006.

Giusy Di Lorenzo. *Methodologies, architecture and tools for automated service composition in SOA*. PhD thesis, Università degli Studi di Napoli "Federico II", 2008.

B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In *26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005), G. Ciardo and P. Darondeau, LNCS 3536*, pages 444–454, 2005.

Marlon Dumas, Wil M. van der Aalst, and Arthur H. ter Hofstede, editors. *Process-Aware Information Systems : Bridging People and Software through Process Technology.* Wiley-Interscience, Hoboken, NJ, 2005.

Schahram Dustdar and Robert Gombotz. Discovering web service workflows using web services interaction mining. *International Journal of Business Process Integration and Management*, 1: 256–266(11), 2007.

Schahram Dustdar, Robert Gombotz, and Karim Baïna. Web services interaction. Technical report, Technical University of Vienna, 2004.

Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

Diogo R. Ferreira. À descoberta dos processos de negócio (invited talk). ComputerWorld Advanced Hot Topics, Hotel Villa Rica, Lisboa (in portuguese), May 15 2008.

Diogo R. Ferreira and Miguel Mira da Silva. Using process mining for ITIL assessment: a case study with incident management. In *Proceedings of the 13th Annual UKAIS Conference.* Bournemouth University, April 10-11 2008.

Florin Fortis and Alexandra Fortis. Tailored business solutions by workflow technologies. *CoRR*, abs/0904.3634, 2009.

Walid Gaaloul, Karim Baïna, and Claude Godart. Log-based Mining Techniques Applied to Web Service Composition Reengineering. *Service Oriented Computing and Applications*, 2:93–110, July 2008.

Christian W. Günther and Wil M. P. van der Aalst. A generic import framework for process event logs. Technical report, Business Process Management Workshops, Workshop on Business Process Intelligence (BPI 2006), LNCS 4103, Springer, 2006.

Robert Gombotz, Karim Baïna, and Schahram Dustdar. Towards web services interaction mining architecture for e-commerce applications analysis. In *Proceedings of the Conference on E-Business and E-Learning*, 1999.

Robert Gombotz and Schahram Dustdar. On web services workflow mining. In *Business Process Management Workshops*, pages 216–229. Springer, 2005.

Shuangxi Huang and Yushun Fan. Model driven and service oriented enterprise integration—the method, framework and platform. In *Sixth International Conference on Advanced Language Processing and Web Information Technology (ALPIT 2007)*, 2007.

Kenneth C. Laudon and Jane P. Laudon. *Management Information Systems: Managing the Digital Firm.* Pearson, eleventh edition, 2009.

Christine Legner and Kristin Wende. Towards an excellence framework for business interoperability. In *BLED*, volume 29, 2006.

Frank Leymann, Dieter Roller, and Marc T. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2):198–211, 2002.

A. K. A. de Medeiros, A. J. M. M. Weijters, and Wil M. P. van der Aalst. Genetic process mining: A basic approach and its challenges. In *Business Process Management Workshops*, pages 203–215, 2005.

A. K. Alves de Medeiros. *Genetic Process Mining.* PhD thesis, Technische Universiteit Eindhoven, 2006.

A. K. Alves de Medeiros, A. J. Weijters, and W. M. Aalst. Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov.*, 14(2):245–304, 2007.

A. K. Alves de Medeiros, A. J. M. M. Weijters, and W.M.P. van der Aalst. Using genetic algorithms to mine process models: Representation, operators and results. Technical report, Eindhoven University of Technology, 2004.

Francesco Moscato, Nicola Mazzocca, Valeria Vittorini, Giusy Di Lorenzo, Paola Mosca, and Massimo Magaldi. Workflow pattern analysis in web services orchestration: The BPEL4WS example. In *HPCC*, pages 395–400, 2005.

John Novatnack and Jana Koehler. Using patterns in the design of inter-organizational systems - an experience report. In *OTM Workshops*, pages 444–455, 2004.

Mike P. Papazoglou and Willem J. Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, July 2007.

Gabriel Pedraza and Jacky Estublier. Distributed orchestration versus choreography: The focas approach. In Qing Wang, Vahid Garousi, Raymond J. Madachy, and Dietmar Pfahl, editors, *ICSP*, volume 5543 of *Lecture Notes in Computer Science*, pages 75–86. Springer, 2009.

Chris Peltz. Web services orchestration. a review of emerging technologies, tools and standards. *Hewlett Packard White Paper*, January 2003a.

Chris Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, 2003b.

Maja Pusnik, Matjaz B. Juric, Marjan Hericko, Bostjan Sumak, and Ivan Rozman. Business process orchestration and ebusiness. In *Proceedings of the 16th Bled eCommerce Conference, Bled, Slovenia, June 9-11*, 2003.

Stephen Ross-Talbot. Orchestration and choreography: Standards, tools and technologies for distributed workflows. In *NETTAB Workshop - Workflows management: new abilities for the biological information overflow*, 2005.

A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.

Mehmet Sayal, Fabio Casati, Umesh Dayal, and Ming chien Shan. Integrating workflow management systems with business-to-business interaction standards. In *In Proceedings of the 18th International Conference on Data Engineering*, pages 287–296, 2002.

Alexander Sterff. Analysis of service-oriented architectures from a business and an IT perspective. Master's thesis, Technische Universitat Munchen, 2006.

José Tribolet. *Sistemas de Informação Organizacionais*, chapter Organizações, Pessoas, Processos e Conhecimento: Da Reificação do Ser Humano como Componente do Conhecimento à "Consciência de Si" Organizacional (in portuguese), pages 433–454. Edições Sílabo, November 2005.

Bruno Wassermann, Wolfgang Emmerich, Howard Foster, and Liang Chen. Web service orchestration with BPEL. *28th International Conference on Software Engineering (ICSE'06)*, 0:1071–1072, 2006.

A.J.M.M. Weijters and W.M.P van der Aalst. Process mining: Discovering workflow models from event-based data. In *CAI Workshop on Knowledge Discovery and Spatial Data*, pages 78–84, 2002.

A.J.M.M. Weijters and W.M.P van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, 10:151–162, 2003.

Lijie Wen, Jianmin Wang, Zhe Wang, and Jiaguang Sun. A novel approach for process mining based on event types. Technical report, Eindhoven University of Technology, 2004.

Ivy Xiying Zhang. Economic consequences of the Sarbanes-Oxley act of 2002. *Journal of Accounting and Economics*, 44(1-2):74–115, September 2007.