Automatic extraction of communication protocols for web services composition

Kreshnik Musaraj

N° d'ordre |2|8|8|-|2|0|1|0| Année 2010

THESE DE L'UNIVERSITE DE LYON
ECOLE DOCTORALE INFORMATIQUE ET MATHEMATIQUES

délivrée par

L'UNIVERSITE CLAUDE BERNARD LYON I

préparée au

LABORATOIRE LIRIS UMR 5205 CNRS

pour l'obtention du

DIPLOME DE DOCTORAT
(arrêté du 7 août 2006)

soutenue publiquement le 14 Décembre 2010

par

Kreshnik MUSARAJ

Extraction automatique de protocoles de communication pour la composition de services Web

Directeur de thèse : **Mohand-Said HACID**

devant la commission d'examen :

M Jérôme GENSEL	Professeur Université Pierre Mendès	*rapporteur*
M. Yamine AIT-AMEUR	Professeur ENSMA	*rapporteur*
Mme Karine ZEITOUNI	Professeur Université de Versailles	*examinatrice*
M. Omar BOUCELMA	Professeur Université Marseille 3	*examinateur*
M. Mohand-Said HACID	Professeur Université Lyon 1	*directeur de thèse*
M. Fabien DE MARCHI	Maître de Conférences Univ. Lyon 1	co-*directeur de thèse*

AUTOMATIC EXTRACTION OF COMMUNICATION PROTOCOLS FOR WEB SERVICES COMPOSITION

By

Kreshnik MUSARAJ

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
UNIVERSITY CLAUDE BERNARD LYON 1
VILLEURBANNE, FRANCE
NOVEMBER 2010

RESUME : La gestion des processus-métiers, des architectures orientées-services et leur retro-ingénierie s'appuie fortement sur l'extraction des protocoles-metier des services Web et des modèles des processus-métiers à partir de fichiers de journaux. La fouille et l'extraction de ces modèles visent la (re)découverte du comportement d'un modèle mis en œuvre lors de son exécution en utilisant uniquement les traces d'activité, ne faisant usage d'aucune information a priori sur le modèle cible.

Notre étude préliminaire montre que: (i) une minorité de données sur l'interaction sont enregistrées par le processus et les architectures de services, (ii) un nombre limité de méthodes d'extraction découvrent ce modèle sans connaître ni les instances positives du protocole, ni l'information pour les déduire, et (iii) les approches actuelles se basent sur des hypothèses restrictives que seule une fraction des services Web issus du monde réel satisfont. Rendre possible l'extraction de ces modèles d'interaction des journaux d'activité, en se basant sur des hypothèses réalistes nécessite: (i) des approches qui font abstraction du contexte de l'entreprise afin de permettre une utilisation élargie et générique, et (ii) des outils pour évaluer le résultat de la fouille à travers la mise en œuvre du cycle de vie des modèles découverts de services. En outre, puisque les journaux d'interaction sont souvent incomplets, comportent des erreurs et de l'information incertaine, alors les approches d'extraction proposées dans cette thèse doivent être capables de traiter ces imperfections correctement.

Nous proposons un ensemble de modèles mathématiques qui englobent les différents aspects de la fouille des protocoles-métiers. Les approches d'extraction que nous présentons, issues de l'algèbre linéaire, nous permettent d'extraire le protocole-métier tout en fusionnant les étapes classiques de la fouille des processus-métiers. D'autre part, notre représentation du protocole basée sur des séries temporelles des variations de densité de flux permet de récupérer l'ordre temporel de l'exécution des événements et des messages dans un processus. En outre, nous proposons la définition des expirations propres pour identifier les transitions temporisées, et fournissons une méthode pour les extraire en dépit de leur propriété d'être invisible dans les journaux. Finalement, nous présentons un cadre multitâche visant à soutenir toutes les étapes du cycle de vie des workflow de processus et des protocoles, allant de la conception à l'optimisation.

Les approches présentées dans ce manuscrit ont été implantées dans des outils de prototypage, et validées expérimentalement sur des ensembles de données et des modèles de processus et de services Web. Le protocole-métier découvert, peut ensuite être utilisé pour effectuer une multitude de tâches dans une organisation ou une entreprise.

MOTS-CLES : fouille de données, analyse de journaux d'interaction, inférence de modèles, extraction de connaissances, protocole-métier, workflow, service Web.

DISCIPLINE : Informatique

INTITULE ET ADRESSE DE L'U.F.R. OU DU LABORATOIRE : Laboratoire d'Informatique en Image et Systèmes d'information, LIRIS UMR CNRS 5205, UFR Sciences et Technologies, Bâtiment Nautibus, 8 boulevard Niels Bohr, 69100 Villeurbanne

TITLE : AUTOMATIC EXTRACTION OF COMMUNICATION PROTOCOLS FOR WEB SERVICES COMPOSITION

ABSTRACT: Business process management, service-oriented architectures and their reverse engineering heavily rely on the fundamental endeavor of mining business process models and Web service business protocols from log files. Model extraction and mining aim at the (re)discovery of the behavior of a running model implementation using solely its interaction and activity traces, and no a priori information on the target model.

Our preliminary study shows that: (i) a minority of interaction data is recorded by process and service-aware architectures, (ii) a limited number of methods achieve model extraction without knowledge of either positive process and protocol instances or the information to infer them, and (iii) the existing approaches rely on restrictive assumptions that only a fraction of real-world Web services satisfy. Enabling the extraction of these interaction models from activity logs based on realistic hypothesis necessitates: (i) approaches that make abstraction of the business context in order to allow their extended and generic usage, and (ii) tools for assessing the mining result through implementation of the process and service life-cycle. Moreover, since interaction logs are often incomplete, uncertain and contain errors, then mining approaches proposed in this work need to be capable of handling these imperfections properly.

We propose a set of mathematical models that encompass the different aspects of process and protocol mining. The extraction approaches that we present, issued from linear algebra, allow us to extract the business protocol while merging the classic process mining stages. On the other hand, our protocol representation based on time series of flow density variations makes it possible to recover the temporal order of execution of events and messages in the process. In addition, we propose the concept of proper timeouts to refer to timed transitions, and provide a method for extracting them despite their property of being invisible in logs. In the end, we present a multitask framework aimed at supporting all the steps of the process workflow and business protocol life-cycle from design to optimization.

The approaches presented in this manuscript have been implemented in prototype tools, and experimentally validated on scalable datasets and real-world process and web service models. The discovered business protocols, can thus be used to perform a multitude of tasks in an organization or enterprise.

KEYWORDS: data mining, interaction log analysis, model inference, knowledge extraction, business protocol, workflow, Web service

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank Professor Mohand-Said Hacid, my supervisor, for his many suggestions and constant support during this research.

I would like to thank Professor Fabio Casati and Dr. Florian Daniel for their help during my stay as a Visiting PhD in the University of Trento. I did enjoy working with them. My work at the DISI laboratory was extremely productive, and this would not have been possible without their contribution. Special thanks go to Didier Devaurs for his important contribution to this research work, and to Tetsuya Yoshida for his ideas and fertile research collaboration efforts.

Professor Boualem Benatallah expressed his interest in my work and supplied me with the preprints of some of his recent joint work with Dr. Hamid Motahari, which gave me a better perspective on my own results. I am also thankful to Professor Alexandre Aussem for his guidance through the early months of my research work. Cinzia Cappiello also collaborated and shared with me her knowledge on data quality issues and uncertainty analysis and provided many useful references and friendly encouragement.

I also thank the Rhone-Alpes Region. The *Explora'Doc Scholarship*, which was awarded to me for the period 2009–2010, was crucial to the successful completion of this project. This research work is part of the EU Framework 7 STREP project COMPAS (215175, FP7-ICT-2007-1)[1].

Of course, I am grateful to Sidonie and my parents for their support and love.

Lyon, France — Kreshnik Musaraj
October 15, 2010

[1] http://www.compas-ict.eu/

Chapter 1

Introduction

The design, maintenance, and management of Web services and business processes has historically consumed important human and IT resources. This trend still continues today, despite all the successful efforts to improve the three underlying activities (design, maintenance, and management). The work towards the enhancement of business processes started with the quest for their automation [29, 141]. Process automation attempts to decrease human involvement in process management tasks by employing the integration of systems and an automated execution of the business logic. The common trend of process enhancement moved towards the necessity to analyze and integrate process models, adapt them to specific and generic requirements, and manage their complexity in terms of efficiency and readability of the process model [29, 54, 141].

Web services have emerged recently as the most suitable solution for business process integration, external exposure, and implementation. Web services rely on standards in order to achieve their objectives. Modeling and analyzing the behavior of Web services, implementing and improving business processes, and their mutual integration are hard-impact factors of success for companies and organizations [53,54]. Proof for this is the important number of tools aiming at the analysis of process and service executions (we can mention for example the tools introduced in [46, 73]). This is also backed up by research work in business process and service execution monitoring [14,30], process discovery [56,144,148] and service interoperability analysis [34, 106, 118, 136]. The widespread usage of Web services relies upon the capability of reducing development and integration costs, and to increase business agility [101]. In addition, they enable the connection between process components without being influenced by platform and language heterogeneity, as well as boundaries between enterprises and organizations.

In order to expose the existing achievements and the features they lack, it is important to explore the approaches, techniques and tools in enterprise systems.

This chapter is organized as follows. In Section 1.1, we describe how Web services and their composition are currently considered in regards to the implementation and integration of business processes. In Section 1.2, we discuss the research issues addressed in this dissertation. In Section 1.3, we summarize our contributions. We conclude by presenting the organization of this manuscript in Section 1.4.

1.1 Business Processes and Web services

1.1.1 Business processes

A business process is a *"structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focusing emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. ... Processes are the structure by which an organization does what is necessary to produce value for its customers"* [42]. In other words, a business process is a *"set of coordinated tasks and activities, carried out manually or automatically, to achieve a business objective or goal"* [66, 86, 91, 141, 142]. In the context of today's Web services, a business process specifies *"the potential execution order of operations from a collection of Web services, the data shared between these Web services, which partners are involved and how they are involved in the business process, joint exception handling for collections of Web services, and other issues involving how multiple services and organizations participate"* [90].

As [105] correctly points out, business processes can be divided into different types based on the considered criteria. For example, the separation given in [6, 112] is:

- public business processes, which are those that an enterprise shares with its business partners (e.g., clients and suppliers). This type of business processes is used in the business-to-business integration (B2Bi) context [26],

- private business processes, which are internal to the enterprise. This type of business processes is used in enterprise application integration (EAI) context [6]. In any enterprise, both public and private business processes are used together to perform the overall operations of the business. The main difference between the two is that private business processes include execution details that are not present in the public business processes such as how a purchase order is actually processed by various enterprise applications.

Another distinction can be made between:

- Management processes, the processes that govern the operation of a system. Typical management processes include "Corporate Governance" and "Strategic Management".
- Operational processes, processes that constitute the core business and create the primary value stream. Typical operational processes are Purchasing, Manufacturing, Marketing and Sales, and
- Supporting processes, which support the core processes. Examples include Accounting, Recruitment, Technical support.

Achieving the design, maintenance, and management of business processes, we will need to recall their supporting theories and concepts.

Business process management (BPM).
Business process management is a management approach intended to align all aspects of an organization with the needs of clients. BPM is defined also as being the task of *"supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information"* [143]. It is *"a total management approach that promotes business effectiveness and efficiency while striving for innovation, flexibility, and integration with technology"* [134]. Business process management attempts to improve processes continuously and could therefore be described as a "process optimization process". The Association of Business Process Management Professionals (ABPMP) [1] launched a comprehensive body of knowledge and a professional certification for BPM Professionals in 2007. The ABPMP definition of Business Process Management is:

"Business Process Management is a disciplined approach to identify, design, execute, document, monitor, control, and measure both automated and non-automated business processes to achieve consistent, targeted results consistent with an organization's strategic goals. BPM involves the deliberate, collaborative and increasingly technology-aided definition, improvement, innovation, and management of end-to-end business processes that drive business results, create value, and enable an organization to meet its business objectives with more agility" [1].

The life-cycle of BPM is divided into categories that vary according to the different sources. In [141], the business process management life-cycle is divided into four phases: design, process-aware system configuration, process enactment, and diagnosis. In the design phase, the process is (re-)designed and modeled. In the configuration phase, a process-aware system, e.g., a workflow management system, is configured and in the process enactment phase the operational business process is executed. Finally,

in the diagnoses phase the process is monitored, analyzed, and process improvement approaches are proposed. According to [134], the same life-cycle can be grouped into five categories: design, modeling, execution, monitoring, and optimization. Business process design aims at correctly identifying existing processes coupled with the design of future processes that will be added. The modeling step takes as input the theoretical design in order to provide the functional version of the process. This functional version is enriched with the results of "what-if" analysis on the considered processes. Executing the process comes either in the form of software that simulate the process, or as a definition of the process in a particular language. Moreover, business rules and business rule engines [47,146] are also employed to run the process execution and termination. Process monitoring handles the tracking of individual processes in order to extract activity and execution information. Process mining, sometimes named offline or post-mortem monitoring, is also considered to be part of this stage, and analyzes the event logs instead of the runtime process instances. Process mining is often used for delta analysis, and for many other objectives that will be detailed in Chapter 2. The optimization step handles the analysis of the collected information for the identification of bottlenecks and other errors that need to be corrected or improved. The result of the analysis is then exploited in the design model.

Workflow management system (WfMS).

An intuitive definition of a Workflow Management System would be: *"A computer system that manages and defines a series of tasks within an organisation to produce a final result"* [95]. WfMS allows the definition and design of different workflows adapted to the target objective or process. For example, in a hospital, the medical information regarding a patient might be automatically transmitted from the doctor to a surgeon, in case a surgery operation is required. The workflow precisely defines at each given step, what is the corresponding task to be fulfilled, and which data or resources are transmitted to the next task. Workflow management systems provide a means for the automatization of processes since they literally control the flow of tasks and their order of execution and termination. In the preceding example, if the patient has not undergone all the required medical checks and analysis, then the doctor waits for the notification of their completion. In other words, the tasks are dependent from one another and consequently the WfMS reflects the dependencies required for the completion of each task.

A workflow is defined by the Workflow Management Coalition (WfMC) [95] as:

"... the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules" [74, 95]. It is up to the WfMS to handle the

automated execution of a workflow. The same source defines a WfMS as:

"...a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants, and if necessary, invoke the use of IT tools and applications" [74, 95].

Business process management system (BPMS)

Many vendors have created application suites which enable organizations to better manage their business processes. These technologies typically involve tools to visually design and model business processes, simulate and test business processes, automate, control and measure business processes, and provide feedback and reporting on process performance. Some vendors have combined these functions into business process management suites that provide a complete integrated BPM platform, commonly referred to as a BPMS.

Numerous organizations have a large number of legacy systems, typically designed to support specific functions such as manufacturing or sales. In order to manage the end-to-end work involved in business processes, a BPMS must be able to integrate with legacy systems across the organization in order to control work, get information or measure performance. As we have already pointed out, by leveraging web services in a service oriented architecture construct, organizations can build and manage end-to-end business processes across organizational silos and their legacy systems. Many modern BPM technology solutions include the capability to interface to legacy systems through these standard interfaces, providing the tools to automate and orchestrate work across the entire organization.

As BPM technologies were released into the market, a large number of IT organizations have begun to recognize that the technology can be leveraged to develop applications faster and at a lower cost than traditional methods. The visual design capabilities and standards-based interfaces create an environment where skilled software engineers can quickly define the behavior of software which traditionally requires significant effort to be developed.

1.1.2 Web services

"Web Services are self-contained, loosely-coupled modular business process applications that can be integrated with other services within and across enterprise boundaries to develop value-added applications" [9, 114]. Web services are based on the industry standard technologies of WSDL (service description), UDDI (service advertisement and syndication), and SOAP (service-client communication). A Web service

is a Web accessible application, identified with URIs, that supports direct interactions with other software applications using XML-based messages via Internet-based protocols [9]. In the following, we give an overview of how Web services are related to business processes and in which way this relationship generates not only solutions but also research issues that need to be tackled.

Web service standards

Let us introduce the main standards that are relevant to Web services.

- Simple Object Access Protocol (SOAP).

SOAP is *"a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics."* [24, 102]. SOAP is intended to simplify the deployment of services by omitting many of the features encountered in messaging frameworks of distributed systems. Examples include "reliability", "security", "correlation", "routing", and "Message Exchange Patterns" (MEPs). The SOAP Version 1.2 specification consists in three parts. Part 1 of the SOAP Version 1.2 specification defines the SOAP messaging framework consisting of:

- The SOAP processing model defining the rules for processing a SOAP message.
- The SOAP Extensibility model defining the concepts of SOAP features and SOAP modules.
- The SOAP underlying protocol binding framework describing the rules for defining a binding to an underlying protocol that can be used for exchanging SOAP messages between SOAP nodes.
- The SOAP message construct defining the structure of a SOAP message.

- Representational State Transfer (REST)

The Representational State Transfer concept definition was introduced and defined in [65].

"The World Wide Web architecture has evolved into a novel architectural style that I call 'representational state transfer.' Using elements of the client/server, pipe-and-filter, and distributed objects paradigms, this style optimises the network transfer of representations of a resource. A Web-based application can be viewed as a dynamic graph of state representations (pages) and the potential transitions (links) between states. The result is an architecture that separates server implementation from

the client's perception of resources, scales well with large numbers of clients, enables transfer of data in streams of unlimited size and type, supports intermediaries (proxies and gateways) as data transformation and caching components, and concentrates the application state within the user agent components" [65].

The REST approach, when applied to Web services, presents many positive aspects. The technologies and methods upon which REST relies have been extensively used and widely implemented. Since it uses directly the HTTP methods, it does not add another layer. The main problem with HTTP messaging protocols and formats such as SOAP and XML-RPC is that they add a supplementary burden by adding another layer of abstraction onto HTTP rather than using the protocol as it was designed. REST is centered around resources, rather than methods and functions. This means that, provided an URI, all users know how to use it. In the case of Web services, an additional information is required in order for a client to communicate with a service of interest. This led to the following standard.

- Web Services Description Language (WSDL)

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is designed to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

The W3C consortium defines gives in [33] the following description for WSDL: *A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations...*

The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. A WSDL document uses the following elements in the definition of network services:

- Types, represented by a container for data type definitions using some type system (such as XSD)
- Message, an abstract, typed definition of the data being communicated,
- Operation, an abstract description of an action supported by the service,
- Port Type, an abstract set of operations supported by one or more endpoints,

- Binding, a concrete protocol and data format specification for a particular port type,
- Port, a single endpoint defined as a combination of a binding and a network address,
- Service, a collection of related endpoints.

In this dissertation, the main importance of WSDL lies in the the fact that it provides the entire set of message types and operations employed in a Web service. As we will see in the following chapters, this is one of the two main blocks upon which our discovery methods rely, the second one being their timestamps.

Interactions of Web services

Interoperability schemas. As reported in [11], standardization is a fundamental part upon which rely the interactions of Web services. Web services standardization initiatives such as SOAP and WSDL, as well as the family of WS-* specifications (e.g., WS-Policy, WS-Security, WS-Coordination) aim at ensuring interoperability between services developed using competing platforms. Figure 1.1 [11] provides an overview of the hierarchy of Web services standards.

The information on how to interact with the services, i.e., what is the valid order of operations invocation is visualized as a model, called interaction model [6,19,22,40]. The interaction model provides what an interface definition of the service cannot do, i.e. the potential behavior of the service during its interactions with other entities. Basically, there are three interoperability scenarios for Web services.

1. Client interaction

As we have already pointed out, during the execution of a Web service instance, the client can communicate with the service by exchanging messages. Messages need to follow one of the allowed sequences in order to obtain a correct communication. This method is similar to a dialog, thus the possible sequences of messages that a client may employ to communicate with a Web service is called a conversation. The set of conversations are merged into a general model that describes the behavior of the Web service with its external clients. In the case of Web services, this behavioral model is called a business protocol [19,51] (Figure 1.2). The business protocol defines the order in which the operations of a Web service can be executed.

2. Service orchestration

According to [117], orchestration refers to an executable business process that can interact with both internal and external Web services. Orchestration always represents control from one party's perspective and is associated with the private business

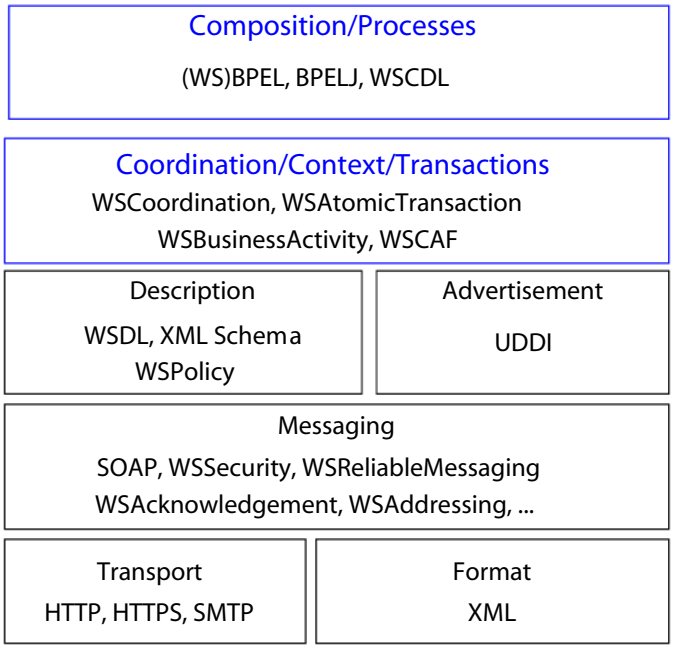


Figure 1.1: Hierarchy of the Web services stack [11].

process of a service. In the enterprise-related context, the term business service is used to refer to a service that uses (composes) a set of other services to implement the private business process of the enterprise. The orchestration model, in this case, is equivalent to the private business process model of the enterprise. Orchestration models could be expressed using languages such as BPEL (as an executable BPEL process) [55, 117] or formal approaches such as Petri nets and process algebra [27, 72].

3. Service composition and choreography

User queries could include both functional and non-functional requirements, and may not be fulfilled by a single Web service. Hence, there is a need to combine a number of suitable services to create a composite service that meets the requirements of the user. This implies the need to establish composability of the component services before the service composition can be instantiated. The core research activities within this area include:

- Design and implement efficient algorithms for service composability based on both functional and non-functional criteria,
- Develop algorithms and solutions for service composition to enable automatic, on-demand and customized composition of services,

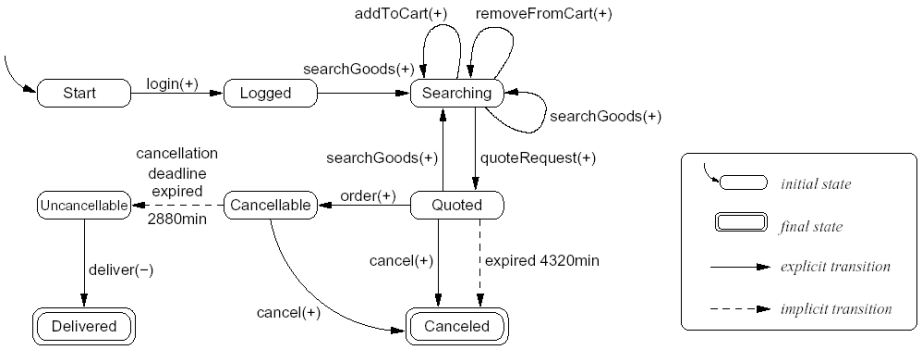


Figure 1.2: A business protocol determines the execution order of operations in their allowed sequences of messages. This model is mandatory for correctly terminating the Web service from all software or human clients [18].

- Evaluate the performance of the algorithms and architectures for Quality of Service based service composition.

The different types of abstraction levels in which service composition takes place are shown in Figure 1.3 [31].

Service choreography on the other hand is a form of service composition in which the message-based interactions between partner services is defined from a global perspective (PO-JRA-2.2.1 [48, 120]). In other words, the term defines the collaborative processes involving multiple services where the interactions between these services are considered from a global perspective [11].

The concept of service choreography can also be seen as an approach that allows each involved party to describe its part in the interaction [117], and a more collaborative approach compared to service orchestration. Service choreographies can be specified using graphical notations such as BPMN [79, 111] and Let's Dance [152], or using XML-based language such as WS-CDL (Web Services Choreography Description Language) [84], and WSCI (Web Service Choreography Interface) [8] and BPEL4Chor [49]. As reported in [105], these standards can also be employed for expressing formal approaches such as process algebra [25, 117].

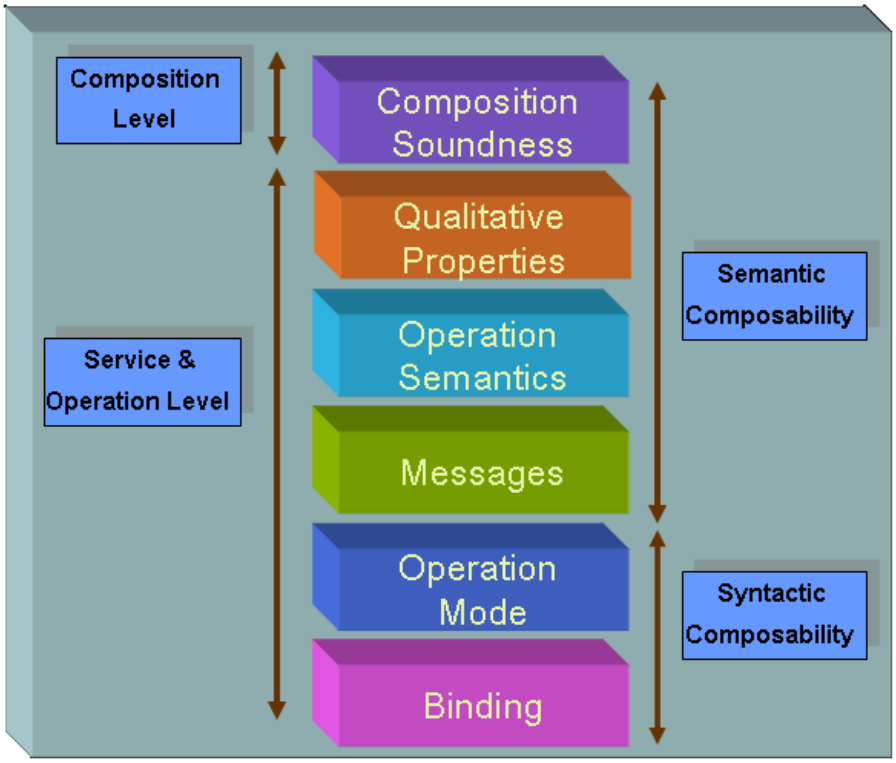


Figure 1.3: Layer-notation schematics of Web services composition [31].

Business protocol

A business protocol is the specification of all possible conversations that a service can have with its partners [6, 19]. A conversation consists of a sequence of messages exchanged between two or more services to achieve a certain goal, for example to order and pay for goods. Modeling business protocols brings several benefits to Web services:

- it provides developers with information on how to program clients that can correctly interact with a service;
- it allows the middleware to verify that conversations are carried on in accordance with the protocol specifications, and hence relieving developers from implementing the exception handling logic;
- it allows the middleware to check if a service is compatible (can interact) with another or if it conforms to a certain standard, thereby supporting both service

development and binding [20];

- it provides the basis for monitoring and analyzing conversations, as the availability of a model can greatly facilitate the exploration and visualization of conversation logs (logs storing messages exchanged among services).

We adopt the deterministic final state machines (FSM) as a formalism for protocols [19]. It supports the constructs that are required to model biparty interactions between a Web service and a client, i.e., the sequence and branching. In addition, it is a simple and easy formalism to use and understand for non-expert users. Furthermore, it is suitable for modeling reactive behaviors.

The formalism has the notion of state, which makes it a useful tool for monitoring service interactions. In general, many of the concepts presented in this dissertation apply regardless of the protocol formalism being adopted. It should be noted that constructs such as parallelism and synchronization are not needed in the bi-party interactions between two services, and the adopted model (FSM) does not support them, as well. In a FSM-based protocol model, states represent the different phases through which a service may go during its interactions with a client (i.e., during a conversation). Each state is labeled with a logical name, such as Logged In. A protocol has one initial state and a set of final states. Transitions are labeled with message names, with the semantics that the exchange of a message (with the conversation in a given state) causes the state transition to occur. A complete conversation corresponds then to a possible instantiation of the protocol model, i.e., a path (set of transitions) in the FSM resulting in a sequence of messages from the initial state to a final state. Hence, such FSM describes the possible conversations supported by the service. A business protocol can be expressed in standard languages, e.g., as an abstract BPEL (Business Process Execution Language) process [55, 83, 117]. It can also be modelled using graphical languages with formal foundations such as state machines [19], Information Control Nets [61, 123], state charts and other formalisms [22, 51]. An example of a realistic business protocol is provided in Figure 1.4. We can formally define a business protocol, using finite state machines formalism, as follows [18]:

Definition 1 Business protocol (BP)
A **business protocol** [18] is a tuple $P = (S, s0, \mathcal{F}, M, \mathcal{R})$ which consists of the following elements:
1. S is a finite set of states.
2. $s_0 \in S$ is the initial state.
3. $\mathcal{F} \subseteq S$ is a set of final states. If $\mathcal{F} = \emptyset$, then P is said to be an empty protocol.
4. M is a finite set of messages. For each message $m \in M$, a function $Polarity(P, m)$ is defined which is positive (+) if m is an input message in P and negative (-) if m is an output message in P. The notation m(+) (respectively, m(-)) is used to denote

the polarity of a message m.
5. a finite set $\mathcal{R} \subseteq S^2 \times M$ of transitions. Each transition (s, s', m) identifies a source state s, a target state s' and either an input or an output message m that is either consumed or produced during this transition.

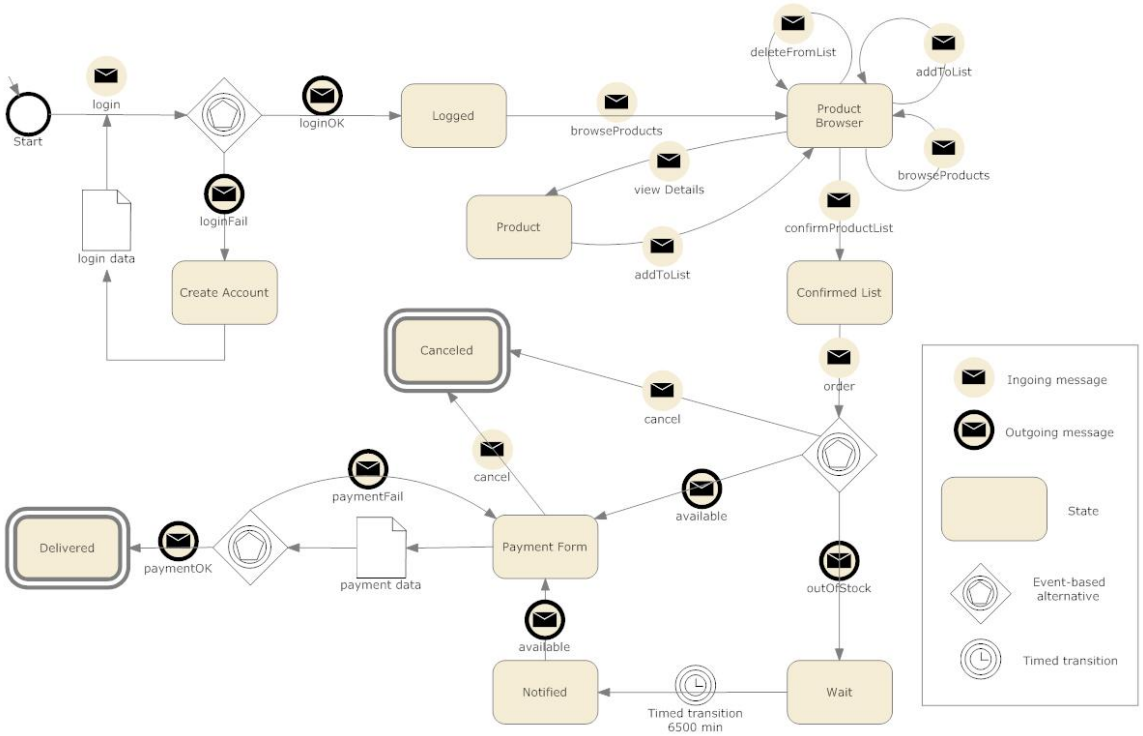


Figure 1.4: The business protocol of the *OnlineTrading* Web service.

Interactions of services and business processes

Figure 1.5 illustrates the two most common interaction scenarios between Web services and business protocols. In the first scenario business processes make use of Web services for their implementation and integration. In the second scenario, business processes employ Web services to externalize their functionalities to other services and/or to exploit existing services from the outside world in order to achieve particular objectives. The figure shows the generic separation into layers, namely the *Resource*, *Internal Service*, *Business Process*, and *External Service* layers. The *Resource* layer includes various information systems including legacy systems, workflows and enterprise resource planning (ERP). In this example, Web services have been used to:

- encompass existing applications as services and also to develop new applications required for the implementation of the business process,
- provide an external interface with users and applications outside the enterprise or organization, and
- allow the consumption of third-party Web services either for particular subcontracted tasks or for integration with an external business process.

Services in both the *Internal* and *External* layers are called component services. Component services may be composed with each other to form business services [21, 58]. In this dissertation, we use the terms "event" and "message", interchangeably, to refer to both meta-data and actual content of messages. A process instance refers to an execution of a process, which involves performing a set of the process tasks, to achieve a business goal. A process instance is also called a conversation in the context of Web services. Correlation allows understanding which events belong to the same process instance. Current business process automation approaches, e.g., workflow management systems (WfMS), only support operational business processes, that is, business processes that are explicitly defined [141].

Web services for high-priority targets of business process

Since Web services are active parts of the steps of a process life-cycle, it becomes obvious that services also need to be submitted to the same cycle in order to prove that they are suitable for the expected functionalities. Yet to achieve these steps, abstract specification information is required. Despite the existing standard specifications for integrating Web services at lower levels of abstractions, i.e., messaging, where many of the issues have already been identified or even solved [89, 114], Web services still lack specifications and standards aimed at higher levels of abstraction, i.e., service interfaces, business protocols, and also policies. In fact, what is provided at the higher levels of abstractions are languages to define the service specifications, i.e., its interface, business protocol, and policies. Web service interfaces are usually described using WSDL (Web Service Definition Language) [124]. If the business protocol specification is provided, it can be expressed using standard languages such as BPEL (Business Process Execution Language) [55] or a modelling diagram such as state diagrams [19, 22, 51].

It is highlighted in [18] that descriptions like WSDL are not sufficient for a sophisticated and automatic use of Web services, because they provide information only about static properties. This is one of the main reasons that motivated authors in [18] to define a higher level model, the so-called *business protocol*, which specifies the external behaviour dynamics of a service (such as allowed conversations or temporal constraints, c.f. Section 2.1). Since it is formalized by a finite-state machine,

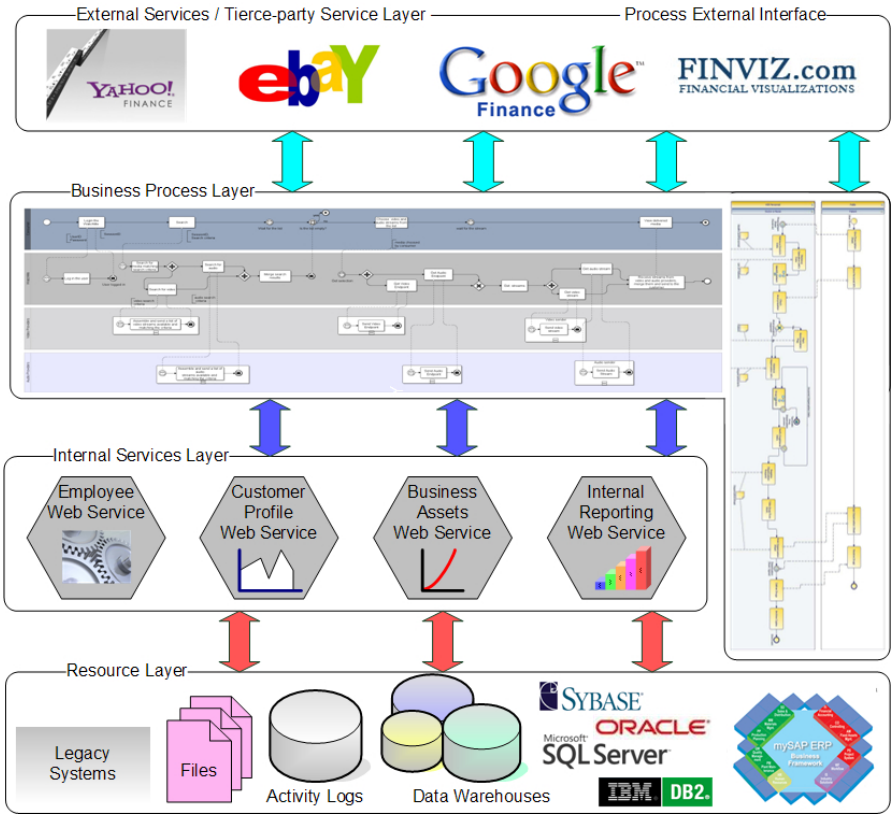


Figure 1.5: Illustration of the two most common interaction scenarios between Web services and business protocols. In the first scenario (blue arrows) business processes make use of internal Web services for their implementation and integration. In the second scenario (magenta arrows), business processes employ Web services to provide external functionalities as input and output.

the business protocol offers automatic reasoning mechanisms for many applications, such as correctness verification, compatibility testing, etc. Nevertheless, the business protocol is not often specified in real life services. The reasons include lack of time during implementation, and uncontrolled service evolution (making the initial model obsolete). A possible solution consists in deriving the business protocol from the *conversation logs* of a service when available. Solving this discovery problem could also be very useful for re-engineering applications, such as service and process implementation correctness checking, or service/process evolution.

Another usage of knowing Web service protocols is their application for business process integration. Web services simplify interoperability and, therefore, application

integration [6, 20, 106, 114] by providing (i) support for loosely coupled and decentralized interactions, and (ii) standardization. Standardization helps reducing the costs of application integration, which are to a large extent due to the fact that interacting services have different interfaces, use different communication protocols, and support different data formats and interaction models. A Web service can be characterized at four levels of abstractions [43, 87, 140]: messaging, which specifies how the service exchanges messages over communication networks, interface which defines the set of operations that the service provides along with message formats and data type definitions, business protocol, which specifies the order in which operations of a service can be invoked, and also policies, which give the details of business rules and technical requirements. Business-to-Business integration [6, 26] requires for two business partners to integrate their services that the services offered by them should be able to seamlessly interoperate at all levels of abstractions. This means both should agree on the communication protocols and the data syntax that is used for message transportation. In addition they should understand the content of exchanged messages in the same way, business documents should arrive in the order they expect, and also the business and enterprise policies and regulations should be respected.

1.2 Key Research Issues

In the previous section we mentioned the need to obtain service and process specifications at a higher level of abstraction than the existing standards allow so far. Moreover, we stated that if the correlation information was not present in logs, then the existing solutions that employ conversation mining [105, 107, 108] are jeopardized. Consequently, this fact calls for an approach that does not rely on conversation mining and thus does not require grammar inference algorithms to extract the state diagrams from conversation logs. Nevertheless, the inverse should remain true i.e. discovering the business protocol or process view must allow in principle to obtain the conversations. In addition, it is clear that employing only a minimal and restricted set of assumptions needs to exploit as much as possible the data available. For example, the usage of timestamps cannot be limited to a mere basic ordering of the messages that compose a conversation.

The first research issue is obviously the characterization of the problems by identifying goals, the assumptions and related scenarios. This needs a "divide to conquer" approach for separating the problems into parts that should be addressed. In the following, we give an overview of the key research issues that we identified in fulfilling above requirements, and which are addressed in this dissertation.

1. Message flow discovery

Starting from the log of events recorded during the execution of a Web service the first task is to determine the distribution of messages flowing through a protocol instance. Let us consider an example in order to clarify the meaning of this step. Given a state of the automata (See Figure 1.4) modeling a business protocol, one must focus on the transitions entering and leaving that state. The set of entering transitions and the set of outgoing transitions are strictly related since modifying the former will have a direct impact on the latter. Our concern in this part is to determine the way this relation takes place, for each single state and for all states combined. This way of approaching the poblem will free us from the obligation to correlate messages into conversations. Nevertheless, this remains a difficult task. Indeed, if one does no longer assume that logs contain the identifiers required for correlating messages, then there is absolutely no indication on how messages follow each other during the execution of a service instance. Timestamps become entirely useless because simple ordering alone based on occurrence time is not sufficient to determine the exact sequence of message occurrence. Let us consider for example a set of messages and their corresponding timestamps without any additional information: $(a, 15)$, $(b, 23)$, $(c, 25)$, $(d, 30)$. If we use a simple order based on these timestamps, we face a set of possible message arrangements that do not contradict this order: (i) a, b, c, d, (ii) a, c and b, d, (iii) a, b, c and d, (iv) a and b, c, d etc. Thus, it is obvious that solving the problem is not straightforward. Therefore, assessing the message flow of a protocol is divided into three stages. First, the message flow dynamics need to be modeled at the protocol state level. At the second stage, the message flow knowledge related to each state of a protocol will be merged in order to obtain a preliminary version of the extracted protocol model. Finally, the result will be evaluated based on the influence of log imperfections and inconsistencies.

2. Mining temporal operators

This issue may be formulated as follows: How to make use, as much as possible, of the timestamps of raw event logs in order to detect the occurrence order of messages? Is it possible to obtain a deterministic model that converges towards the optimal solution? This task relates to the domain of temporal pattern mining [15,75,129,139,150]. As we have already pointed out, timestamps alone cannot be used for trivial temporal ordering because logs represent collections of messages that originate from multiple service instances executing in parallel. The execution of service instances being independent from one instance to another, the timestamps will be without any correlation. A solution to this problem would have been to correlate messages in order to identify the protocol conversations [105,108]. Yet, in the scenario of realistic logs in which the correlation information is entirely absent, conversation mining is no longer possible. In such case, in order to establish the relationships between occurrences of messages,

we need a solution to extract the temporal facts that will allow us to define the required temporal operators and how they relate messages [5].

3. Business protocol inference

Discovering the business protocol of a service includes many technical challenges. First, the model of the extracted protocol has to take into account the uncertainty of the result, and to evaluate its relevance by means of confidence indexes and quality criteria. This uncertainty comes mainly from the fact that service logs can contain some "noise" (i.e. some errors). Thus, tools are needed to analyze and clean data before treating them. Finally, it is also important to propose tools that enable to correct or refine the discovered protocol, according to what the user wants or knows about it. Nevertheless, we look at protocol mining in such a way that most of these issues are either already addressed during the mining process, or irrelevant. Inferring the business protocol from message flow information and temporal operators is a cross-solution that needs to avoid result ambiguity, effects of imperfections, and high uncertainty. We will show how these flaws may be addressed in a satisfactory manner, while using a minimally assumed information content.

4. Extracting timed transitions Up to this point, we have discussed the issues on discovering the protocol whose traces are visible in logs. Now we intend to extract the time-triggered transitions that do not send or receive messages in the process, thus not directly visible in activity logs. Indeed, one of the most difficult problems to tackle is the extraction of temporal constraints called *timed transitions*. This is because they are not explicitly recorded in the logs. Some state changes are not related to the emission of explicit messages but to temporal constraints (validity period, expiration date, *etc*). We recall that in general, transitions are triggered when the service sends or receives messages. A *timed transition* (also called implicit transition) occurs automatically, after a time interval is elapsed since the transition has been enabled (i.e. source state of the transition has become the current state), or after some date is reached; it is labelled by the corresponding time constraint. Note that, since the model is deterministic, a state cannot have several outgoing timed transitions. Detecting timed transitions also requires that the conversations are known in advance. In other words, the un-timed business protocol has already been inferred. This calls for an update of the mined model once that timed transitions have been discovered.

1.3 Contributions

Investigating the issues that are related to the topics described in the precedent sections calls for formal frameworks, techniques, and implementation tools that enable to achieve the set of objectives, as well as to assess and test the conformance and suitability of the results obtained from the discovery process. To summarize, we need mathematical tools for modeling the flow of messages and events in a business protocol, determining the temporal relations between these events, simulating and validating the models, and confirming the set of assumptions upon which rely the different discovery modules.

Besides the characterization and definition of the problems in this area and identifying some research problems, we contributed to the following issues:

1. Correlation of the flow of message occurrences.

We present an approach for correlating the flow densities of messages and extracting the business protocol of a web service in the realistic scenario in which correlation information is entirely absent from interaction and activity logs. Correlation is achieved through deterministic computations that provide a tested solid reliability, robustness when dealing with complex structures, and very high performance and scalability. An implemented algorithm is provided that is capable of recognizing complex structures such as ramifications, simple and composite loops. Moreover, we give a proposal of combining statistical computations in a deterministic result that discovers the class of discoverable behavior models. The algorithm addressing these issues aims at (i) providing the starting point for every single log-based mining method: correctly correlated and instance-sequenced data, and (ii) allowing the easy and immediate reconstruction of the model that visually displays the web service business protocol (WSBP). In addition it has the capability of noise-proof robustness, thus avoiding the necessity and risks of noise-filtering solutions.

2. Analyzing time series of message timestamps.

We present a variable grain-size algorithm that extends the usage of temporal operators, and based on the study of cardinality properties allows the correlation between timeseries. The approach does not operate on any assumption on the existence of extracted facts and is capable of inferring temporal data facts and handling the pre-processing step. This approach allows to obtain a unique graph that models the protocol and acts as an assessing step for the detection of protocol structure that are complex to handle during the mining process. We present an implementation of the approach in a separate tool that can also be integrated with the other algorithms.

3. Mining business protocols and inferring graph models.

We provide an algorithm that employs the results obtained from the extraction of flow density models and the temporal operators between events and messages, in order to build a unified business protocol. This corresponds to the almost-refined log mining result, only to be enriched with the timed transitions, that we discover in the next stage. The main advantage here is the flexibility of the extracted model with regards to the formal model employed to represent the behavior of the business protocol. Indeed, determining the graph-based formalism (Finite-state machine(FSM), Petri nets, Workflow nets [144], Information Control nets [61,123] etc...) for modeling and analyzing the behavioral model is of no use and has no impact on the way our algorithms proceed and the provided results.

4. Addressing issues related to mining non-logged timed transitions.

In the context of the discovery of the timed business protocol of a Web service from its conversation logs, we also have focused on extracting timed transitions. We provide an approach for recognizing and discovering these time-triggered events occurring in a business protocol. We formally define a class of patterns called *proper timeouts* and show that they reveal the presence of timed transitions in the business protocol. Our contribution is based on a formal framework leading to the definition of proper timeouts. We show that proper timeouts are the best representations of timed transitions in conversation logs. We have given a simple characterization of the set of proper timeouts satisfied by the logs. Moreover, we provide a polynomial algorithm that has been implemented for extracting these patterns.

5. Technical issues: implementation and tools.

The previous contributions have been implemented in order to provide a general framework for mining business processes and protocols. Model assessment and model mining merge together, especially regarding the tasks: (i) checking the equivalence between the specification and implementation, and (ii) obtaining the specification if it does not exist. We provided **DOBS** (**D**ynamic m**O**del **B**ehavior **S**imulator), a modeling-and-emulating generator tool which allows the expert of business processes, web services, or any other dynamic behavior-based systems, to design, test and simulate a behavioral model such as a business process or a web service / application. *DOBS* is designed to be used both as a testing and data generation tool, and is the main tool upon which all the algorithms have been tested. This tool and the other implemented algorithms are intended to improve model compliance while helping decision support during all stages of model assessment evaluation and process life-cycle.

1.4 Organization of the manuscript

The reminder of this dissertation is organized as follows. In Chapter 2 we present an overview on the current state of research in this area. In Chapter 3 we present our approach for establishing the correlation between flow densities of messages. In Chapter 4, we present our approach for extracting temporal patterns and relationships. This will provide a first view of the protocol in the form of a temporal graph, and afterwards as a complete business protocol. In Chapter 5 we show how timed transitions will be mined from the conversations of the model discovered in Chapter 4. In Chapter 6, we present the tools for assessing and validating the mining results, generating model-related logs, and assess the overall performances of our approaches. We conclude the manuscript in Chapter 7.

Chapter 2

Log mining synopsis

2.1 Introduction

This chapter provides a relatively comprehensive and updated survey, as well as a taxonomy of the works that seem to be the most appealing in regard of their potential power of discovery and robustness towards imperfections in data. The taxonomy of the survey is divided according to the main types of logs, namely (i) extraction of web service protocols, (ii) workflow and process mining and (iii) pattern mining. A common framework is provided for all the research papers on mining these types of logs. Algorithms and formalized approaches are compared based on an analysis of strong points and weaknesses. The dimensions used for comparing and analyzing the references studied in this survey are chosen to reflect past achievements and future directions. This survey details key papers and presents the novelty of related work. The survey shows that the issues of correlating messages and events, using semantic content, dealing with noise, and addressing privacy problems need to be better resolved in future research.

2.1.1 Overview and motivation

The number of systems that continually produce interaction and operation-related data becomes important. In many cases these systems are composed of web services (WS), workflows, business processes, software architectures, heterogeneous network transactions etc. The trends, patterns, and all the low-level data recorded during the execution of all these systems, hold relevant information that needs to be mined. These mining activities lead unavoidably to business decisions improvements and

many other applications. Moreover, this helps the optimization of the aforementioned systems. Nevertheless, the size of datasets composed of recorded activity is prohibitive for manual analysis and efficient decision-making. Therefore, analysis based on automatic mining methods and algorithms is mandatory for dealing with huge quantities of logs. Knowledge discovery and data mining techniques aim at extracting strategic information hidden in very large datasets. Their usefulness at achieving this objective, and many others, is established.

Knowledge discovery has grown into a very large domain. This includes issues that have emerged from three mainstreams, pattern mining, process workflow mining and the more recent web services protocol discovery. The main goals and the problems which arise during mining processes are often similar. The overall objective is to be able to analyze logs of events or activities. No prior knowledge of the underlying software, process or service is assumed. This analysis must obtain the hidden behavior description structure in the presence of negative-influence factors (noise etc.). The importance of mining and its applications is reflected by the quantity of publications and tools in this field (as seen in the surveys provided in Section 4.1.2). This underlines the need of leading a global investigation inside the fundamental approaches and related methods.

The global issue handled in this survey is the discovery of behavior models from logs. A behavior model is intuitively represented as a diagram of different forms and variations. Examples of the behavior to be modeled are the possible interactions with a web service, a workflow, or a business process. The common framework that unifies all the papers presented in this survey is Data mining (DM) in the domain of Knowledge discovery in databases (KDD). DM is commonly defined as the extraction of patterns or models from observed data. *"Knowledge discovery in databases is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data"* [63]. We also underline that in this survey the terms "mining", "extraction", "discovery", and "inference" have the same meaning. In our work we deal with information recorded inside logs. Some logs come from logging applications of web services and collections of process and workflow data. Other logs are constituted of recorded transactional data (commercial or other). Stored logs contain automatically recorded entries for every detected activity. In this sense, logs are a form of text-like databases. Their attribute-oriented structure makes them very similar to the tables encountered in relational databases. Figure 2.1 illustrates the main types of log data and extracted models that this survey considers. Each row of the table corresponds to a predefined mining context. The left column of each row provides the raw data, and the right column shows the model extracted from that data. This clearly depicts the models that are sought in this state-of-the-art analysis. Moreover, it shows the similarities and differences which motivate our separation into three main directions.

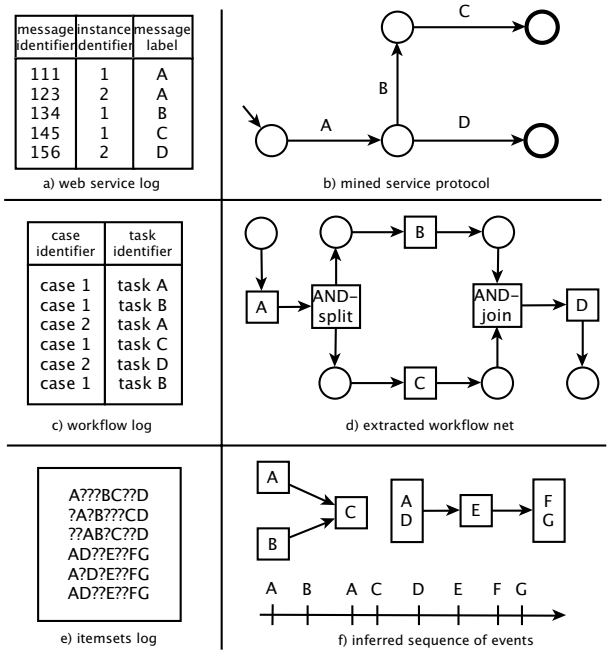


Figure 2.1: Raw data log samples (left column) and mined models from the corresponding logs (right column)

Indeed, in this study, the sub-domain of DM included in the area of KDD is divided into selected directions, from the more recent to the legacy one. These are (i) web service business protocol and interaction extraction, (ii) workflow and process discovery, and (iii) pattern mining (mining of sequential items). This separation into three directions is based upon the origin of the data considered. Data issued from logs of web services is of a different type than the type of data issued from workflow logs, or database commercial patterns. This clusters the approaches in a natural way, according to the criteria of data type. A more detailed presentation of the common framework that unifies the papers in this study into these three directions is provided in the first paragraph in Section 2.2 and Section 2.2.3.
The contributions of this Chapter can be listed as follows:

1. provide an overview of existing approaches and techniques on mining logs of patterns, processes, workflows, and web services.
2. present a taxonomy based on features that classifies knowledge discovery and data mining approaches according to relevant features.
3. survey existing knowledge discovery and data mining approaches based on this taxonomy.

The Chapter is organized as follows: Section 2.1.2 details the topics and categories of log mining that are intentionally not included in this report. Section 2.1.3 provides other existing survey projects and the main reasons that call for a more recent and complete survey. In Section 2.2 we provide the background knowledge and preliminaries required for introducing and analyzing the considered methods and approaches. Section 2.3 explores the backbone problems addressed in web service-related discovery, throughout comparison between existing approaches. Section 2.4 contains a categorized and detailed review of issues and solutions for mining workflows and processes. In Section 2.5 we focus on mining methods for sequential patterns. Finally, we draw the analysis conclusions in Section 2.6, along with future guidelines on open research issues.

2.1.2 Out-of-scope topics

Knowledge extraction from interaction logs is a very large and rich domain. Surveying all the topics related to this field would require a long time. Therefore, the study in this Chapter limits its scope within well defined boundaries. Here we deal with logs generated during the execution of state diagram structures. The main representatives of such structures are workflows, processes, interaction models of web services, etc.

This choice is based on the discovery of behavioral models based on graph representation. There are many other data sets which fall into the log mining domain. Yet, all the data sets that are not issued by graph-based behavioral models are considered to be out of scope for the present study. Such data logs include, but not limited to, genetic data, click streams, web search queries and rare patterns. In conclusion, the only perspective of this survey study is mining data that are associated with the behavior of a given model, viz. behavior and interaction data. Another area that is not addressed in this survey is grammar inference. Very good references on this research field have already been provided in [7, 44].

2.1.3 Other survey studies

Efforts have been made in the past to provide surveys and partial taxonomies. Examples are:
- [62] where the authors provided a survey on mining methods in the context of temporal databases and time-related data behavior.
- In [148], the authors start from the defintion of the workflow mining concept and propose a common format for workflow logs. Several workflow mining approaches are presented.
- In [69] the authors provide a description of the basic frequent itemset and association rule mining problems and the most authoritative algorithms.
- In [99] the authors provide a survey on sequential pattern mining.
- The Data Mine [121] provides a large collection of downloadable papers, tools and a very rich bibliography.
- In [125], the authors present a survey on papers addressing issues of temporal, spatial and spatio-temporal data mining.
- In [68], the authors compare software tools on data mining and knowledge extraction.
- [137] provides a general overview of the state of the art regarding the application of rough set theory in mining processes from data.

From these surveys, two conclusions can be drawn. First, an updated survey is required on knowledge extraction from log textual databases. Second, a study that specifically targets behavior model extraction from activity logs is much more appropriate. The advantage for such a study becomes visible when during the investigation, the underlying taxonomy emerges. This allows to identify the issues to be better addressed in the future.

2.2 Context and Preliminaries

Our taxonomy comes in the form of a hierarchical structure that relies on a level-based categorization. We base our taxonomy on the type of the mining results and on what constitutes the subject of the mining approach. The mining results can be divided based on the extracted models. We may mention Petri nets, workflow nets, finite-state machines, Markov models, and weighted or basic graphs. Clustering of references can also be based upon the mined formal representations. We have for example intervals, order operators, and sets of elements. Moreover, patterns can be divided by splitting them into sequential patterns and temporal patterns. In sequential patterns the appearance in groups of items is sought, while in temporal patterns it is the evolution of given variable values over time that is relevant. Regarding models we distinguish between web services (WS) protocols, and workflow nets. Intuitively, WS protocols designate the messages exchanged during the execution of a WS or the interactions between several WSs. Workflow nets model the sequences of activities and tasks involved during the execution of a workflow or process.

2.2.1 Terminology

The following terms used throughout the survey, are defined in the bibliography of this domain, and we recall their definitions in the following: A *log* (also called a *trace*) is a collection of data entities that are most often recorded by software, or manually entered by humans into computers. The different approaches make assumptions on the type of logs they consider. Types of logs include event logs, conversation logs, databases of itemsets of transactions, timeseries datasets or other variants. An *event* refers to the log entry correponding to different types of activities in a system. This includes different user-generated or automatic operations, or any triggered activity such as tasks, messages etc. A *task* stands for a well-defined step in the workflow. A *message* is the data entity exchanged during the cycles of client request-server response during the execution of web services. Examples of a step would be an activity, or a procedure outcome. A *sequence* is an ordered list of events. An *execution instance* designates a sequence of events leading from the starting point, to the terminal point of a behavioral model. The *behavioral model* is the model providing all the possible correct behaviors that a given system may display (see Section 6.2.2). A *conversation* represents an execution instance in the context of web services. A *transaction* consists of a set of attributes. An *itemset* is a non-empty set of items. The term *time series* is defined as the time course of a set of variables under controlled conditions. A *temporal pattern* is the time interval in which one or more time series assume a behavior of interest. An *occurrence* of an entity represents an instance of that entity. The *occurrence time* of an entity stands for the timestamp (recorded time in the log)

associated to the instance of that entity. *Noise* will designate all the different types of imperfections occurring in logs, whatever their cause and nature. *Uncertainty* represents the notion of confidence attached to different sources of information when different values exist for the same variable or attribute.

2.2.2 Employed models

Here we give the formal definitions of the models that are employed in the approaches described in this chapter.

Definition 2 Deterministic finite state machine (DFSM)
A **deterministic finite state machine** is a tuple $(\Sigma, S, s_0, \delta, F)$, where:
1. Σ is the input alphabet (a finite, non-empty set of symbols),
2. S is a finite, non-empty set of states,
3. s_0 is an initial state, an element of S,
4. δ is the state-transition function,
5. $F \in S$ is the set of final states. A DFSM is mainly employed as the basis for modeling Markov chains and Business protocols. For more information on DFSM please refer to [133].

Definition 3 Markov chain
A **Markov chain** is a sequence of random variables X_1, X_2, X_3, ... with the Markov property, namely that, given the present state, the future and past states are independent. Formally speaking a Markov chain takes the form:
$Pr(X_{n+1} = x|X_1 = x_1, X_2 = x_2..., X_n = x_n) = Pr(X_{n+1} = x|X_n = x_n)$ The possible values of X_i form a countable set S called the state space of the chain. Markov chains are often described by a directed graph, where the edges are labeled by the probabilities of going from one state to the other states. A finite state machine can also be used as a representation of a Markov chain. Additional information on Markov chains can be found in [109].

Definition 4 Workflow nets (WN)
A **workflow net** [144] is a tuple $N = (P, T, F)$ where:
1. P is a finite set of places,
2. T is a finite set of transitions such that $P \cap T = \emptyset$,
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed edges, called the flow relation,
and such that if $\bar{t}$ is a fresh identifier not in $P \cup T$ then N satisifies:
4. Object creation: P contains an input place i such that $\cdot i = \emptyset$,
5. Object completion: P contains an output place o such that $o\cdot = \emptyset$,

6. Connectedness: $\bar{N} = (P, T \cup \bar{t}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ is strongly connected.

Points from 1 to 3 define a Petri net, also called a place/transition net. It is then easy to see that a workflow net is a particular case of a Petri net. This is indicated by points from 4 to 6 in the definition. $\cdot i$ denotes the input nodes of P (i.e. the nodes i such that exists a directed edge starting from i), and $o\cdot$ the output nodes (i.e. the nodes o such that exists a directed edge ending at o). A workflow net is used to model business processes as well as workflows in general.

2.2.3 Classification criteria

The taxonomy used in this survey is based on the event and the data types contained within the logs. When considering event types, we have: workflow tasks, human activities and decisions, software and human operations, and web service messages. The first reasonable separation is thus based on the context of the behavior model that is to be extracted. In this work we deal with three models: web services, workflows and patterns. We recall that the approaches considered in this survey are also clustered based on the model that is used for representing the discovered behavior. Those models are discussed in the previous sections. A distinction between the existing papers can also be made using the stochastic nature of the approach. In this case we separate into probabilistic and deterministic methods. In a probabilistic algorithm the results are bound to probability values and multiple solutions may exist. In deterministic methods a single solution is provided for all discovery attempts.

2.3 Model extraction in service-oriented architectures

Authors in [18] highlight that descriptions like WSDL are not sufficient for a sophisticated and automatic use of Web services (WS). This is because they provide information only about static properties. For this reason, authors in [18] proposed a higher level model, the *business protocol*. This model specifies the external behavior dynamics of a Web service. Nevertheless, the business protocol is not often specified in operational services. Potential reasons include lack of time during implementation, or uncontrolled service evolution.

A solution is to infer the business protocol from the *interaction logs* of a service. Solving this discovery problem could also be very useful for re-engineering applications. These applications include implementation correctness checking, or controlled

service evolution. Business Protocol Discovery (BPD) [108] is a particular case of a more general issue: extracting a model from its instances.

2.3.1 Application on service protocols

Service protocol discovery is not an isolated issue. Model discovery is always part of larger multi-functional service discovery architectures with wide-range objectives. Some examples of these objectives are service compliance, comparison, verification, testing, composition etc. Nevertheless, discovering service business protocols includes many technical challenges. We can mention for example cleaning logs from noisy data, and correlating events and messages into WS instances.

The first contribution to the above problems is proposed in [108]. The authors deal with the problem of discovering protocol models by analyzing real-world interaction logs. One of the achievements of this survey consists in identifying the different kinds of log imperfections. Other contributions are a quantitative measure and an algorithm to determine noisy conversations. An approach is proposed for generating protocols of small sizes by leveraging algorithmic minimization techniques. A user-driven protocol refinement is also made available. This is obtained through the definition of a notion of distance between protocol models and conversations that cannot be generated by the model. Furthermore, the method uses the manipulation operations that can modify these conversations so that they can be accepted. The protocol discovery approach proposed in [108] has two stages: (i) analyzing the log to automatically estimate which conversations are noisy, and (ii) deriving a protocol that can accept conversations estimated to be correct. The protocol derivation part also addresses the problem of non exhaustive logs via discovery heuristics. This work handles most of the problems mentioned in the previous paragraph, but relates only to *untimed* business protocols. With the importance of temporal constraints in real life services (viz. expiration dates), it becomes crucial to extend this discovery technique to *timed business protocols* (TBP). Timed business protocols contain not only explicit transitions (related to a message sending or reception), but also *timed transitions*. Timed transitions represent implicit state changes that are triggered due to temporal constraints.

Based on the elements above, one can identify the main reasons and motivations that led to the discovery of timed transitions. One solution that explicitly deals with this problem is proposed in [52]. Here, the authors use a deterministic approach for mining timed transitions in conversation logs. The challenging issue of timed transitions detection is that no information is recorded when such transitions are triggered. A class of patterns called *proper timeouts* is defined, and it is shown that they are the best representations of timed transitions in the logs. A formal

characterization of the set of proper timeouts satisfied by the logs is also provided. The advantage of the method proposed in [52] is its performance in terms of temporal complexity.

The work in [50] is very similar to [108], despite their different concerns. In [50] the concepts of *interaction* and *interaction protocol* are used instead of *conversation* and *business protocol*. It presents an interoperability problem between a client and services that expose the same interface but have different protocols. The main advantage of the approach in [50] is that it is fully automatic. However, this can be a serious drawback for service re-engineering, where it is essential to allow a user driven refinement of the model [108]. Two other limitations of [50] are: (i) model discovery modules are supposed to be associated by service brokers with all new services, and (ii) noise is not taken into account.

Let us focus our attention on combined approaches that employ finite-state machines (FSM) for modeling tasks and graph-based approaches for event correlation and mining purposes. Indeed, graph-based solutions using FSM modeling were proposed in [107]. This work deals with the discovery of process views. A process view is the equivalent of a database view applied in the context of business processes. During the first step, messages found in logs are correlated together into execution sequences by means of time ordering and correlation conditions. Thus, an undirected correlation graph is built using the set of messages to define vertices. Edges are defined by the correlation conditions discovered between messages. At this stage, a graph decomposition algorithm is run, which returns the maximal connected components of the correlation graph. These components represent sequences of execution instances. The identified sequences are afterwards used as the basis for the reconstruction of the FSM which models the web service behavior in the form of a process view.

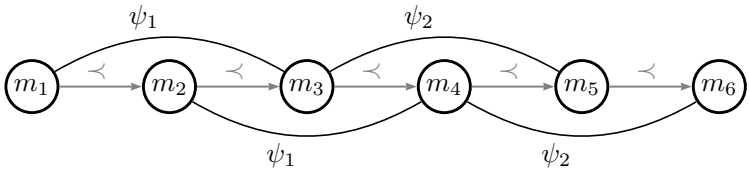
It should be noted that the approach exploits the monotonicity property for enhancing the efficiency. An example of the graph obtained through disjunction of correlation conditions is given in Fig. 2.2 ($\prec$ shows time precedence). Note that m_x and m_y stand for two messages and m_x.OID (respectively m_y.OID) relates to the corresponding attribute of messages whose values are used for the correlation condition.

Following this work, we encounter the discovery method of [130], where the authors model logs using graphs and employ graph theory techniques to extract the conversations and build the Business Protocol. For converting the initial message graph into a reduced one, the authors use the Dempster-Shafers mathematical theory of evidence [131]. However, a more detailed analysis reveals that the assumptions which allow to determine the first message of the service protocol, might not always hold.

The main objective here is to establish methods for mining web service protocols.

name		oID	invID	pID
PO	m_1	o_1		
PO	m_2	o_2		
Inv	m_3	o_1	i_1	
Inv	m_4	o_2	i_2	
Pay	m_5		i_1	p_1
Pay	m_6		i_2	p_2

(a)



(b) $\psi_1 : m_x.\mathsf{OID} = m_y.\mathsf{OID}$, $\psi_2 : m_x.\mathsf{invID} = m_y.\mathsf{invID}$

Figure 2.2: (a) snapshot of the log of a process view node, (b) corresponding correlation graph $G_{\psi_1 \vee \psi_2}$; we notice the two conversations that are represented by the two sub-graphs [105].

So far, the set of topics that are addressed include log cleansing and log imperfections characterization and detection. Afterwards, in the same research context we encounter and include protocol conversation identification and mining as well as defining a model for protocol extraction solutions. Finally, we can mention user-interactive mining and temporal constraints discovery. For the issues to be improved or dealt with in the time to come, we can recall data quality which, does have an impact on the discovery process. As data quality problems one can mention (i) temporal accuracy, in which the event order recording does not follow the real-time execution sequence event order, (ii) completeness, from the point of view of WS execution instances recorded in logs compared to the existing paths in the protocol which are potentially not explored or are discarded, and (iii) consistency, i.e. compare the design-time model with other mined models that are inferred from the same data.

Table 2.1 displays some relevant papers in this area based on three characteristic dimensions encountered in service protocol discovery. The analysis on the previous references clearly indicates that research issues still open include the enhancement or adaptation of precedent solutions by further relaxing initial hypothesis (e.g. assumption of correlation information). Other open issues are the usage of less naive approaches concerning support-based noise filtering and in-depth research on more realistic and complex structures.

Table 2.1: Axis of comparison between some approaches on service protocol mining

Ref.	**Temporal aspects**	**Accounting for imperfections**	**Interoperability**
[108]	Untimed business protocols are considered, temporal constraints cannot be defined.	Support-based filtering of erroneous or infrequent messages.	Identical protocol used on both sides (service and client).
[50]	N/A	Imperfection-free data is assumed.	Different business protocols are employed for the service and client side.
[52]	Timed protocols are used with temporal constraints being the primary target.	The method assumes clean and correct logs.	Approach engages classic web services based on a unique protocol.

2.3.2 Application of protocol mining to service interactions

Web service interaction patterns [12] are abstractions of representative interaction scenarios. They are not always extracted from execution data, but empirically derived from the literature, standardization activities, and real-world use cases. They are mainly used to benchmark service functionalities related to WS choreography and orchestration.

In [57] a study on Web service interaction mining (WSIM) is reported. Instead of directly offering a solution to mining, this works restricts to an introductory analysis of the context in which mining takes place when considering WS activity traces. The authors in [57] present a simplified view of only three levels of abstractions, namely the WS operations, interactions and workflows. A further classification is also done inside the WS interaction layer. The logs considered here are supposed to contain any data that might be necessary during the mining process. Log events are mined using a Prolog-based representation of mining rules.

A restrictive assumption is nevertheless used, that is, a workflow-ID and an instance-ID are required and can only be available if provided by the WS itself in advance. Moreover, it is assumed that message extraction is always possible due to the fact that all interactions between web services take place through the exchange of

SOAP messages. We now know that with REpresentational State Transfer services being increasingly implemented, this assumption is not always correct. This work was continued in [56], where a classification of service-oriented systems is proposed, altogether with descriptions of the directions taken by log mining in the context of both rich and poor content log. The major contribution of [56] is a potential future agenda in WS mining.

In short, these methodologies could be quite useful for a preliminary study prior to the mining process. Thus, these might be used for defining the expectations on both the discovery process and the results to be obtained from it. These include questions such as: "In which scenarios does the extraction take place?", "Which level of abstraction should be utilized?", or "How results are to be interpreted?", etc.

2.4 Workflow model discovery

Workflow patterns [70] are defined as substructures of the directed graph representing a workflow, and are extracted from execution logs. They do not contain explicit temporal constraints. This model is supposed to be known for the generation of the patterns, and only the frequent ones are considered. Indeed, the method in [70] does not aim at knowledge extraction but at diagnosis and termination prediction. The importance of log mining applied to workflows and business processes is fundamental. As is underlined in [71], workflow modeling does need some requirements that jeopardize its efficiency and make it difficult to achieve. Such requirements include human experience, time and financial resources. Other aspects that make modeling even more difficult are also implementations which quite often do not obey to the designed model. Moreover, models tend to change relatively quickly in order to adapt to particular needs. These disadvantages, and others that we describe in the following can be avoided by mining these models.

2.4.1 Markov chains for workflows

In [39] the authors have improved two existing methods and proposed a novel one for software process model discovery from event logs. The first one, the $RNET$ method, is a statistical approach based on neural networks. It can characterize an execution state based on the past behavior. While being an extension of the implementation found in [41], it permits to take into account more event types. This is combined with an easier extraction of the mined model.

The second method, KTAIL, computes a potential state by looking at the future behavior instead of the past executions. The theoretical method for this extended

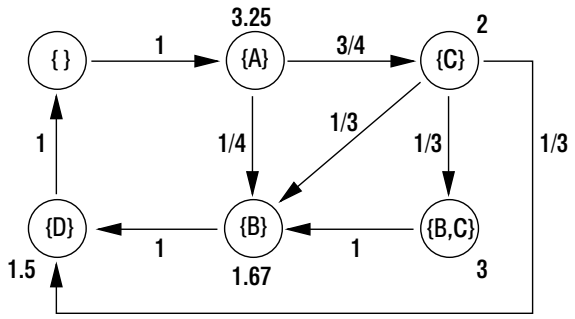


Figure 2.3: Markov chain computed using the method in [85]

implementation can be found in [23]. The last and novel approach introduced in [39], is based on Markov models. This method tends to ally the precedent solutions into one, by making usage of both past and future executions.

In [85], the authors propose a solution for mining service models in cross - organizational workflows by modeling services using continuous-time Markov chains. A cross - organizational business process is specified as requiring an external organization in order to fulfill at least one of its tasks or activities. The concept of service hereby is to be taken in a broad sense as an activity exchanged inside a requester/provider environment, and not to be identified with web services. The basis for the method is composed of service logs, each describing a single execution of the service to be modeled. One should note that in this approach service execution instances are supposed to be uniquely identified. Thus, it does not address the session identification problem. An example of the Markov chain that results from this approach is shown in Fig. 2.3.

More recently, the authors in [37] proposed a purely Bayesian approach aiming at process inference by learning Markov models. This approach proves to be robust especially against random noise. It provides another important answer regarding model validation. The validation test is based on the probabilistic estimation that two process-generated sequences belong to the same model.

2.4.2 Learning Petri-net models

In [144], the issue considered is the discovery of business process models in the form of workflows based on event logs. These logs store events associated with a given process. The workflow log containing the run-time process execution data represents the basic information which is used in order to obtain a Petri-net representation of the process model. The so-called α algorithm, constitutes a milestone for many

other improvements and methods that appeared later, namely [103, 127, 128]. Note that in [103] noise and log imperfection issues are considered, while in [144] they are not. The main idea in [144] lies on the definition of a class of workflow processes, called *Structured workflow net* (SFW). This class describes the processes that can be discovered by the α algorithm.

A formal proof is provided in order to show that there is a conditional equivalence between the sound log-based discovered workflow and the initial SFW-net. This discovery method assumes that the workflow instance to which an event refers, is known in advance. Moreover, a unique task is associated with each event, thus implying that the appearance of an event remains singular in respect to a task. These assumptions may pose restrictions if this method is adapted for other domains such as BPD (see Section 4.2.1). Despite the fact that the class of discoverable processes subject to this algorithm is quite large, efforts are still needed in the future to cope with its main limitations.

The first of these limitations comes from the fact that, as underlined by the authors, actually there is no evidence that the coverage of the discoverable process class is maximal. Another limitation to a more widespread usage of the method is imposed by the mandatory requirement that no short cycles must reside inside the initial workflow. A strong point of this approach is its linear complexity in the size of logs that overwhelms the exponential complexity in the cardinality of the tasks' set. Indeed, under normal circumstances the number of events is many times inferior compared to the number of log entries. Again, this hypothesis may not always be verified. Nevertheless, it does not eliminate adaptation perspectives in the future. An improvement of [144] is provided in [45] which deals with single and double-cycle identification in workflow logs.

In [103] a method is proposed for constructing the process model from process log data. This is done by determining the relations between process tasks. The relation mining is achieved through machine learning techniques for inducting rule sets. In order to obtain this, the so-called *Ripper* algorithm is employed [36]. The rule sets themselves are induced from simulated process log data generated by varying process characteristics specifically noise and log size.

[103] is also an extension of [144], studying the effects of noise, log size and imbalance in the process discovery. The concept of imbalance describes the scenario in which, given the Petri net of a business process, a task may take place in different percentages compared to another task. [103] relates on the effects of these three criteria (noise, log size, and imbalance) on the discovery result.

Potential mining applications also include conformance testing as reported in [127]. Here, two aspects of conformance issues are addressed. First, a metric is defined to calculate the fitness between a generated log and a process description. The fitness determines whether or not a given log may potentially be the product of a given

process execution. Second, two additional metrics are proposed to measure the appropriateness of a process model. The appropriateness verifies that the extracted model reflects correctly the behavior that generates the logs. These two metrics, the structural appropriateness a_S and behavioral appropriateness a_B, combined with the fitness metric, allow for the quantification of conformance.

Recent improvements in the discovery of process models are provided in the following papers. In [64] the authors address the problem of discovering the process model when the event log is provided as an unlabelled stream of events. The improvement consists in the absence of the identifier that correlates events to a process instance. The model is evaluated by means of an iterative Expectaction-Maximization procedure via a probabilistic approach.

In [81], the authors define commonly used process model constructs in the event log and adopt pattern definitions that capture these considerations. This is used as a method for transforming traces. This transformation is envisioned to be applied as a pre-processing step for existing process mining techniques.

In [28] we encounter a set of techniques for employing the theory of regions to transform a log into a Petri net, while avoiding the potential high complexity. The approach proceeds by using decomposition techniques, or log event clustering in order to work on projections. This allows the usage of classical region-based techniques for process mining techniques.

The work in [92] provides a search algorithm whose discovery of the process is based on past process modifications. This approach allows the extraction of a reference model. This reference model serves as a prototype that is used as a measure against the future evolution of the process. Another interesting feature is the capability of measuring the divergence between the mined reference model and the original one.

2.4.3 Learning graph-based models

Here we present approaches whose tools used for modeling as well as mining are supplied exclusively by graph theory. As stated in [78], the original finite-state machine (FSM) model appears to be easily understood and the corresponding activity-based model can be conveniently represented. Nevertheless, in the case of a reconstructed FSM, a given state will probably be incapable to represent the semantic meaning associated to it. This is because the transitions to multiple states may output the same execution outcome.

This case is illustrated in Fig. 2.4. Indeed, Fig. 2.4(a) represents the FSM of the software development process [38], which involves three sequential steps: code modification, compilation, and testing. After the code is modified (G), the subsequent

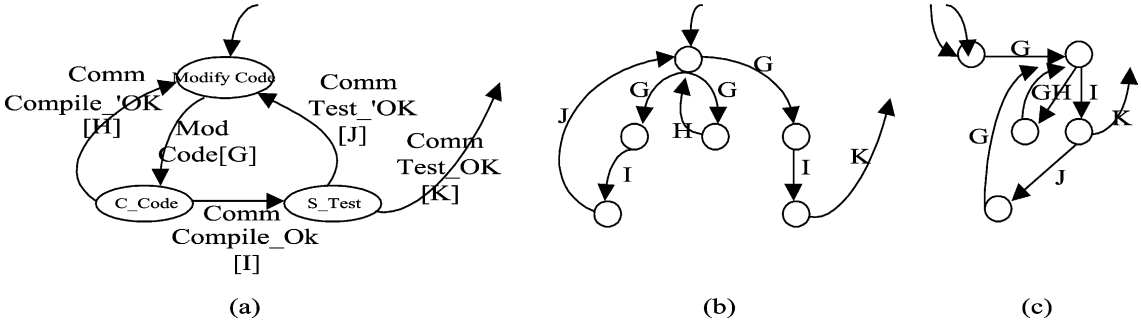


Figure 2.4: (a)The original process definition, (b) a KTAIL discovered FSM (c) a Markov-based discovered FSM [78]

compilation is performed and produces the result of either OK (I) or not OK (H). If the compilation is not okay, the code has to be modified again and the procedure has to be repeated; otherwise, a testing activity is performed. A successful testing (K) ends this process, and a failed testing (J) calls for the repetition of the entire procedure. Fig. 2.4(b) and (c) show the FSMs discovered from two different algorithms viz. KTAIL and Markov [38]. It is worth to notice that in Fig. 2.4(a), each state corresponds to the execution of exactly one real-world activity. The transitions leaving a state represent the possible execution results. Nevertheless, as Hwang and Yang correctly point out [78], in a derived FSM, such as that in Fig. 2.4(b) or (c), a state may not have its clear semantic meaning. Moreover, the execution outcome may appear in the transitions of multiple states. Yet, this weakness had been addressed in [2], where Agrawal et al. introduced the definition of a process using directed graph modeling. In their paper [2], vertices represent the process activities and edges stand for the set of control dependencies between the activities.

The authors in [2] define the issue of process mining in the context of graph theory. Named *Graph Mining*, this definition considers the discovery as being the construction of a compliant process graph, starting from the process execution logs. The heuristics of the algorithmic solution in [2] is to represent each activity of a process execution as an event associated with a precise instant. Therefore, a process execution instance can be represented in its turn as a sequence of activities. At this point the algorithm attempts to locate all detectable dependencies from the set of execution instances. In other words, the objective is to obtain the dependency graph from the process execution logs. A transitive reduction step is undertaken in order to minimize the constructed graph. This reduction step is responsible of the polynomial complexity of the overall algorithm. Cycles are considered by merging the vertices of a same activity.

Another important contribution of Agrawal et al. is their probabilistic study of noise occurrence in log sequences. The authors compute the probability of random errors occurring during the process executions, in the sequences of activity during the recording phase. Consequently, it can be proven that the probability of constructing the correct graph is δ. This δ value is bounded by the condition in Formula 2.4.1:

$$\delta \geq 1 - max(C_m^T \epsilon^T, C_{m-T}^m (\frac{1}{2})^{m-T}) \tag{2.4.1}$$

where T represents the number of errors occurring during m executions. In this formula, C_m^T is the notation of the number of k-combinations, and all variables of the formula have already been introduced and known in advance. Formula 2.4.1 allows to compute the probability of finding the correct result using the algorithm presented in [2]. This result being generic for similar scenarios and more sophisticated and robust than support-based noise estimations, it is thus more relevant in answering noise detection issues.

This work was afterwards improved in [78] by considering the executions of activity instances not as instant events but as time intervals delimited by the *start* and *end* time. These instants are either directly available or inferred from raw data. While the main algorithm, which presents a better performance, is a quite solid contribution, it is clear that the thorough study of noise is of greater importance. Noise filtering is achieved through statistical calculations of the probability of basic transition cases, assuming that the noise probability of an instance is known in advance. Actually, the same approach described in [2] is used in [78] for handling noise in data, but by focusing on optimization issues regarding error occurrences.

Another solution which combines graph-modelling and statistical methods is encountered in [132]. By employing a measurement error model, this approach allows to learn a restrained class of directed acyclic graphs. These particular graphs are called the AND/OR workflow graphs and are inferred by means of polynomial algorithms. The core algorithm proceeds by incrementally including child nodes to a partially built graph following a specific order. The assumptions upon which relies the whole approach are the classical ones that are found in the majority of workflow log based solutions. One main limit is obviously the exclusion of cycles in the considered class of graphs.

Table 2.2 summarizes the approaches described so far. The comparison dimensions are based on graph cycles and error effects, both being major issues that greatly influence graph mining efforts.

We summarize the entire Section 2.4 by listing the most relevant achievements in this area. These include obtaining the representation of a given workflow using event-based activity traces of the latter, providing approaches anchored on the past

Table 2.2: Axis of comparison between graph-based methods

Description	**Cyclic graphs**	**Error handling**
[78] Polynomial algorithm employing transitive reductions for minimizing the induced graph. Deals with basic patterns.	The loops considered are relatively simple and do not account for composed structures viz. parallel split inside a larger loop.	Very solid noise solution based on computing error bounds for threshold estimation. Noise-prone data is handled.
[105] Polynomial graph construction method. Step-based and user-guided graph refinement and enhancement.	Loops are addressed but approach could still be improved. The sequence length can have effects on the cycle discovery.	Support-based error analysis. Threshold is defined based on data ratios and does not rely on a statistical approach as in [78].
[2] Poynomial algorithm centered around the construction of (a)cyclic graphs followed by a transitive reduction step and edge marking method.	Cycles are easily identified since process execution instances are known in advance.	Minimize the effects of noise by computing error probabilities in sequences. Does not directly deal with the impact of noisy data in the resulting graph.

and potentially future behavior of a process. Other objectives are the usage of runtime process execution information and finally, conformance-testing using metrics for fitness. As suggested by the present analysis and other works [71], future directions for the upcoming research effort are numerous. These directions include (i) process instance identification during the process discovery, (ii) extension of the class corresponding to the discoverable processes, and (iii) taking short cycles into account. In addition, the analysis of imperfections (noise, data integrity and quality) in activity logs needs further investigation. Table 2.3 shows an organized and descriptive view of references on workflow mining, based on the three dimensions that dominate the actual bibliography.

Table 2.3: Comparative description of model solutions addressing workflows

Markov chains	**Petri nets**
[38] Software process discovery from event logs using neural networks for studying past behavior and potential future executions.	[144] Business process model discovery employing order relations between tasks to infer the model as a Petri net (α algorithm).
[85] Service model mining by continuous time Markov chain modeling. The parameters are directly driven by occurrence and time-stamped data.	[103] Obtain the process model by determining relations between tasks employing machine learning techniques (Ripper algorithm).
[37] Extraction of software process in the form of Markov model. The dynamics of the model are translated into transition matrices.	[127] Further extension of [144] considering conformance testing by means of metric-based evaluation of fitness and appropriateness.

2.5 Pattern mining

The main idea in pattern mining is to recognize, model, categorize and abstract patterns in data sequences. When considering this domain one can see numerous patterns studied in the literature. For the purpose of this survey we focus our attention on sequential patterns [3, 77, 97, 116, 144]. An analysis on temporal patterns [15, 35, 75, 76, 82, 88, 113, 129, 145, 149] is provided in Chapter 4, since these patterns are particularly related to the content of that chapter. We provide more information on the core approaches and related papers in the following.

Sequential patterns [3, 116] are basically event subsequences. In [97] they are called *episodes* and defined as directed acyclic graphs of events. They are extracted from a sequence database [3, 116] or from event sequences [97], using levelwise [3, 97] or pattern-growth [116] methods. It should be noted that only frequent patterns are sought here. Even if they provide some information about the order of events, they do not contain explicit temporal constraints.

One of the key references in this domain is [3]. Here the authors introduce the discovery of sequential patterns and the set of algorithms which deal with a sequence database. The sequences considered here represent a list of transactions and the transaction itself a set of patterns. We stress on this data representation because it shows a very powerful aspect of sequence mining. This capability of sequence mining is the very rich data granularity that can be encapsulated inside a sequence.

The solution proposed in [3] uses time constraints in the algorithm for separating adjacent pattern elements (Figures 2.1(e) and (f) illustrate this case). We recall that this is a technique that is very often encountered in interval-based temporal data mining (see Sect. 6.2.2). It should be noted that the term *time constraint* employed here does not stand for a temporal constraint. The latter refers to time-triggered events, while the former simply implies the usage of time-stamps in data during the mining process for determining time intervals. At the same time, transaction-bound constraints are not taken into account. This differs from what is done in many process, workflow [103, 127, 128, 144] and WS protocol mining methods [52, 56, 108].

In [97] we encounter a similar level-based organization of data. Sets of events, called episodes, represent user actions. The main issue here is to detect patterns of events, i.e. episodes, which have a high occurrence frequency. The algorithm performs a level-wise search in the class of episodes. It should be noted that the minimal frequency threshold employed here is basically the same as the one found in all support-based frequency computations [57, 108, 144]. This approach is based on the monotonicity property, stating that if an episode is frequent in an event sequence, then all its sub-episodes are also frequent.

The algorithm presented in [116] adopts a different approach, compared to other *Apriori*-like methods. Indeed, to reduce the number of potential candidates, the $Prefix - Span$ algorithm [116] obtains sequential patterns of larger size by exploiting uniquely shorter frequent patterns. The objective is to start with a sequence database which is in turn recursively projected into another set of smaller size databases. The corresponding sequential patterns are therefore incremented in size inside each projected database by using only the shorter frequent patterns provided by the $Prefix - Span$ algorithm. This allows to improve the overall performance.

In conclusion, the main achievements of the research community in sequential mining are: (i) the extraction from sequences of frequent patterns based on occurrence frequency using incremental approaches, and (ii) visualization of task-related order relationships and process mining in the collaborative context. In upcoming works regarding sequence pattern mining, the following aspects need further attention :

(i) Better account for resource dependencies and their effects on workflow discovery. Doing so, would allow to further enrich discovery results and thus allow for attractive enterprise solutions, knowing that they rely heavily on resource-related workflows.

(ii) Use more generic approaches for process instance identification. This problem seems to be currently bypassed using strong hypothesis as for example, existing session identifiers [57, 144].

2.6 Summary

In the context of knowledge discovery we presented in this Chapter a survey which went through existing solutions to the general problem of discovering behavior models from activity and interaction logs. The logs are assumed to be generated by structures described by models similar to a graph or a state diagram. The main subjects for such structures are processes, workflows, web services and patterns. The first contribution of the Chapter is to try giving a clear view on the existing approaches in this area, focusing on key papers and concisely reporting and analyzing related references. Its second contribution is a taxonomy that structures the bibliography of behavior model mining into separated layers in the context of DM in KDD.

The advantage offered by this taxonomy is materialized throughout the survey, and the common framework that is proposed for encompassing references. The common framework, the in-depth analysis and comparison of the approaches, and the updated and enriched bibliography constitute the third contribution.

Table 2.4 summarizes the capabilities of some of the key references based on their approach in dealing with behavior model extraction in the presence of noise. Noise was selected because it is arguably the most important factor that can jeopardize the model mining process.

Based on our work, several remarks can be drawn. Integration of approaches will improve the discovery process provided by accommodating noise, incorrect log entries, incomplete log data sets and heterogeneous data sources and formats. The analysis conducted suggests that modeling tools (automatons, Petri-nets, etc.) need to be adapted in order to support complex systems so that they can stress on and reflect the level of granularity of real business processes and web services. Extensive use of existing standards is mandatory for future research since it can in many cases justify the usage of information data employed during knowledge discovery.

The quantity of information available in raw logs, has also to be reconsidered. This would open new horizons for unexplored applications, for example using semantic content for discovering constraints, and for improving existing ones. Another example could be the usage of WS protocol message content, so far ignored, for session identification, or false-constraint detection. Privacy issues also need to be considered, be it for businesses or individuals, this being also supported by the study in [71].

In summary, there are a great number of open issues that need to be addressed. Without being exhaustive, we mention the topics of correlating messages and events, using semantic content, dealing with noise and data uncertainty, and privacy problems.

Table 2.4: Overall synthesis of methods employed for discovery and noise robustness

Ref.	**Mining method**	**Noise filtering**
[45, 144]	Algorithmic discovery of the class of structured workflows nets. Further enriched with a taxonomy of recurrent patterns.	The impact of noise is considered in the resulting decision rule sets but no data screening is provided.
[105, 107, 108]	Graph-based message correlation and finite-state machine construction using graph decomposition algorithms.	Support-based filtering, may present problems for completeness of discovery.
[2]	Extraction of directed cyclic graphs using transitive and edge marking reductions for inducing subgraphs.	No data cleansing, yet offers a probability value for successful results.
[39]	Probability computation of occurrences of event sequences based on a Markov model which is converted into a non-deterministic transition-labelled FSM.	Since the approach is robust to noise, no data filtering is proposed.

Chapter 3

A mathematical model for message dynamics

In this chapter we give a model that describes the mechanics of message flow during the execution of a service, and describe how these mechanics can be retraced by using the principle of the conservation of message occurrences.

3.1 Preliminaries

3.1.1 Notations and definitions

Let *Msg* be a set of *message labels*. A *message type* will refer to the label of a message. A *message occurrence* is a couple $M = (m, t)$, where $m \in Msg$ is the message type, and $t \in \mathbb{R}_+$ is the timestamp of the message (denoted $M.t$). A *message log* (ML) is a collection of entries $e = (MID, m, s, r, c, t)$, where MID is the message unique identifier, m is the message type, s and r denote the sender and the receiver of message m, c the content of the message, and t is the corresponding timestamp. The timestamps are local, in the sense that no global clock is needed. If x is a message type, we will denote by $\bar{x}$ the number of occurrences of x recorded in a message log ML.

An *occurrence log* (OL) is an array in which each column denotes a message type, and the corresponding row value provides the number of occurrences found for that message type in a message log ML. In other words, every message log ML_i is represented as a single line in the occurrence log OL. A *conversation* is a sequence of message occurrences $C = \langle M_1, M_2, \ldots, M_n \rangle$, where $n \in \mathbb{N}^*$, and $M_1.t < M_2.t < \cdots < M_n.t$. Each web service client exchanges a precise sequence

of message occurrences (i.e. a conversation) during the interaction with the web service provider. A *conversation log file* L is a multi-set of conversations.

Definition 5 The *column rank* of a matrix A is the maximal number of linearly independent columns of A. Similarly, the *row rank* is the maximal number of linearly independent rows of A. The column rank and the row rank being always equal, they represent the *rank* of A. The rank of an $m \times n$ matrix is at most $min(m, n)$. A matrix that has a rank as large as possible is said to have *full rank*; otherwise, the matrix is *rank deficient*.

3.1.2 Linear regression

During the runtime of a WS, a conversation of exchange messages takes place between the client and the service provider. The business protocol visualizes all the possible conversations that are allowed by this WS. When considering the protocol of a WS during runtime, one observes that message occurrences do not appear or disappear in a chaotic way. The first intuition is that the number of message occurrences for different message types are correlated. This intuition leads to another one: there is a law to which obeys the appearance of message occurrences in the log. For example, if the loop-free conversation sequence $\langle A, B, C, D \rangle$ is followed three times, then one naturally expects to find three occurrences for each message of this conversation.

The idea in this part of our contribution is to use linear regression methods to derive the equations that describe the relationships which exist between the numbers of different message occurrences. This is relevant for many reasons. First, in order to achieve the correlations between messages, the linear regression method requires only the number of occurrences for the given messages. To extract the full protocol after the correlation step, the approach needs only the recorded timestamps of message occurrences. Thus, an approach based on these intuitions and also linear regression offers the advantage of requiring only a very restricted quantity of log information. Second, linear regression is robust towards some of the forms of noise. In this chapter, noise that may affect the result is considered to come in two different ways. It can appear as missing message occurrences or erroneous timestamps. Both cases were considered during experimentations.

In statistics, linear regression refers to any approach that models in a linear fashion the relationship between one or more variables denoted by y and one or more variables denoted by $X = x_{i1}, ..., x_{ik}$. In such a case, the denomination "linear model" is employed. When considering a data set $\{y_i, x_{i1}, ..., x_{ik}\}$, $i = 1, ..., n$ of statistical variables, a linear regression method can be applied between the dependent variable y_i and the vector of regressors x_i of size k *iff* the relationship between them is, at

least approximately, linear. This linear relationship between y and x_i is expressed as $y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i = x_i'\beta + \epsilon_i,\ i = 1, ..., n$. For additional information on the regression tools used in this part of our work please refer to [80]. We precise that the linear regression inference is obtained by using the linear least squares approach. Using linear regression for solving the issues of this topic requires the assumption that the design matrix X must have full column rank p. For this property to be verified, we must have $n > p$, where n is the sample size. The sample size represents the number of entries in a given log.

Linear least squares (LLS) [32,60] is a method employed for computing the linear regression and for fitting data to a mathematical or statistical model. The general problem can be stated as follows. Consider an overdetermined system:

$$\sum_{j=1}^{n} X_{ij}\beta_j = y_i (i = 1, ..., m) \tag{3.1.1}$$

of m simultaneous linear equations (hereafter referred to as SLE) in n unknown coefficients, β_1, β_2, ..., β_n with $m > n$, written in matrix form as $X\beta = y$ where:

$$X = \begin{pmatrix} X_{1,1} & \cdots & X_{1,n} \\ X_{2,1} & \cdots & X_{2,n} \\ \vdots & \ddots & \vdots \\ X_{m,1} & \cdots & X_{m,n} \end{pmatrix} \qquad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix} \qquad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

In statistics, LLS is the computational foundation for ordinary least squares analysis (OLS), which is the method of regression analysis employed in this chaper. The formulas for linear least squares fitting were derived by Carl F. Gauss. The algorithms presented here employ LLS because LLS is at the same time simple and efficient in inferring the linear equations which describe the flow of messages in a business protocol. Ordinary least squares is the simplest and the most widely used method for estimating the parameter β. OLS is often employed to model experimental and observational data. Suppose b is a "candidate" value for an estimate of parameter β. Then the expression $y_i - x_i'b$ will be called the residual of i-th observation. The value of b which provides the minimal value for this expression will be called the *least squares estimator* for β.

Table 3.1: Occurrence log line OL_1 derived from the log in Table 3.1.2

LR	**CT**	**MO**	**TT**	**LP**
2	2	1	1	1

Table 3.2: Example of raw log ML_1 of SOAP-based service execution messages

MsgID	**Msg. label**	**SenderID**	**ReceiverID**	**...**	**Timestamp**
M_1	LR	192.168.5.1	192.168.15.4	...	10 : 01 : 07 13 : 52
M_2	CT	192.168.15.4	192.168.5.1	...	10 : 01 : 07 13 : 55
M_3	MO	192.168.5.1	192.168.15.4	...	10 : 01 : 07 13 : 60
M_3	LR	192.168.5.1	192.168.21.7	...	10 : 05 : 07 07 : 15
M_4	CT	192.168.21.7	192.168.5.1	...	10 : 05 : 07 07 : 26
M_5	TT	192.168.5.1	192.168.21.7	...	10 : 05 : 07 07 : 35
M_6	LP	192.168.21.7	192.168.5.1	...	10 : 05 : 07 07 : 46

3.2 Correlation and discovery approach

In this section we provide some introductory examples and describe how the process of correlating messages and inferring the business protocol is achieved [104].

Example 1 Table 3.1 depicts the occurrence log OL_1 which records the number of occurrences for each type of message encountered in the log ML_1. An example of the logs considered in this chapter is given in Table 3.2. We recall that a message type is represented by its label. Each single row in OL_1 is deduced from an entire raw message log ML_i. Thus, an occurrence log can be seen as a very compact form of raw web service logs. This form is at the basis of the approach presented here for achieving the correlation of messages.

At this point we consider some characteristic structures obtained by the decomposition of a typical business protocol into simple and composed modules that repeat themselves in a FSM.

Figure 3.1 displays the structures that are most often encountered in a business protocol. We can associate to each structure an equation describing the number of message occurrences that enter and exit the corresponding state. The $|a|$ notation, where a is a message type, denotes the number of message occurrences of a that are recorded in the log during the execution of the given structure.

A detailed study of log data associated to a known portion of a business protocol reveals the strong relationship between the type of transitions and the respective occurrences pattern. We notice that consecutive directed sequences of transitions generate sets of time vectors that have cardinalities of the same order plus/minus some slight difference. These set of vectors are also right-shifted on both bounds in comparison to the time interval of the preceding message.

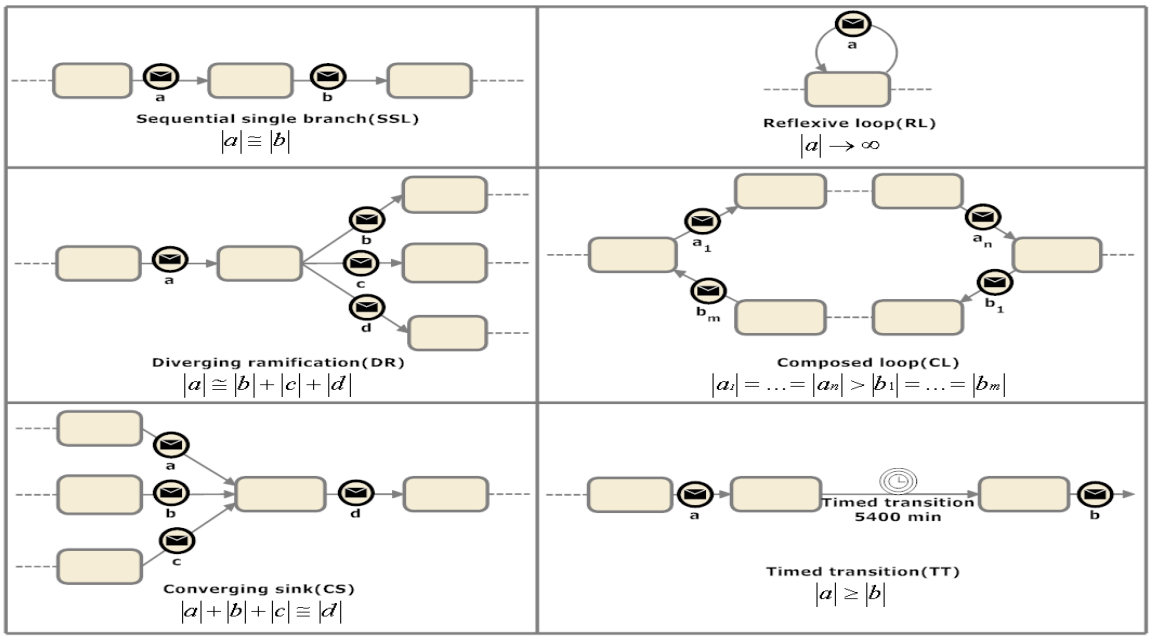


Figure 3.1: Modules of a business protocol P_1.

Figure 3.1 illustrates also the case in which loops exist in the business protocol model. The complexity of the protocol extraction process from logs is increased by the existence of such loops. This is because their presence undermines many of the properties that are otherwise verified in cycle-free automatons. Nevertheless, despite the difficulty introduced by loops, it still remains possible to identify them by means of their typical behavior. Indeed, as different sources and our experimentations seem to indicate, messages associated to loops have very high frequencies (i.e. the cardinality of loop message occurrences is very high). This is in contrast with structures like sequential sequence branches, diverging ramifications, and timed transitions.

Remark: It should be kept in mind that one of the many differences between a business protocol and a process workflow resides in the fact that workflows consider tasks, which often execute in parallel. Parallelism is thus frequently encountered in workflow models. But, in the case of a business protocol, the semantics of a Web service protocol leads us to consider only sequential transitions. To each business protocol instance which executes on a server corresponds only one execution path, i.e. one conversation. If we may speak of parallelism, it has only a meaning in the schematic model representation of a business protocol, but is meaningless when speaking of real-time execution.

3.2.1 Modeling the dynamics of business protocol messages.

Proposition 1 *The algebraic notation of a protocol is equivalent to its FSM representation in terms of descriptive power.*

Proof. The proof is straightforward. Since each equation of the SLE corresponds to the flow of messages related to a state, then a SLE describes the set of states and the set of transitions which connect them. In other words, this is identical to defining a graph by specifying its sets of vertices and edges. The relationship between the SLE of a protocol and the FSM of the same protocol is identical to the relationship between the set-based definition of a graph and the visual graph model. □

Consider the protocol sample shown in Figure 3.2. In all the protocol figures shown in this part of the manuscript, the dashed circles (here S_1) represent terminal states, as opposed to full circles that designate normal states. This example will describe how a set of linear equations describes the dynamics of messages in a business protocol with loops. The SLE provided in Equation 3.2.1 describes this protocol in algebraic terms. The equations marked with the same number of asterisks (*) or (**) are algebraically equivalent.

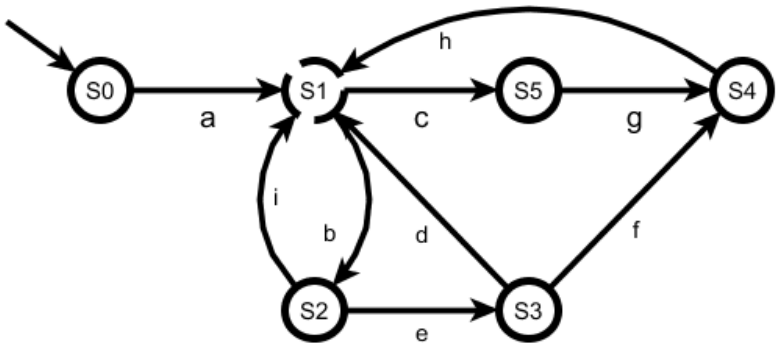


Figure 3.2: Simplified business protocol.

$$\begin{cases} 1.\ \bar{c} = \bar{a} - \bar{b} + \bar{d} + \bar{h} \\ 2.\ \bar{b} = \bar{a} + \bar{d} - \bar{c} + \bar{h} \\ 3.\ \bar{d} = \bar{e} - \bar{f}\ (*) \\ 4.\ \bar{e} = \bar{b} - \bar{i}\ (**) \\ 5.\ \bar{f} = \bar{e} - \bar{d}\ (*) \\ 6.\ \bar{g} = \bar{c} \\ 7.\ \bar{h} = \bar{f} + \bar{g} \\ 8.\ \bar{i} = \bar{b} - \bar{e}\ (**) \end{cases} \tag{3.2.1}$$

The method for constructing the SLE starts by considering each state at a time. Each linear equation is related to a state and provides the flow of messages in and out that particular state. Since there are several potential messages entering or leaving a state, then there may be several equations corresponding to a state. An equation describes a linear relationship between the number of occurrences of all messages that are directly related to a particular state. A given variable (a label with a bar) in an equation represents the number of occurrences of the message having the same label as the variable. On the right side of the equality sign are located all the other variables related to the remaining messages of that same state. Each variable is associated with a coefficient. The variables on the right side have either a + or − sign. This sign is very important since positive-signed variables represent messages that exit the corresponding state, and negative-signed variables stand for messages entering that state. Theoretically, all coefficients are ± 1, since a message can neither be split in two, nor created or disappear. Nevertheless, since the logs we consider here may exhibit defaults, such as missing messages, and because of existing loops then the experimental coefficients may be non-integer values. Experimental results however allow to define the interval inside which a coefficient value is considered as trustworthy. Moreover, we exploit the high-value coefficients to detect the existence of complex structures with unprecedented ease.

Consider the generic form of the state of a business protocol depicted in Figure 3.3.

For a given existing message A_k, $1 \leq k \leq m$ the following equation can be stated:

$$\bar{A}_k = \sum_{j=1}^{n} \bar{B}_j - \sum_{l=1}^{m} \bar{A}_l + \bar{A}_k = \sum_{j=1}^{n} \bar{B}_j - \sum_{l=1, l \neq k}^{m} \bar{A}_l \tag{3.2.2}$$

where $\bar{A}_k$ denotes the number of occurrences of message type A_k that have transited outside state *State*. Note the difference between the number of types of messages (denoted by n) and the number of occurrences of each type of message. A type

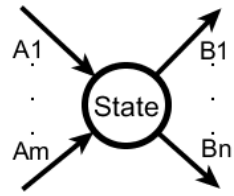


Figure 3.3: General form of a protocol state.

of message example corresponds simply to a message label e.g. *"browseProducts"*. Equation (3) is correct since the total of messages occurrences entering a state is obviously the same as the number of those exiting the same state. In other words:

$$\sum_{j=1}^{n} \bar{B}_j - \sum_{l=1}^{m} \bar{A}_l = 0 \tag{3.2.3}$$

One can now see that Equation (3) is obtained by adding $\bar{A}_k$ to both sides of Equation (3.2.3) which is arithmetically equivalent. Moreover, Equation (3.2.3) states the law of conservation of the number of message occurrences.

3.2.2 Algorithmic procedures

Let us introduce the algorithms that exploit the SLE approach in different ways.

The naive approach: the $n - delta$ algorithm

The naive version of the algorithm uses as input the log containing all message occurrences recorded during the web service activity. The corresponding pseudo-code is shown Algorithm 1. The algorithm proceeds by taking a message type vector at a time, and by computing the coefficients of all the possible equations that can be obtained throughout all possible combinations of message type in equations of size i. The provided result is the set of correlation matrices. The logical AND-sum of the correlation matrix yields the final correlation matrix.

Theorem 1 *The complexity of the naive-delta (n-delta) algorithm is* $O(2^n)$

Proof. Let n be the number of variables, i.e. the number of messages occurring in the given protocol. We note by c_j, $j = 0, ..., k$ the coefficients of each variable in the

Algorithm 1 *naive − delta*

Require: Occurrence log A of n message types
Ensure: M_{xi} the set of correlation matrixes // $n-1$ *in total*

```
for i = 1 to n − 1 do
    B = A
    while B ≠ ∅ do
        Choose an occurrence vector b ∈ B
        Remove the column of b from B
        Rs = B // Rs is the matrix of regressors
        RegressorsSet = combinations(Rs, i) // the set of non-redundant combinations
        of i elements.
        while RegressorsSet ≠ ∅ do
            Choose r_i ∈ RegressorsSet
            // r_i is an occurrence vector of size i
            Remove the column of r_i from RegressorsSet
            C = OrdinaryLeastSquares(r_i, b)
            Insert correlation vector C in matrix M_xi
        end while
    end while
end for
```

equation. Let k represent the size of the right-hand part of a given equation. For example, if we have $a_0 \times c_0 = a_1 \times c_1 + a_2 \times c_2 - a_3 \times c_3$ then $k = 3$. Let i be the index of the variable on the left-hand of the equation.

We want to compute the number of correlation tests that are necessary for establishing all existing correlations between a variable x_i and all the other existing variables, in equations of size $k+1$. Then, the number of correlation tests is straightforward:

$$C_{n-i}^{k} = \frac{(n-1)!}{k!(n-i-k)!} \tag{3.2.4}$$

For n variables and fixed size k, there are $n-k$ elements of the sequence. Hence, the

number of tests for a sequence of size k is :

$$\sum_{i=1}^{n-k} C_{n-i}^{k} = \sum_{i=1}^{n-k} \frac{(n-1)!}{k!(n-i-k)!} \tag{3.2.5}$$

Thus, the number of correlation tests for all sequences' lengths (i.e. for all possible values of k) is:

$$\sum_{k=1}^{n-1} \sum_{i=1}^{n-k} C_{n-i}^{k} = 2^n - n - 1 \approx O(2^n) \tag{3.2.6}$$

Since n is known in advance for a given system, the above value of the last equation is fixed, and represents the number of all the correlations that are to be tested using the naive and exhaustive method to determine the existing satisfying coefficients between all the variables. Because the linear complexity of n obviously has no effect whatsoever on the overall complexity, therefore only the exponential term of the expression is useful. □

Remark: The correlation matrix M_x is obtained by the scalar sum of all the matrices M_{xi} provided by Algorithm 3.2.2. Formally speaking $M_x = \sum_{i=1}^{n-1} iM_{xi}$.

Improved version of $n - delta$

An improvement of the naive method is quite of interest, since it leads the path toward *delta* with its optimal efficiency and very low complexity. This improvement starts with the maximal value allowed for k. If, for the first value of k not all variable correlations are established, then the GLS is re-run once again with $k-1$. The loop iterates decrementing k, as far as all of the variables are not correlated. This leads to an interval of potential complexities whose values depend on the scenario considered. The boundaries of this interval are:

(1) Worst case: Correlations must be tested until $k = 1$, at which stage all possible combinations are to be run. In this case the complexity is $O(2^n)$. An illustration of this worst case is provided in Figure 3.4.

(2) Best case: Only $k = n - 1$ has to be tested (i.e. correlations for a single value of k are to be tested), in other words a loop on the entire matrix. Thus, the complexity is $O(n)$. An illustration of this optimal case is given in Figure 3.3. Of course this requires that the entire protocol is contained in a single state.

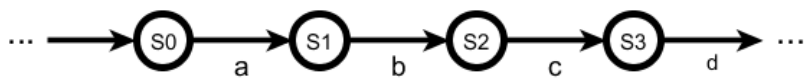


Figure 3.4: Example of worst-case complexity scenario.

The optimal version of the *delta* algorithm, which is shown in Algorithm 2, uses the same input as $naive - delta$. The algorithm proceeds in a much simpler fashion by taking a message type vector at a time, and by computing *only* the coefficients of the equations of size $n - 1$. This algorithm directly provides the final correlation matrix.

The optimal approach: *delta* algorithm

Algorithm 2 *delta*

Require: Occurrence log A of n message types
Ensure: M_x the correlation matrix

```
B = A
while B ≠ ∅ do
    Choose an occurrence vector b ∈ B
    Remove the column of b from B
    Rs = A // Rs is the matrix of regressors
    Remove the column of b from Rs
    C = OrdinaryLeastSquares(Rs, b)
    Insert correlation vector C in matrix M_x
end while
```

The complexity of Algorithm 2 is lower, as shows the following theorem.

Theorem 2 *The complexity of the delta algorithm is $O(n^2)$*

Proof. Let n be the number of the variables, i.e. the number of messages occurring in the given protocol. We want to compute the number of correlation tests between a variable and the remaining variables. The right hand of the equations in this case is set to $n - 1$. Then the number of all correlation tests that are needed is:

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \approx O(n^2) \tag{3.2.7}$$

□

In most web service protocols the number of message types n is generally low. To give an idea on the order of magnitude, very large protocols such as eBay Trading service achieve a maximum of 64 message types [59]. Thus the square polynomial complexity has no significant impact on the overall performance of the algorithm. On the other hand, what has more influence is the number of message occurrences stored into logs. The complexity of the algorithm that counts the number of occurrences for each message type will then determine the overall complexity of the approach. If N is the number of all the different message occurrences stored in the service logs and M the number of message types, then the complexity of the counting algorithm is $O(M \times N)$. Since N is by many orders of magnitude greater than M, thus the resulting complexity is $O(N)$. In conclusion, the occurrence counting algorithm has a linear complexity. A direct consequence of this result is that the overall process of extracting the SLE that allows to mine the service business protocol has a good performance. The minor influence of the polynomial sub-algorithm was confirmed by experimental results on synthetic logs, as it is shown in Section 3.4.

3.2.3 Result interpretation and visualization

The result provided by the algorithm is a matrix composed of correlation vectors. The vectors correspond to the columns of the matrix. This correlation matrix allows to easily obtain the SLE corresponding to a protocol. Table 3.3 shows the generic form of this array. Let m be the number of message types in a protocol and a_1, a_2, ... a_m the message types that need to be correlated. The correlation matrix provides the coefficients that are to be used in an equation involving a variable a_i on the left side, and variables $a_j, (j \neq i)$ on the right side. If we have the matrix column shown in Table 3.4, the corresponding equation that is derived from this column is:

$$1 \times \bar{a} = 1 \times \bar{b} - 1 \times \bar{c} + 1 \times \bar{d} + 0 \times \bar{e} + 0 \times \bar{f} = \bar{b} - \bar{c} + \bar{d} \tag{3.2.8}$$

This equation describes two possible cases, that are illustrated in Figure 3.5. Note that the only difference between the two cases is the direction of message flow. In Figure 3.5(a), messages b, d enter the state and a, c exit it, while in (b) we have the reverse situation. The choice between these two possibilities is made by employing the timestamps of message occurrences to establish order relationships.

The matrix is to be interpreted as follows: we consider each column at a time. If for row k and column l the value i_{kl} is 0, then the message type corresponding to that row is not correlated to the message type associated with column l. Thus, each column expresses via a proper linear equation the correlation between the corresponding

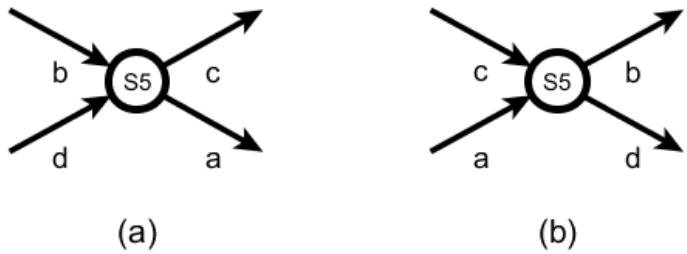


Figure 3.5: Graph-represented equivalent of a linear equation where incoming messages (a) become outgoing (b).

Table 3.3: General form of the correlation matrix with method result.

	a_1	a_2	.....	a_{m-1}	a_m
a_1	0	$i_{1,2}$	.....	$i_{1,m-1}$	$i_{1,m}$
a_2	$i_{2,1}$	0	.....	$i_{2,m-1}$	$i_{2,m}$
...	...	...	.....	...	...
...	...	...	.....	...	...
...	...	...	.....	...	...
a_{m-1}	$i_{m-1,1}$	$i_{m-1,2}$	.....	0	$i_{m-1,m}$
a_m	$i_{m,1}$	$i_{m,2}$	.....	$i_{m,m-1}$	0

message type and the other message types. Note in Table 3.3 that the diagonal values of the correlation matrix are always zero. This is done arbitrarily since the correlation between a message type and itself is always valid, thus not relevant. Furthermore, experiments have shown that for a correlation to be correctly estimated, the coefficient values a_i are to be found in the interval: $0.91 < a_i < 1.09$.

Table 3.4: Single occurrence vector

	a
a	0
b	+1
c	−1
d	+1
e	0
f	0

Table 3.5: Occurrence log of sample protocol in Figure 3.6

a	**b**	**c**	**d**	**e**
10	5	5	3	2
12	7	5	3	4
15	8	7	5	3
20	13	7	9	4
31	14	17	8	6

3.3 Correlation and discovery: a use-case example

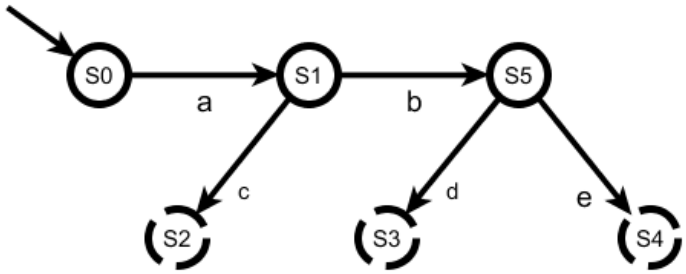


Figure 3.6: Use-case for correlation and business protocol discovery.

In the *delta* correlation method, not the naive or improved naive versions, if i stands for the number of rows and j for the number of columns, then for the correlation algorithm to correctly succeed, the data matrix *must* respect the condition $i < j$. This condition was first discovered empirically during experiments. The mathematical reason for this condition does exist, as the following example shows. Let us illustrate how the approach achieves the correlation of messages and protocol discovery using the following example.

Example 2 Consider the simple protocol in Figure 3.6. Table 3.5 shows the occurrence log deduced from message logs issued by the protocol in Figure 3.6. When the algorithm runs on the first three rows of this log, we obtain the resulting matrix in Table 3.6. On the other hand, when the algorithm runs on the entire log, we obtain the resulting matrix in table 3.7. Only now we observe that the result corresponds to the expected solution. Thus, we observe that the condition $i < j$ is necessary because it provides the most convergent solution that the LSF method can find on a rank deficient occurrence log.

Starting from the correlation matrix in Table 3.7 we obtain the linear system shown in Equation 3.3.1. This linear system leads to the straightforward states shown in Figure 3.7. Equation 1 is translated into the first state (Figure 3.7(a)), while equations 2,

Table 3.6: Matrix of coefficients computed from the first three rows of Table 3.5

	a	b	c	d	e
a	0	1	1	1	1
b	1	0	0	0	0
c	1	-1	0	-1	-1
d	0	0	-1	0	-1
e	0	0	-1	-1	0

Table 3.7: Coefficient matrix computed from the entire log in Table 3.5

	a	b	c	d	e
a	0	0	1	0	0
b	1	0	0	1	1
c	1	0	0	0	0
d	0	1	-1	0	-1
e	0	1	-1	-1	0

4, 5 provide the state shown in Figure 3.7(b) and equation 3 gives the state (c) of the same figure. From the SLE in Equation 3.3.1 we see that $\bar{b} = \bar{d} + \bar{e}$ and the log timestamps of b always precede those of d and e. Thus, the only possible outcome that unifies these three states is the one already shown in Figure 3.6.
Loops are easily identified by the fact that the coefficients of message types generated by loops have an absolute value much greater than 1. This is due to the fact that the number of executions of a loop is in general higher than those of linear transitions. In addition this number of executions is independent from the number of executions of the other sequential transitions.

$$\begin{cases} 1.\ \bar{a} = \bar{b} + \bar{c} \\ 2.\ \bar{b} = \bar{d} + \bar{e} \\ 3.\ \bar{c} = \bar{a} - \bar{d} - \bar{e} \\ 4.\ \bar{d} = \bar{b} - \bar{e} \\ 5.\ \bar{e} = \bar{b} - \bar{d} \end{cases} \tag{3.3.1}$$

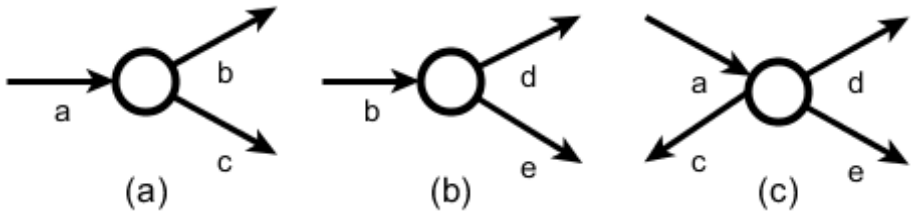


Figure 3.7: States obtained from (a) equation 1, (b) equations 2, 4, 5, and (c) equation 3 from the linear system in Equation 3.3.1.

Table 3.8: Impact of M on scalability and performance.

# Msg. type (M)	**Min. MT** (s.)	**Max. MT (s.)**	**Avg. MT(s.)**
10	0.004	0.030	0.017
25	0.010	0.060	0.035
50	0.070	0.140	0.105
75	0.190	0.280	0.235
100	0.370	0.520	0.445
200	3.170	4.960	4.065

3.4 Experiments

Synthetic data generated by a business protocol simulator were used to test the correctness and performance of the algorithms [1]. Experiments were conducted to study the scalability of *delta*. The influence of noise was also investigated by introducing errors in the logs. The desired percentage of errors introduced was variable so that the evolution of the behavior of the algorithm would be clearer. The business protocols employed for synthetic data generation are of realistic sizes (up to 100 message types and 10^6 web service instance conversations). The implementation of the algorithm and the experiments are conducted using Matlab. Nevertheless, the migration towards other mathematical software requires little effort.

Table 3.8 shows the experimental results on the comparative performance of the *delta* algorithm versus the scalability of message types. $Min.\ MT$, $Max.\ MT$ and $Avg.\ MT$ stand for respectively Minimal, Maximal and Average Measured Time. Table 3.8 clearly proves that employing OLS for constructing the correlation matrix is efficient. Table 3.9 depicts the same comparative performance of the algorithm but this time it is based on the number of message occurrences. One should observe the

[1] The simulation tool can be downloaded at http://liris.cnrs.fr/kreshnik.musaraj/technology/ simulation/index.html. The Matlab models used as well as the source code of the algorithms can be downloaded at http://liris.cnrs.fr/kreshnik.musaraj/technology/ws/index.html

Table 3.9: Impact of N on scalability and performance.

# Msg. occ. (N)	**Min. MT** (s.)	**Max. MT (s.)**	**Avg. MT(s.)**
1000	0.10	0.50	0.30
2000	0.30	1.10	0.65
5000	0.92	2.50	1.71
10000	2.10	3.62	5.13
50000	27.50	32.06	29.78
1000000	57.01	67.29	62.15

Table 3.10: Impact of noise on coefficient estimation with OLS

Δ #Occurrences (%)	**Avg. Δ #Coefficients (%)**
1	$\pm$ 0.03
5	$\pm$ 0.7
10	$\pm$ 15.7
20	$\pm$ 45.0

convergence of $Min.\ MT$ and $Max.\ MT$ towards $Avg.\ MT$ for increasing values of M and N.

Table 3.10 shows the impact of noise on the values of coefficients estimated using OLS. The first column provides the difference in percentage between the accurate number of occurrences and the value reported on the noisy occurrence log. The second column shows the percentage of divergence between the coefficients estimated using perfect data and the values estimated using message logs subject to missing message occurrences. It is important to notice that, when exposed to incorrect occurrence logs, the method will provide the correlation matrix that best fits the data. In this sense, the resulting protocol will be adapted to each occurrence log, thus the result will evolve as a function of the accuracy of the log. Nevertheless, noise impacts mainly the data contained in the recorded messages, and the rate of missing messages is much lower than the rate of incorrect data. The only exception is the case of error-prone log software, which goes beyond the scope of the present chapter.

3.5 Summary

In this chapter we have shown that extracting the protocol model without the assumption of correlation information in service logs is a difficult, yet useful task. In order to achieve this objective we have presented an approach that employs only the existence of message occurrences and their corresponding timestamps. We have proven

that with only two message attributes, namely the message type and timestamp, it is possible to extract the non-oriented graph modeling the protocol. This result is obtained through an equivalent representation of a business protocol in the form of a simultaneous linear system. We have shown that the least squares method is capable of obtaining this linear system, while at the same time being noise-resistent. Finally, one should note the restriction that currently exists regarding this approach. Indeed, a message type must appear only once in order for the approach to provide a correct result. We are already investigating a solution for this limitation. Meanwhile, the *delta* algorithm offers a solution to the issue of correlating messages related to the corresponding states in the business protocol.

Chapter 4

Time series analysis of log data

In this chapter we present a variable grain-size algorithm that extends the usage of temporal operators, and based on the study of cardinality properties, it allows the correlation between timeseries. The approach does not operate on any assumption on the existence of extracted facts and is capable of inferring temporal data facts and handling the pre-processing step.

4.1 Preliminaries

4.1.1 Notations and definitions

The term *time series* is defined as the time course of a set of variables under controlled conditions. A *temporal pattern* is the time interval in which one or more time series assume a behavior of interest. The T_1 notation will denote the type of time series based on simple timestamps. This corresponds to logs that are timestamped using a relative clock. The T_1-type time series provides a graph modeling two-by-two message relationships with Allen operators. The T_2-notation represents the type of time series that are based on the temporal evolution of the number of message occurrences. This type corresponds to logs that are timestamped using a universal clock. The T_2-type time series provides proof on temporal order and the correlation between message types based on the message flow density of occurrences. The terms *time interval* and *time window* will be employed as synonyms of a continuous and finite time duration precisely defined on the timeline. *Noise* will designate all the different types of imperfections occurring in logs, whatever their cause and nature. *Uncertainty* represents the notion of confidence attached to different sources of information when different values exist for the same variable or attribute.

Definition 6 A **universal clock** is the absolute timeline that follows the flow of the real-world time. Its implementation corresponds to the system clock, mainly in the UTC format.

Definition 7 A **relative clock** is one that is initialized at the runtime start of a WS instance. Its existence and implementation are not real. The relative clock is used only to obtain T_1-type data for determining the message relationships based on the Allen operators. A relative clock is obtained by shifting universal clock timestamps by the timestamp value of the first message to occur in the protocol instance.

Note: Determining the first message to occur in the protocol instance is not straightforward and we will show how to achieve it in the following.

Definition 8 The **rate** of the occurrence of a message type is defined as the ratio of the number of occurrences $\#M_{type}$ for that message type, per *time unit* T_{unit}. Hence we denote:

$$\mathcal{R} = \frac{\#M_{type}}{T_{unit}} \tag{4.1.1}$$

Definition 9 The **flow density** $\mathcal{D}$ of a message type is defined as the ratio of the number of occurrences $\#M_{type}$ for that message type, per *time interval* I_{id}. Formally speaking:

$$\mathcal{D} = \frac{\#M_{type}}{I_{id}} \tag{4.1.2}$$

Definition 10 A **piecewise linear function (PLF)** is a segmented mathematical function that is composed of several linear functions, each function being defined on a given interval of values. The algebraic notation requires the linear function equation to be provided altogether with the interval of values in which the given linear function applies. Intuitively, a PLF can also be defined as the set of points that separate two intervals from one another.

Proposition 2 *Let M be a message, $\#M_{type} \in \mathbb{N}$ the number of message occurrences, and I_{id} the identifier value of a temporal (sub)interval. The values of flow density $\mathcal{D}$ are described by a positive, integer-valued, discrete domain and discontinuous function:*

$$f : \mathbb{N} \to \mathbb{N}$$

$$f(t) \rightarrow \#M_{type}$$

Proof. The number of occurrences of messages during a temporal subinterval is obviously a zero or positive integer value. The temporal interval being divided into a constant number of subintervals, then the domain of the function is discrete. Since the domain of the function is discrete, so will be the values of f. Thus, f as a discrete-valued function is therefore discontinuous. □

Proposition 3 *The real-valued function f obtained by connecting the measured values of flow density $\mathcal{D}$ of a message type M_{type} is a continuous PLF, also differentiable except at D points.*

Proof. The segment which connects two values of $\mathcal{D}$ is real valued and a linear function, thus a single segment is always continuous and differentiable. Since all the segments composing the PLF are, given the definition of f, connected at the computed values of flow density $\mathcal{D}$, the PLF is entirely continuous. On the other hand, the derivative of F at the connection points is no longer valid. Informally this is due to the fact that the gradient (slope) of two adjacent segments is different and abrupt in change. □

Affine transformation

An affine transformation corresponds to shifting the origin of the coordinate system according to a specified vector. It can also be interpreted as shifting each point of a plane following the same unique vector. An affine transformation is verified iff: (i) lines are transformed into lines (the co-linearity property) and (ii) the distance ratios between points remain constant after the transformation. An affine transformation can also be expressed as combined vertical and horizontal shifts.

Definition 11 Let L_A, L_B be the occurrence log entries for message types A and B, $P = \begin{pmatrix} x \\ y \end{pmatrix} \in L_A$, $Q = \begin{pmatrix} x' \\ y' \end{pmatrix} \in L_B$ be two points of type T_2 time series. The ***affine transformation (AT)*** of P into Q is formally computed as:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & S_x \\ 0 & 1 & S_y \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4.1.3}$$

In Equation 4.1.3 we notice the horizontal shift constant $S_x = x' - x$ and the vertical shift constant $S_y = y' - y$. For more details on affine transformations, the reader is referred to [153].

4.1.2 Problem statement

Now that we have introduced the notations and basic definitions along with the required motivation, we may provide a rigorous definition of the issue of protocol mining from activity logs in a realistic scenario.

Context: *Let ML be a message log, i.e a log of message occurrences. Let $m \in Msg$ be a message type. By definition, only the message label and the timestamps are available from ML. The occurrences of messages are timestamped using a universal clock.*

Problem: *Is it possible to reconstruct the business protocol of the web service that generated the log ML in the first place?*

4.2 Discovery heuristics and approach

4.2.1 Theoretical considerations and methodology

Let $D_A(t)$ and $D_B(t)$ be two time series describing the values over time t of the flow density functions of A and B. We recall that the flow density function is defined by the set of points that are computed from the occurrence log OL by the $computeFlowDensityData$ algorithm. The cardinality of $D_A(t)$ and $D_B(t)$ is the same for all message types. In other words, at a given interval, each message type has a unique value of flow density.

Property 1 *Let $D_A(t)$ be the flow density function of message type A. Let $f_{D_A}(t)$ be the PLF constructed from $D_A(t)$ according to Definition 10. There is a defined function $\mathcal{F} : f_{D_A}(t) \rightarrow D_A(t)$ such that $\mathcal{F}$ is surjective.*

Proof. Since each $f_{D_A}(t)$ is obtained from a flow density function $D_A(t)$, then the conclusion is straightforward that for every $f_{D_A}(t)$ there is at least one flow density function $D_A(t)$ such that $\mathcal{F}(D_A(t)) = f_{D_A}(t)$. $\mathcal{F}$ is then, by definition, a surjective function. □

To each PLF corresponds at least one flow density function, i.e. several FD functions may define the same PLF, but every PLF is defined by at least one existing FD.

Remark: We will see how in reality this function is bijective, by studying the granularity level and noise effects on the differentiation between two $PLFs$.

Up to this point, we announce another property of the flow density function that is mandatory for the approach to work correctly.

Property 2 *Let $P_i \in D_A(t)$ and $Q_j \in D_B(t)$ be two flow density values of their respective message type. There is a bijective function $i : D_A(t) \times D_B(t) \rightarrow (P_i, Q_j)$ that correlates each point P_i to a unique point Q_j based on their indexes i and j.*

Proof. Consider $P_i \in D_A(t)$ and $Q_j \in D_B(t)$. We have already shown that to each subinterval value (x axis) corresponds an image of a flow density function. In other words, all flow density functions are defined at every subinterval value. Two cases are possible when correlating the points of $D_A(t)$ and $D_B(t)$. Either P_i and Q_j are located in the same subinterval, or they belong to different subintervals. In the first case, the bijectivity of the function i is simply a consequence of the fact that the values indexing subintervals are natural integers, thus being unique. In the second case, to each point of P_i corresponds another point located at a fixed distance. More formally:

$$\forall P_i \in D_A(t), \exists Q_j \in D_B(t), j - i = \Delta, \Delta = Const. \tag{4.2.1}$$

Since Δ is fixed, and since all interval identifiers are unique, to each value i corresponds only one image $i + \Delta$. This is by definition a bijective function. $\square$

This property simply reflects the fact that the number of values reflecting the number of occurrences is the same for message types. Intuitively this means that since the number of sub-intervals is the same for all flow density functions, then the non-null values will be in the same number too (See Definition 9 and Proposition 2). Each time we will write $(P_i, Q_i$, it will be assumed that the bijection has been applied between the two points P and Q. We will say that P_i and Q_i are *corresponding points*. The bijective property is important because it allows to determine whether a given PLF can be obtained by the affine transformation of another.

Nevertheless, the bijective function that correlates points of two flow density functions into unique couples of points needs to be defined according to our needs. Consider for example the two flow density functions illustrated in Figure 4.1. There is no apparent way of determining the function that associates $D_A(t)$ to $D_B(t)$. If an AT test takes place by simply coupling the points of these two functions based on the same interval value, the test will obviously fail. Yet, there is an affine transformation that perfectly matches $D_A(t)$ into $D_B(t)$.

The solution to this problem comes from what we will call in this manuscript the *"spike"* phenomenon. A spike is observed as a sharp and considerably important change between two consecutive values of a FD function. Spikes occur naturally during the activity of a Web service, and are most often due to sudden changes in the traffic load of a Web service server. The most known (and worst) occurrence of spikes are bottlenecks in a server, but they generally occur for many reasons. For example, it is easy to explain why the traffic of users exchanging messages with the server of a stock exchange broker drops almost dead after 8 o'clock PM. Once the stock market closes at this hour, no service is available on the stock values (except services running for maintenance and backup purposes, not available to common client brokers).

Let us detail the procedure that allows to define the correlation function $i : D_A(t) \times D_B(t) \rightarrow (P_i, Q_j)$ by employing the spike events. First, we detect the highest differences between two consecutive points of $D_A(t)$. These differences will be marked with the δ symbol in the upcoming illustration. Then, we identify the intervals between which these extreme differences of flow density values occur. We compute the distance (in number of subintervals) separating the most important spikes. Then we iterate the same sequence of operations for $D_B(t)$. If the distance between spikes for $D_A(t)$ is the same as those for $D_A(t)$, then this distance will define $i : L_A \times L_B \rightarrow (P_i, Q_{i+\Delta})$, where Δ corresponds to the computed distance between spikes. Note that using L_A and $D_A(t)$ is equivalent, since the value of L_A at a given row is equal to the image of the same subinterval value of $D_A(t)$. The newly defined

function means that at a point P of $D_A(t)$ whose x coordinate is a ($P_x = a$), corresponds the flow density value Q of $D_B(t)$ at the subinterval $a + \Delta$ ($Q_x = a + \Delta$). The whole process of determining the δ and Δ values is illustrated in Figure 4.2. On the other hand, if the spikes of $D_A(t)$ are entirely disconnected from the spikes of all other flow density functions, then A is marked as a potential candidate for a loop.

One should beware of an incorrect interpretation that might lead to a misunderstanding of the correlation function. Detecting the order of spikes will never allow to determine the order relationships between message types. First, the distance between two spikes is accounted for, even if one of the spikes is a positive difference, and the corresponding spike on the other function is a negative difference. Second, even if spikes are of the same sign of difference, detecting that the spikes of $D_A(t)$ occur before their corresponding spikes on $D_B(t)$ is by no means sufficient to state that the PLF of A can be obtained from an affine transformation of the PLF of B. Figure 4.2 is an obvious counter-example of such a misunderstanding.

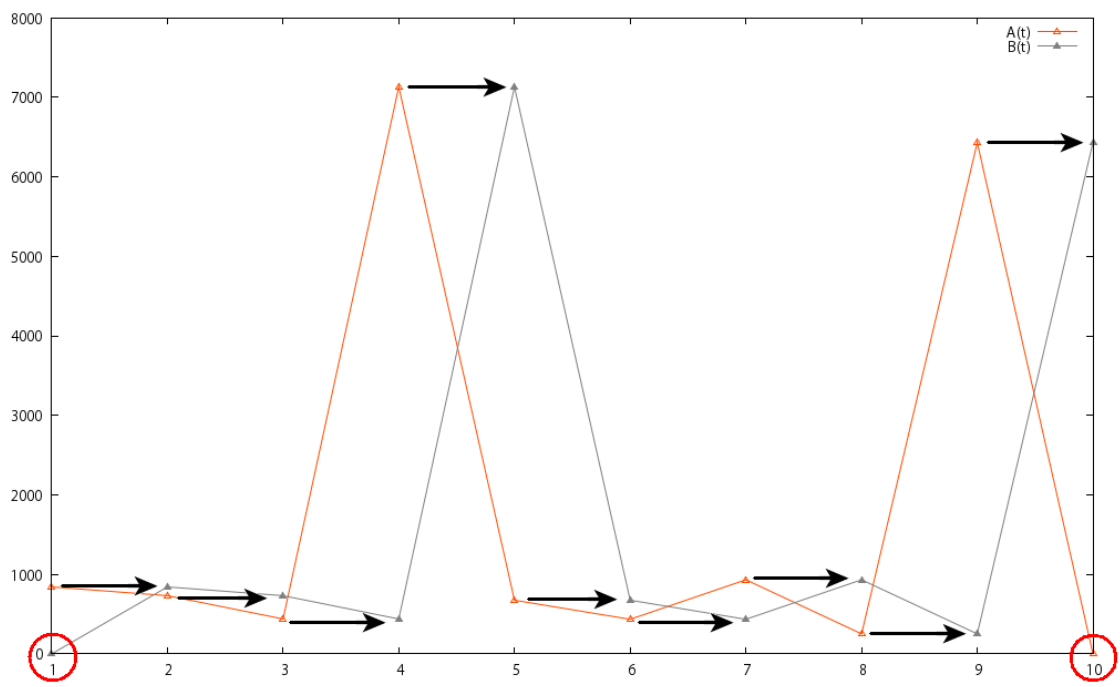


Figure 4.1: Illustration of the problem of correctly defining the bijective function that correlates points for the assessment of an affine transformation between two flow density functions. The points circled in red need to be excluded and express the shift to be taken into account by the bijective function.

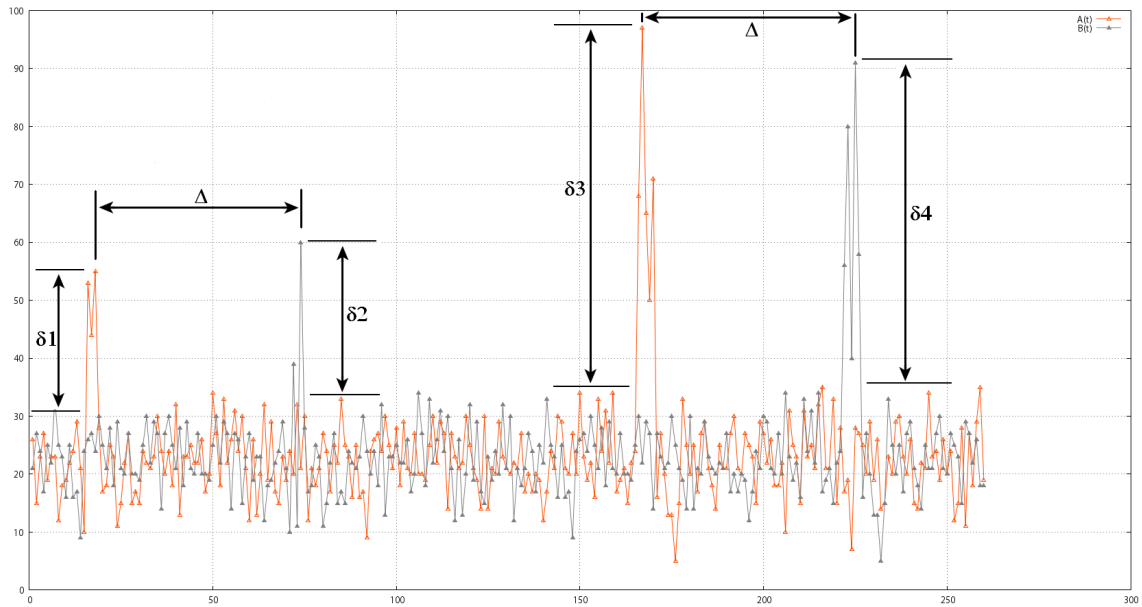


Figure 4.2: Illustration of how spikes allow to define the distance separating points of two flow density functions that are to be correlated.

Based on Proposition 3, we know that to the function of flow density D of a given message type corresponds a unique PLF. In other words, each PLF describes the temporal and quantitative dynamics of a message type. Therefore, the problem of correlating message types m_1 and m_2 can be formulated as:

"Establish whether the PLF of m_2 can be obtained from a translation transformation of the PLF of m_1 (or vice-versa)".

Euclidean geometry provides many translation transformations. Not all of these allow to reach our objective. The transformation that we need should be capable, in principle, to shift a PLF in the direction given by a two-dimensional vector. Simply put, all the coordinates of the points of a PLF will be modified according to fixed values. By doing so, the distance between the points of a PLF will remain the same, and consequently, the gradient of every segment of the PLF will be the same too. The *affine transformation* provides such capability. Figure 4.1 that we have already presented shows also a particular case example of an affine transformation.

Equation 4.1.3 in Definition 11 provides us with a straightforward way to assess whether a given PLF may be the result of the AT of another PLF. To achieve this we proceed as follows. We take two corresponding points of type T_2 time series,

$X = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \in D_A(t)$, $Y = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \in D_B(t)$. From points X and Y we compute $S_x = x_2 - x_1$ and $S_y = y_2 - y_1$ according to Definition 11. Then, for all points $X \in D_A(t)$, we compute the AT of those points using Equation 4.1.3.

$$\begin{pmatrix} x_3 \\ y_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & S_x \\ 0 & 1 & S_y \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \tag{4.2.2}$$

If $D_B(t)$ can be expressed as an AT of $D_A(t)$ (or vice-versa), one would expect that $Z = \begin{pmatrix} x_3 = x_2 \\ y_3 = y_2 \end{pmatrix} = Y$, within a certain tolerance degree in order to count for potential imperfections.

In the case of business protocols, several types of AT may occur. We define these types as follows.

Definition 12 Let $D_A(t)$, $D_B(t)$ be two time series describing the values over time t of the flow density functions of A and B. $D_A(t)$ and $D_B(t)$ will be labeled *elementary* flow density functions. If verified, the AT between $D_A(t)$, $D_B(t)$ is a **first-order affine transformation (FOAT)**. A $FOAT$ is defined as an assessed and verified AT between only two elementary $PLFs$.

Definition 13 Let $D_A(t) = D_{A_1}(t) + ... + D_{A_m}(t)$, $D_B(t) = D_{B_1}(t) + ... + D_{B_n}(t)$, $m, n \in \mathbb{N}$, be the flow density functions defined respectively as the sum of the flow density functions of message types A_1 through A_m and B_1 through B_n. $D_A(t)$ and $D_B(t)$ will be labeled a *composed* flow density functions. A **multiple-order affine transformation (MOAT)** is defined as an assessed and verified AT between two additive groups of $PLFs$, in this case one between $D_A(t)$ and $D_B(t)$.

One should observe that the sum of two or more FD functions corresponds to the FD of a message type that is equivalent to the merging of the initial message transitions.

Note: If $m = n = 0$ then we see that a $FOAT$ is a particular case of a $MOAT$.

Flow density and granularity level effects

According to Definition 9, flow density is defined as the ratio of the number of occurrences counted for a message type, per time interval. This time window on the other hand has an arbitrary length. It might have a length value of two seconds or two days.

It is obvious that this will have drastic effects on the flow density values. Moreover, when considering a message log of a web service, the message labels and timestamps cannot be used directly. Algorithm 4.2.1, illustrated below, is mandatory for obtaining the flow density values, while accounting for the desired granularity level, i.e. the length of the time window. The most important parameter of the algorithm is the variable $IntervalLength$.

It is obvious that the smaller the window size, the greater will be the number of values of the flow density functions. A very small sub-interval length will yield a high frequency-like signal function. In contrast, when the interval size increases, the PLF will continue to flatten until, at a given value (only 2 intervals are computed), the PLF becomes a purely linear function defined by one single equation on the considered timeline. Both of these two extreme situations are to be avoided during the discovery process. The first case (minimal length) increases the computation time of D in a futile way. This time may be potentially very high for very long duration timelines (months, years). The second case (maximal window length) will provide an over-simplifed version of D that is useless for determining the temporal order of messages. Nevertheless, variations of these cases present advantages as we will see in the following. Figure 4.3 provides a very clear illustration on the impact that the number of sub-intervals have on the output of the Algorithm $computeFlowDensityData$. This Figure plots the flow density function of the message type *login* of the $Trading$ Web service (shown in Figure 1.4). The distinction is obvious between Figure 4.3(a), where the number of sub-intervals is set to 195, and Figure 4.3(b), where the same parameter is set to 20. In the second case, assessing the affine transformation with another flow density function will be far easier, but detecting the temporal order will prove to be quite difficult, whereas in the first case we have the opposite situation.

Definition 14 Let W_s be the time window size and T_t be the average transition time of messages during the execution of the web service. The **zoom value (Z_v)** of the flow density time series is defined by the following ratio:

$$Z_v = \frac{T_t}{W_s} \tag{4.2.3}$$

Z_v has a high impact on the result of the log analysis. If the window size value W_s is lower than the time required for a transition, then the horizontal shift will be proportional to Z_v. In other words, the horizontal shift will increase proportionally with Z_v. This proportionality is always valid.

By employing a relatively high value of Z_v (denoted HZ_v) it is possible to determine the temporal order between two $PLFs$ involved in a $FOAT$. This is achieved thanks to the horizontal shift that becomes visible at high ranges of Z_v. Moreover, using the HZ_v range enables the recognition of the first message type that is issued at

Algorithm 3 *computeFlowDensityData*

Require: The array of message occurrence vectors $VectorsArray$
Ensure: The matrix FD with the flow density data of all messages

```
IntervalLength = 1000 //length of time window
NumberOfVectors = length(VectorsArray)
CVM = createCompleteVectorsMatrix(VectorsArray);
Min = min(min(CVM))
Max = max(max(CVM))
min = Min
for VCounter = 1 to NumberOfVectors do
    min = Min
    max = min + IntervalLength
    SingleVectorSize = size(VectorsArray(VCounter))
    SingleVectorSize = SingleVectorSize(1, 1)
    MCounter = 0 //Message counter
    NextRowIndex = 1
    ICounter = 1 //Interval counter
    while NextRowIndex ≠ (SingleVectorSize + 1) do
        CurrentVector = CVM(NextRowIndex, VCounter)
        if CurrentVector ∈ N then
            if min ⩽ CurrentVector ⩽ max then
                OccurrenceValue = CurrentVector
                MCounter = MCounter + 1
                NextRowIndex = NextRowIndex + 1
            else
                FD(ICounter, VCounter + 1) = MCounter
                MCounter = 0
                ICounter = ICounter + 1
                min = max + ϵ
                max = min + IntervalLength
            end if
        end if
    end while
end for
```

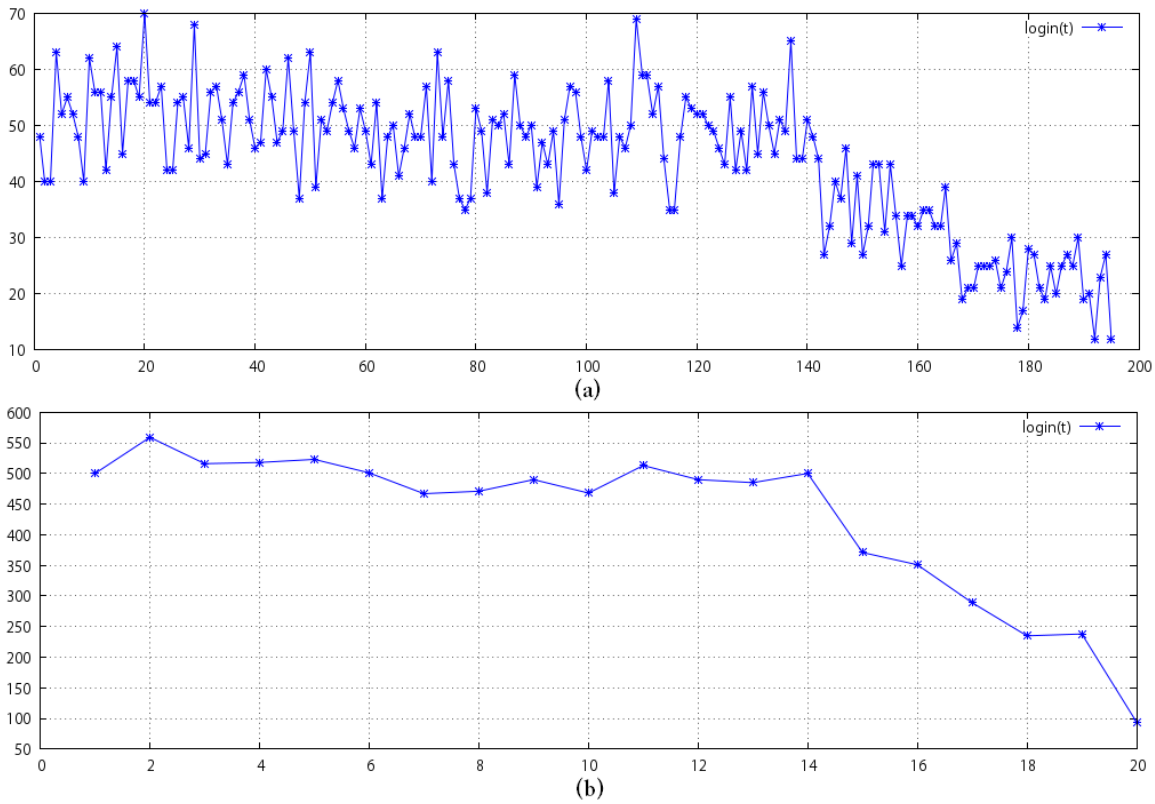


Figure 4.3: The impact of the number of sub-intervals($\#S_{Int}$) on the flow density functions. The PLF is (a) much more detailed with $\#S_{Int} = 195$, than with (b) $\#S_{Int} = 20$. Observe that the number of message occurrences (y axis) is inversely proportional to $\#S_{Int}$.

the beginning of the service execution. Consequently, this provides the possibility to calculate the relative T_0 time instant. The T_0 time instant denotes the starting point of the service instance execution. Determining the value T_0 in a deterministic way is extremely important since it makes possible to (i) compute the values of the T_1 type time series, and (ii) to use existing approaches that do not have this capability [130].

On the other hand, using a relatively low or medium value of Z_v (denoted respectively LZ_v and MZ_v) allows to determine the affine transformations with a high confidence level. Except some variations, two main scenarios encompass the possible cases of affine transformations:

1. Purely horizontal shift of two flow density $PLFs$ at MZ_v or LZ_v ($S_x \neq 0$, $S_y = 0$). This identifies message types with the same occurrence cardinality but with a temporal latency. Only $FOAT$ linear sequences have this pattern signature. An illustration

of this case was already provided in Figure 4.1.
2. No AT can be directly assessed between two given $PLFs$ unless an extremely LZ_v is employed. The segments composing the two $PLFs$ might not necessarily have the same gradient. Virtually, only $MOATs$ can be detected between FD functions with a satisfactory confidence level.

An exception may be the case in which $S_x = 0$, $S_y = 0$ yet there is a $FOAT$ between two elementary $PLFs$. If such a pattern is recognized, it is a proof of log incompleteness.

4.2.2 Time series analysis for temporal graph extraction

Let us now introduce the temporal operators that will be used to determine the order relationships between messages involved in $FOATs$ and $MOATs$.

Definition 15 Let P be a business protocol. Let ML be the log of messages of P collected during its execution. Let $D_A(t)$ and $D_B(t)$ be two $PLFs$ and $ATM = \begin{pmatrix} 1 & 0 & S_x \\ 0 & 1 & S_y \\ 0 & 0 & 1 \end{pmatrix}$ the affine transformation matrix assessed between the two $PLFs$.

1. If $D_B(t) = ATM \times D_A(t)$ and $S_x > 0$ then $D_A(t)\ BEFORE\ D_B(t)$
2. If $D_B(t) = ATM \times D_A(t)$ and $S_x < 0$ then $D_A(t)\ AFTER\ D_B(t)$
3. If $D_B(t) = ATM \times D_A(t)$ and $S_x = 0$ then $D_A(t)\ IDENTICAL\ D_B(t)$

Allen introduced in [5] thirteen temporal operators to express all the possible relationships between events. Then why do we limit to three the number of operators used in the present approach? In order to understand this, we recall that the Allen operators are applied on sequences of occurrences. We use a specifically tailored definition of three of the original operators, namely B (BEFORE), A (AFTER), and I (IDENTICAL). Our definition is not based simply on the comparison of relative clock timestamps but on the parameters of the affine transformation itself. In other words, we do not determine the temporal relationships directly on the events (messages in our case). Instead, we assess the temporal relationships between FD functions. When considering the order of FD functions, only three cases are sufficient to relate any $PLFs$ in binary ordering. In addition, the original Allen operators describe the relationships between time series of type T_1, while in our case, we need to correctly order time series of type T_2. This limits the number of required operators to three, as opposed to the original thirteen, thus achieving a highly accurate result with a much lower complexity.

Temporal analysis for digraph-modeled service protocols

We recall that a *digraph (DG)* is a graph that may contain circuits and cycles, but cannot contain loops. The most important part of digraph extraction is the recognition of sequential patterns and their correct reconstruction. If message types A and B are located in a linear sequence, then two consequences follow: (i) the number of occurrences of A equals that of B, and (ii) all occurrences of A take place $BEFORE$ their corresponding occurrences of B. In such a case, $D_A(t)$ and $D_B(t)$ will yield identical $PLFs$ except that $D_B(t)$ will be shifted rightwards of $D_A(t)$. It should be obvious that this shift is due to the elapsed time between the occurrence of A and B, since no transition is instantaneous. This case is identified by the fact that $S_x > 0$ and $S_y = 0$. In short, a purely linear sequential pattern of two messages will be identified by a horizontal shift at MZ_v or HZ_v.

Nevertheless, detecting and reconstructing linear sequence patterns depends on the type of the considered digraph. Major distinctions exist between two main types of structures that generate patterns of different sequence length behavior. The following example illustrates this point.

Consider the protocol graph shown in Figure 4.4 (a). It can only generate (exclusively) sequences of message occurrences of fixed length. The protocol graph in Figure 4.4 (b) on the other hand, can yield sequences of variable length, due obviously to the presence of a cycle. This deeply affects the required approach. The second case being a more complex one than the first, we focus on the distinction between two categories: the first is the DAG type, and the second is the more generic DG type. When we consider a DAG, we immediately observe that this graph type contains also sequences that are not linear (i.e. of the form $a_1 \rightarrow a_2 \rightarrow a_3$). For example the fragment in Figure 4.4 (a) $b \rightarrow d \rightarrow e$ is not linear. In order to understand how to deal with the general pattern scenario, we recursively analyze the elementary case of linear DAG, and then apply deductions to the more generic digraph type. This elementary case of purely linear sequence graph is defined as follows.

Definition 16 Let $G = (V, E)$ be a graph. G is a directed acyclic graph of type DAG^2 if $\forall v \in V$, $d(v) \leqslant 2$.

In other words, for each node of a DAG^2 graph, there must be at most two edges with other nodes of the graph. Unless we explicitly write the maximal degree allowed for all nodes in the graph, by means of the preceding notation. The reader should assume that the DAG abbreviation refers to its normal definition without any degree restriction.

Lemma 3 *Let $\mathcal{P}$ be a protocol modeled using a DAG^2, in which message occurrences are recorded using timestamps of type T_2. Consider the order operator*

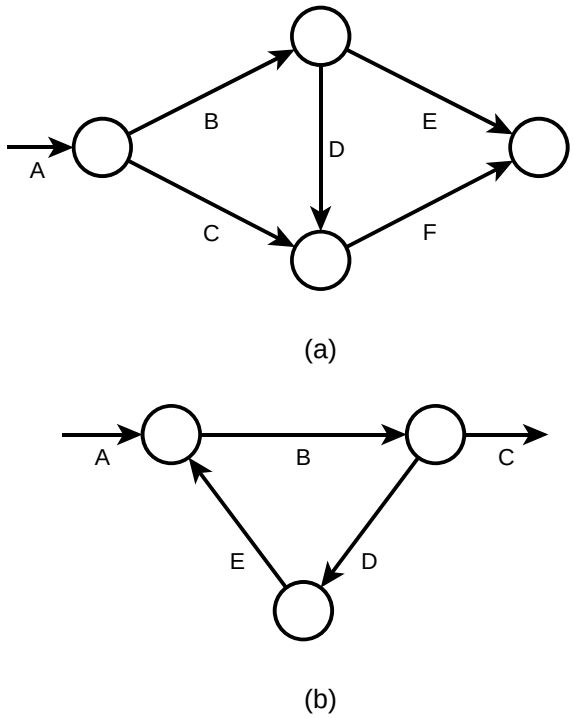


Figure 4.4: Slight changes in the FSM to be discovered may have consequences on the difficulty level of the task. FSMs with less connections between states are not necessarily the easiest to infer, despite their intuitively simpler structure.

$BEFORE$ (also denoted $<$), its inverse $AFTER$ (denoted $>$), and the equality operator $IDENTICAL$ (denoted $=$). Let $\mathcal{S}$ be the set of flow density functions associated to each message type of P. The order operator $<$ applied to the elements of $\mathcal{S}$ defines a strict order. In addition, this strict order is total.

Proof. A formal proof of this lemma would require a formal check of the properties of transitivity, antisymmetry, and totality. In our case, a simpler proof is provided by the structure of a DAG^2. Since in a DAG^2 all message types are already ordered in a linear sequence, separated by vertices, then the temporal operators in Definition 15 also define a strict and total order between the message types of a DAG^2. □

Lemma 4 *Consider the set of message types $\mathcal{M}$. The order operator $<$ appplied to the elements of $\mathcal{M}$ defines a total and strict order.*

Proof. Property 2 proves the existence of a surjective function between $\mathcal{S}$ and $\mathcal{M}$ (via the existing transitivity property $\mathcal{S}$, $\mathcal{D}_{m,m\in\mathcal{M}}$ and$\mathcal{M}$). Moreover, $\mathcal{S}$ and $\mathcal{M}$ are linearly ordered sets. Hence, $\mathcal{S}$ and $\mathcal{M}$ are order isomorphic, and therefore have the same order type. $\mathcal{S}$ being strictly and totally ordered, one can state the same thing about $\mathcal{M}$ (Lemma 3). □

Lemma 5 *Consider $\mathcal{P}$ a protocol modeled using a DAG^2 and $\mathcal{M}$ the associated set of message types. Then, $(\mathcal{M}, <)$ divides the elements of $\mathcal{M}$ into a partition.*

Proof. The proof of this lemma is supported by Lemma 3. Indeed, since the order in a DAG^2 is strict and total, the sets containing each a given message type of this graph type will form a partition. If this was not the case, then it would imply that the message types in a DAG^2 are not strictly and totally ordered, which is a contradiction. □

Notice that in the case of Lemma 5 each set of the partition is composed of one single message. This lemma appears to be too obvious, yet we will immediately see why this partitioning of $\mathcal{M}$ is fundamental.

Definition 17 Let $G = (V, E)$ be a graph. A super-node (abbreviated s-node) is a newly created node, whose label encompasses two or more nodes $v_i \in V$.

Definition 18 Let $A_1, \cdots, A_m \in \mathcal{M}$ be m message types. An *abstract message type* $\mathcal{A}_\mathcal{M}$ is a subset $A_i \subseteq \mathcal{M}$.

In other words, an abstract message type is a label identifier for a group of message types. All the properties of individual messages composing an abstract message are thus inherited by the latter. Occurrence log values of the elements of $\mathcal{A}_\mathcal{M}$ are summed and their timestamps of occurrences are merged together.
Remark: The label of an abstract message type is also the label of an s-node.

Theorem 6 *Let $G = (V, E)$ be a DG^{2+} ($G \in DG^{2+} \leftrightarrow G \in DG - DG^2$). There exist (i) a minimal set of nodes and s-nodes, denoted $\mathcal{S}_{min}$, (ii) a corresponding minimal set of message types and abstract message types, denoted $\mathcal{M}_{min}$ such that G can be decomposed into purely linear sequences.*

Proof. This theorem is a direct consequence of the model based on simultaneous linear equations, that was introduced in the previous chapter. Indeed, each linear equation of a simultaneous linear system represents a linear sequence. On the left side of such an equation we have a message type, and on the right side of the equation we have either a message type, or an abstract message type. The latter is formally defined in Definition 18 as a set of two or more message types. In other words, once that the simultaneous linear system of a protocol is constructed using the *delta* algorithm, we immediately obtain the decomposition of G into purely linear sequences, as described in this theorem. □

Definition 19 Let $G = (V, \mathcal{M})$ be the *DG* modeling a protocol $\mathcal{P}$. The *temporal graph* $T = (\mathcal{S}_{min}, \mathcal{R})$ of G is a potentially non-connected, directed graph defined as:
- $\mathcal{S}_{min} = \{s | s \in \mathcal{M}\}$: the set of nodes and s-nodes. The labels of nodes and s-nodes are subsets of message labels. The edges of G are transformed into element nodes in T.
- $\mathcal{R}$: the set of arrows connecting those elements of $\mathcal{S}_{min}$ between which the temporal order relation $(<, >, =)$ is defined.

Corollary 4.2.1. *Consider G of Theorem 6. If G is a DAG^2 then $\mathcal{S}_{min} = V$*

Corollary 4.2.2. *Consider G of Theorem 6. The order $<$ between the nodes in each linear sequence of the decomposed graph G is strict and total.*

Corollary 4.2.3. *Let $T = (\mathcal{S}_{min}, \mathcal{R})$ be a temporal graph as defined in Definition 19. The linear sequences stated in Theorem 6 are identifiable as the connected subgraphs (connected components) of T.*

Definition 20 Let $T = (\mathcal{S}_{min}, \mathcal{R})$ be the temporal graph of the digraph $G = (\mathcal{V}, \mathcal{M})$ modeling a protocol $\mathcal{P}$. The *order matrix* M_R is the equivalent array notation of T. Formally speaking:

$$M_R(i, j) = x, x \in \{A, B, I, U\}, i, j \in \mathcal{S}_{min} \tag{4.2.4}$$

The order matrix M_R is very similar to the adjacency matrix of T, but it differs in the sense that it provides not only the information on the existence of edges (0/1 as in the case of the adjacency matrix of an (non-oriented, non-directed graph) between nodes and s-nodes, but also the exact order operator validated between two (s-)nodes. M_R is very useful to compute and visualize in a compact format the result of the detection of ATs between $PLFs$ and also the order relationships between the sets of nodes.

Lemma 7 *Let $G = (V, E)$ be a DG^{2+}. The repartition of $v \in V$ into the elements of $\mathcal{S}_{min}$ is not a partition.*

Proof. It is sufficient to employ the following counter-example: A message type may often be common to two equations in an SLE. Thus, this message type would be an element of more than one set of the partition, which contradicts the initial hypothesis of the existence of a partition of $v \in V$ into the elements of $\mathcal{S}_{min}$. □

Lemma 8 *Let $\mathcal{S}_{min}$ be the minimal set of nodes and s-nodes of a DG^{2+}. $(\mathcal{S}_{min}, <)$ is not a totally ordered set.*

Proof. The comparability condition is not satisfied since in a DG^{2+} we encounter structures, such as for instance composed loops displayed in Figure 3.1, in which message types may not always be compared using the three order operators AFTER, BEFORE, and IDENTICAL. □

Property 1 and Lemma 3 allowed us to deduce Lemma 4, which in turn shows that once $FOATs$ are assessed, message types can always be strictly and totally ordered. Lemma 5 proves that this method solves the case of DAG^2 discovery. Lemmas 7 and 8 clearly state that a DG cannot be discovered by assessing the discovery of $FOATs$ only, and that detecting and mining $MOATs$ is mandatory for a complete DG extraction. Theorem 6 provides the solution for extending the solvability proof of DAG^2 to DAG^{2+}. Corollary 4.2.1 of Theorem 6 unifies the elementary case with the generic case of the recursive study of DG mining analysis. Corollary 4.2.2 of Theorem 6 assures us that despite the lack of a partial order between the elements of $\mathcal{S}_{min}$ (proven in Lemma 8), the message types in the linear sequences are strictly and totally ordered. In conclusion, if we succeed in obtaining $\mathcal{S}_{min}$ and detecting $MOATs$, Theorem 6 and its Corollary 4.2.2 provide us with the theoretical framework that will enable the total discovery of a DG.

A DG may contain cycles that disrupt the total and strict order defined by $<$. To understand this, let us consider once again the cycle in Figure 4.4 (b). From this figure we can write $\{a\} < \{b, d, e\} < \{c\}$ and $\{d\} < \{e\}$, but nothing can be stated on the cycle $b \rightarrow d \rightarrow e$ without prior knowledge of this protocol graph fragment. If we check the affine transformations of first order between a, b, c, d and e, the detection will result in only two ATs that can be assessed: $\{a\} < \{c\}$ and $\{d\} < \{e\}$. This is quite visible when the $PLFs$ of $D_{\{a\}}(t), D_{\{c\}}(t), D_{\{d\}}(t)$ and $D_{\{e\}}(t)$ are plotted from experimental data in Figure 4.5. Notice in this figure that since there exists a shift between flow density functions, the functions do not match on the entire value interval, but this is not a sufficient reason to invalidate the existence of an affine transformation.

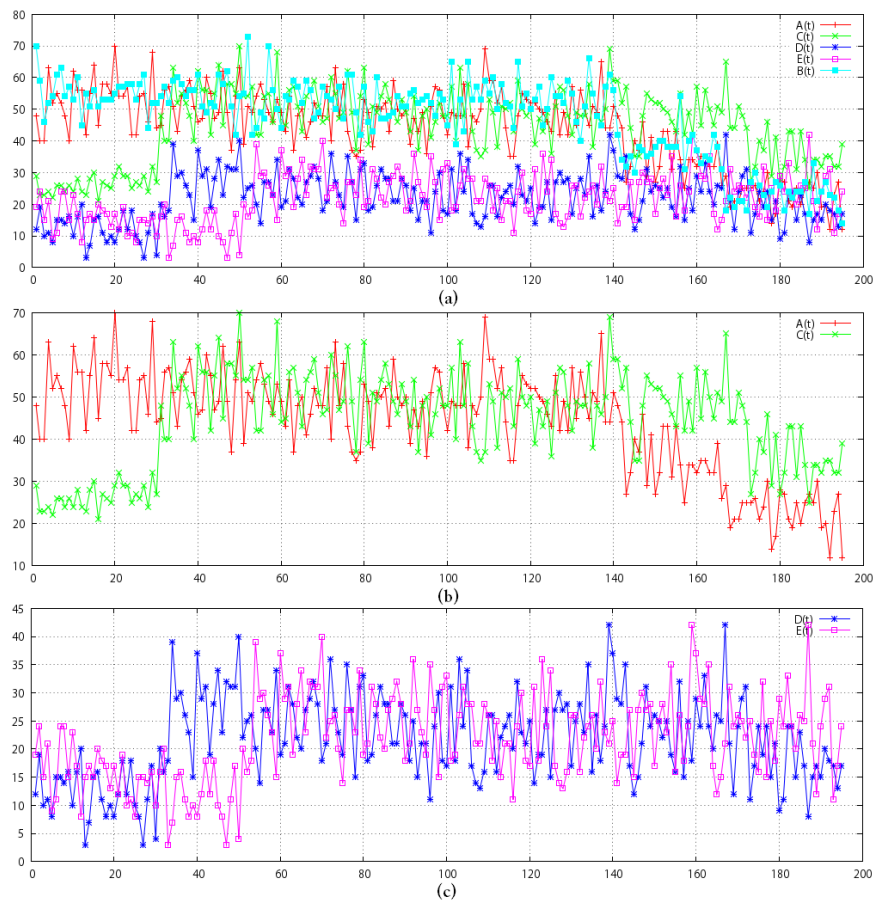


Figure 4.5: Illustration of (a) density flow functions of messages a, b, c, d and e, (b) assessment of a $FOAT$ between a and c, (c) $FOAT$ validation between d and e, issued from benchmark service protocols.

At this stage we know how to detect $FOATs$ starting from flow density data, but acquiring the targets of $MOAT$ candidates remains an open issue. Let us focus once again on Definition 13: (i) *"A MOAT is defined as an assessed and verified AT between two additive groups of PLFs..."* and (ii) *"$D_A(t)$ and $D_B(t)$ will be labeled a composed flow density functions"*. The first sentence indicates that a $MOAT$ will be tested only in the presence of at least one flow density (FD) function which is the sum of two or more elementary flow density functions. Obviously, an elementary flow density function is simply a function corresponding to a single message type. A composed flow density function is thus the algebraic sum of the flow density values of several message types. This algebraic addition is made according to the bijective function $i : D_1(t) \times D_2(t) \rightarrow (P_i, Q_i)$, introduced in Property 2, that correlates all the points of the flow density functions that need to be summed together. The second sentence of the definition citation tells us that the $MOAT$ will be valid as long as the additive groups of $PLFs$ are considered as such, i.e. splitting the composed PLF will make a $MOAT$ void and null. If we employ the terminology of Definitions 17 and 19, we may state that a $MOAT$ will define the edge label (i.e. the temporal order operator) between a s-node and a node, or between two s-nodes. Here we see the strong equivalence connection between a s-node, a composed flow density function, the connected components of T in Corollary 4.2.3, and the linear sequences of a decomposed protocol graph in Theorem 6. Indeed, a s-node corresponds exactly to the set of message types that define the flow density functions that are summed to form a composed FD function. For example, if $\Delta = \{A, B, C\}$ is a super-node then we have unavoidably $D_\Delta(t) = D_A(t) + D_B(t) + D_C(t)$. Moreover thanks to Corollary 4.2.3, we see that a couple s-node/s-node or node/s-node connected via an edge, is the inverse graph notation of a linear sequence. For example, if we consider the simple temporal graph in Figure 4.6 (a), in which A and B are two s-nodes, and the linear sequence constructed from $\mathcal{S}_{min}$ shown in Figure 4.6 (b), then it is clear that the two notations are symmetrically the same.

Proposition 3 states that the PLF obtained from a flow density function $D_M(t)$ is continuous, and also differentiable except at the definition points of $D_M(t)$. This means that a composed flow density function will also yield a composed PLF having the same properties as an elementary PLF. As a result of the whole analysis, we deduce that a $MOAT$ can be expressed and tested in exactly the same way as a $FOAT$. Only in this case, if we write $MOAT(\{A, B\}) \leftrightarrow FOAT(\{A, B\})$, one must not lose sight of the fact that A and B can never be simultaneously message types in the same expression. At least one of them is the label of a s-node, if not both. An illustration is depicted in Figure 4.7 based on the DG protocol fragment provided in Figure 4.7 (b). The computation method for assessing a $FOAT$ is therefore identical to the method for validating a $MOAT$. Nevertheless, one must keep in mind that the semantics associated to the parameter labels are strictly different.

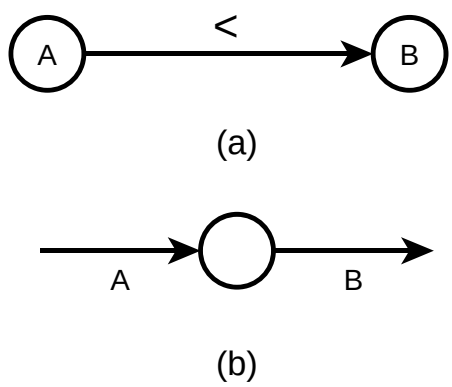


Figure 4.6: Example of (a) two nodes in a temporal graph ordered via the label of the edge that connects them. The direction of the arrow indicates the order to be applied. (b) the linear sequence in $\mathcal{S}_{min}$ that is equivalent to the connected component.

One last obstacle needs to be addressed in order to complete the basis of DG discovery. In order to validate the existence of $MOATs$ from flow density data, we need to compute the composed $PLFs$ by adding the elementary ones. Given n message types for a protocol that is to be discovered, the complexity of computing all the possible composed flow density functions is O(2^n). Therefore, computing all the combinations of additive functions is definitively not a good idea. In the case of $FOATs$ on the other hand, the number of AT tests for 1-to-1 ordering is $\frac{n(n-1)}{2}$ thus yielding a complexity of O(n^2) which is an acceptably low value for $n = 50$. But then, how to identify the exact additive functions to be used? What defines the only $MOAT$ candidates that are worth to be tested? This is where the *delta* algorithm, introduced previously, comes into play. We recall that the equations of an SLE established through *delta* describe the quantitative dynamics of the flow of message occurrences when all logged instances of a Web service are merged together. An equation is the equality between the sets of message occurrences. When we discussed the granularity level effects in Section 4.2.1, we stated that horizontal shift alone of 2 $PLFs$ at MZ_v or LZ_v ($S_x \neq 0, S_y = 0$) identified equal occurrence cardinalities shifted because of a temporal latency. This is the unique pattern of $FOAT$ linear sequences. As a consequence, obtaining the equations that provide the correct combinations of message types having equal occurrence cardinalities puts us in the optimal position for assessing MOATs between the PLFs of abstract messages.

Let us show with an example how delta solves the problem of obtaining s-nodes from abstract messages, therefore allowing us to check for MOATs and define all the temporal order operators existing between nodes and s-nodes. Consider the protocol shown in Figure 4.8. Notice that this example contains a cycle: ($F \rightarrow E \rightarrow D$), and

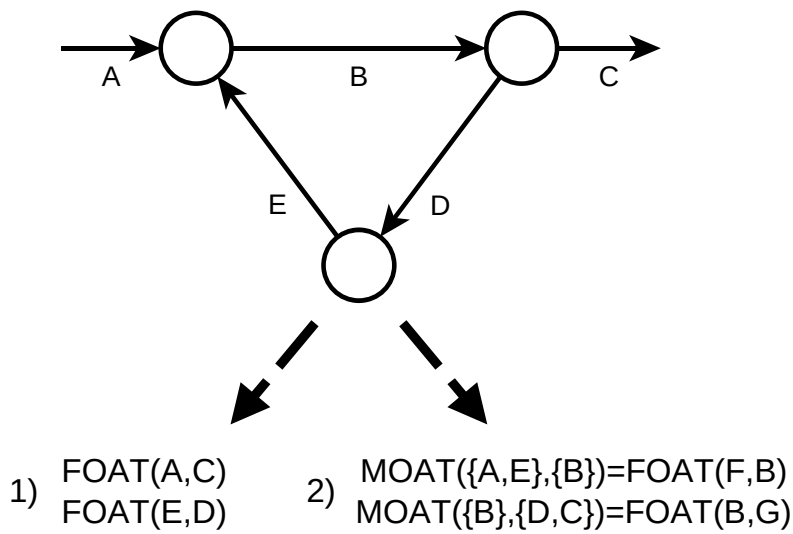


Figure 4.7: An illustration of the computational equivalence between a $FOAT$ and a $MOAT$. Once that abstract message types (labels) are introduced in the notation, only the semantic difference between the label of a message type and abstract message type of a s-node remains.

a circuit: $(B \rightarrow E \rightarrow C)$. Thus the protocol has the complete profile of a DG despite its simplicity.

First let us present the discovery result of delta upon this protocol. The resulting SLE is provided in Equation 4.2.5. If we run our method on the flow density data of message types, only one FOAT candidate will be eventually assessed: $\{A\}, \{G\}$. We will now focus on how delta reveals the MOAT candidates.

$$\begin{cases} 1.\ \bar{A} = \bar{B} + \bar{C} \\ 2.\ \bar{C} + \bar{F} = \bar{E} \\ 3.\ \bar{B} + \bar{E} = \bar{D} \\ 4.\ \bar{D} = \bar{F} + \bar{G} \end{cases} \tag{4.2.5}$$

As a matter of fact, MOAT candidates are immediately revealed by the equations composing the SLE in Equation 4.2.5. In other words, each equation indicates the message types, and therefore the corresponding flow density functions that need to be merged in order to obtain such PLFs that only horizontal shifts (within the appropriate Z_v range) will be observed. One must be careful not to erroneously consider (or interpret) this procedure as one of data adaptation. Employing data for obtaining

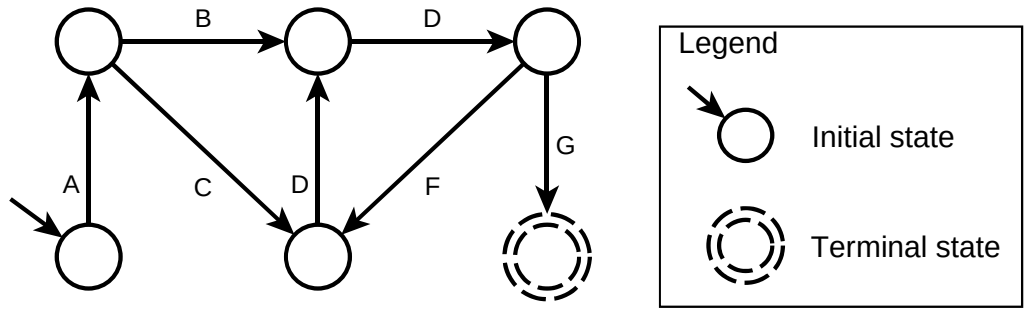


Figure 4.8: A protocol sample modeled by a DG, thus containing a circuit and cycle.

the list of MOAT candidates is a process of pruning the research space of MOAT candidates by an exponential factor, therefore dramatically improving the efficiency of the AT detection solution.

The first step before continuing is to rewrite each equation of the SLE containing a negative-valued variable, such that every single variable has a positive sign. For example, if an equation comes in the form $X = Y - Z$ we rewrite it as $X + Z = Y$. This of course is algebraically the same equation. The rewriting operation is not mandatory, but still it is easier to interpret the abstract messages and s-nodes when their message labels are of the same sign. Moreover, Definition 13 uses positive-valued functions, and in the preceding chapter we saw that employing only positive signs in an equation of a SLE made easy the usage of the conventional identification of messages entering a protocol state (variables on the left of an equation) from those exiting the same state (variables on the right of the equation). Let us continue with our list of MOAT candidates: $MOAT(\{A\},\{B,C\}), MOAT(\{C,F\},\{E\}), MOAT(\{B,E\},\{D\})$ and $MOAT(\{D\}\ ,\{F,G\})$. The next step consists of constructing the equivalent abstract messages. Given an equation, each side of the equation presenting more than one variable will be replaced by a single new variable that will be created and stand for the label of a new abstract message. In this example we obtain Equation 4.2.6.

$$\begin{cases} 1.\ MOAT(\{A\},\{B,C\}) = FOAT(\{A\},\{H\}) \\ 2.\ MOAT(\{C,F\},\{E\}) = FOAT(\{I\},\{E\}) \\ 3.\ MOAT(\{B,E\},\{D\}) = FOAT(\{J\},\{D\}) \\ 4.\ MOAT(\{D\},\{F,G\}) = FOAT(\{D\},\{K\}) \end{cases} \tag{4.2.6}$$

Up to this point, all that remains to be done is to run the algorithm for assessing the FOATs between these variables. This will determine the temporal relations between them, something that the delta algorithm simply couldn't provide. For example

$FOAT(\{A\}, \{H\}) = " < " \leftrightarrow \{A\}\ BEFORE\ \{H\}.$

Temporal analysis for multigraph-modeled service protocols

A *multigraph (MG)* is a digraph that contains loops. Despite being intuitively a slight modification, loops are important in process and service protocol mining. Let us consider the following protocol modeled using the MG of Figure 4.9 in which we introduce a loop transition.

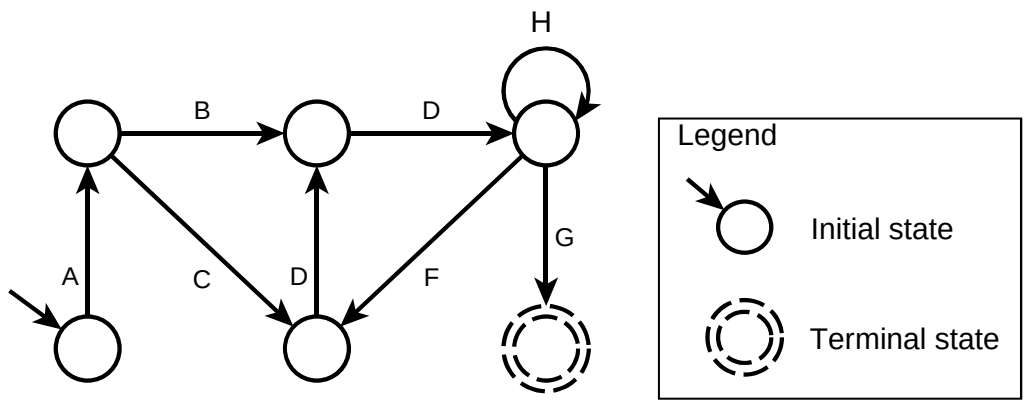


Figure 4.9: A protocol sample modeled by a MG, thus containing not only circuits and cycles, but loops as well.

The existence of loops in a protocol requires a different approach of interpreting the result. The loop transition will be labeled as H. In this case, the creation of a new label required by the existing approach does not solve the problem. This is because the very nature of the loop transition is such that no matter how we sum its occurrence vector with the occurrence vectors of other messages, its values will always be different from all the other $PLFs$. Therefore, no affine transformation can be deduced and only one indirect solution remains. Once that delta detects that H is a loop, we need to establish its precise location in the graph. To achieve this we exploit the influence of the loop on the flow density functions of the messages that are located after the loop node. One must pay attention to the fact that H being a simple loop, i.e. it enters and exits the same state, it can be be connected to any state if we use only the SLE solution, since adding $\bar{H}$ on both sides of an equation works for *every* equation! When the loop message H is followed during the execution of a Web service protocol, it is obvious that it will introduce a time latency between the occurrence time of D (the message type entering the loop node) and the occurrence time of F and G (the message types exiting the loop node). Moreover, This latency

(delay), will necessarily be proportional to the number of times the loop is executed in a row during the execution of a single protocol instance. So, one should expect the PLF of H to delay (shift rightwards) the PLFs of following messages. This may not be visible, unless we use a very high Z_v. With a HZ_v we observe that the flow density of messages following a loop is inversely proportional to the flow density of the loop itself. In addition, the PLF of G will be shifted on the right much more compared to the PLFs of say, A and B. In this case, the value of S_x will be greater and most importantly, even variable. In conclusion, a very high Z_v is needed *only* for locating a loop, but to assess the rest of the protocol, a MZ_v range is needed, otherwise S_x and S_y will be variable. To summarize, we stress that there is no optimal Z_v range. Speaking of such a range is meaningless since different Z_v values must be used according to the objective of each individual step of the Web service mining process.

4.3 Discovery algorithms and experiments

Let us now focus on how the theoretical results of the previous sections are assembled into algorithms, and how the implementations of these algorithms perform during experimentation tests. Algorithm 2 is a central piece of this discovery approach since it handles the assessment of an AT between two message types. In other words, it can provide an answer to the question *"Provided the log data, is a given temporal operator validated between two message types?"* .

Algorithm 4 $checkAffineTransformation$

Require: Flow density log L_A, L_B for message types A and B and the Allen operator code $OperatorCode$.

Ensure: The updated $FOAT$ matrix containing the code value for the Allen operator between A and B.

1: **for all** $(P_i, Q_i) \in i : L_A \times L_B \rightarrow (P_i, Q_i)$ **do**

2: $\begin{pmatrix} R_i \\ 1 \end{pmatrix} = \begin{pmatrix} x_{R_i} \\ y_{R_i} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & S_x \\ 0 & 1 & S_y \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x_{P_i} \\ y_{P_i} \\ 1 \end{pmatrix}$

3: **if** $x_{R_i} = x_{Q_i} \pm \epsilon_3$ **then**

4: $FOAT(A, B) = OperatorCode$

5: **else**

6: $FOAT(A, B) = 0$

7: $Break$

8: **end if**

9: **end for**

Algorithm 3 allows to compute the supposed AT parameters, that will be used as the input of Algorithm 2. In addition, Algorithm 3 addresses all the different scenarios depending on, and according to, the values of S_x and S_y. Finally, Algorithm 4 is employed in order to transform the order matrix into a temporal graph. Obtaining the temporal graph is useful since it provides a far more readable mining result than the array notation.

Algorithm 5 *gamma*

Require: Flow density log L_A, L_B for message types A and B.
Ensure: The assessment of a FOAT between A and B.

1: $SampleSet = selectSampleValues(L_A, L_B)$
2: $P_k = \begin{pmatrix} x_{P_k} \\ y_{P_k} \end{pmatrix} \in L_A$, $Q_k = \begin{pmatrix} x_{Q_k} \\ y_{Q_k} \end{pmatrix} \in L_B$
3: **for all** $(P_k, Q_k) \in SampleSet$ **do**
4: $\quad S_{x_k} = x_{Q_k} - x_{P_k}$
5: $\quad S_{y_k} = y_{Q_k} - y_{P_k}$
6: **end for**
7: $S_x = \overline{S_{x_k}} = \frac{\sum_{k=1}^{\#(P_k,Q_k)} S_{x_i}}{\#(P_k,Q_k)}$
8: $S_y = \overline{S_{y_k}} = \frac{\sum_{k=1}^{\#(P_k,Q_k)} S_{y_i}}{\#(P_k,Q_k)}$
9: *//Variance of S_{x_k} and S_{y_k} are below tolerance margin*
10: **if** $\sigma_{S_{x_k}} \leqslant \epsilon_1$ AND $\sigma_{S_{y_k}} \leqslant \epsilon_1$ **then**
11: $\quad$ **if** $S_x > \epsilon_2$ AND $S_y < abs(\epsilon_2)$ **then**
12: $\quad\quad checkAffineTransformation(L_A, L_B, 1)$
13: $\quad$ **else if** $S_x < -\epsilon_2$ AND $S_y < abs(\epsilon_2)$ **then**
14: $\quad\quad checkAffineTransformation(L_A, L_B, 2)$
15: $\quad$ **else if** $S_x < abs(\epsilon_2)$ AND $S_y < abs(\epsilon_2)$ **then**
16: $\quad\quad checkAffineTransformation(L_A, L_B, 3)$
17: $\quad$ **end if**
18: **end if**

This algorithm is thus not concerned with the discovery process itself, but with the visualization task of the result. The present algorithms were implemented in the Matlab programming language. Several additional algorithms were required in order to complete the central core methods presented here with the required functionalities and input data. Consequently, the entire set was constructed as a Matlab library[1]. Extensive testing was undergone in order to see how the approach in general fitted the discovery expectations. The tests were run on an Intel processor on a virtual machine

[1]The library with the source code of the algorithms and the data samples, along with the Matlab models used, can be downloaded at http://liris.cnrs.fr/kreshnik.musaraj/technology/ws/index.html

with 1 GB of RAM. Table 4.1 shows the results obtained during the experimentations.

Algorithm 6 *buildTemporalGraph*

Require: S_{labels}: the set of labels that will serve as vertexes, which are provided by delta(Occ.log), The order matrix M_R with operator codes for the states.
Ensure: Graph of temporal relations between transition labels.

```
1: G = (Vertexes, Edges)
2: //Each edge will be identified by a label of the form B_{A→B} where A, B are respectively
   the inbound and outbound vertex labels.
3: V = S_labels
4: E = ∅
5: while NextRowIndex ≠ (SingleVectorSize + 1) do
6:    Chose a label l from S_labels
7:    for all k ∈ S_labels, k ≠ l do
8:       if M_FOAT(k, l) = A OR M_FOAT(l, k) = B then
9:          E = E ∪ {B_{l→k}}
10:      end if
11:   end for
12:   Remove l from S_labels
13: end while
```

Measuring and assessing the performance of the *gamma* approach is more complicated. This is because, as we saw in the previous sections, there are three different variables that evolve simultaneously and independently from one another: (i) $\#M_{Type}$, the number of normal and abstract message types, (ii) $\#S_{Int}$, the number of sub-intervals defined in Algorithm 1, that translates the granularity of the flow density data retrieved from message logs, thus directly defining the Z_v range, and (iii) $\#AT_{Eq}$, the number of linear equations provided by *delta* that need to be checked by Algorithm 2 for their temporal relationship.

Figure 4.10 is obtained from plotting the experimental data measured during tests.The data interpretation becomes much clearer, and new conclusions appear when this figure is analyzed. Indeed, all experimental complexity plots remain linear, but we also notice an almost perfect alignment between the functions having the same number of sub-intervals ($\#S_{Int}$), but different numbers of message types ($\#M_{Type}$). One can then immediately deduce that the influence of $\#M_{Type}$ on the overall complexity is extremely limited. Hence, the most important parameter having a consequent impact upon the complexity of *gamma* is $\#S_{Int}$. Meanwhile, we have already explained that $\#S_{Int}$ is in fact the direct reflect of the magnitude of Z_v, so at the end the granularity level will be the most influential factor when *gamma* is deployed on real-world applications. In every case, the complexity remains strictly linear on the magnitude of $\#AT_{Eq}$ (whose values are listed on the x axis of the plot).

Table 4.1: Experimental data

$\#M_{Type}$	$\#S_{Int}$	$\#AT_{Eq}$	**Min.MT(s.)**	**Max.MT(s.)**	**Avg.MT(s.)**
10	100	45	0.02	0.04	0.03
	500	45	0.08	0.10	0.087
	1000	20	0.07	0.09	0.077
		45	0.16	0.20	0.17
25	100	45	0.02	0.04	0.025
		150	0.07	0.12	0.09
		300	0.15	0.18	0.16
	500	45	0.08	0.11	0.09
		150	0.29	0.32	0.30
		300	0.59	0.64	0.60
	1000	45	0.17	0.19	0.18
		150	0.57	0.62	0.58
		300	1.14	1.20	1.16
50	1000	300	1.13	1.18	1.14
		600	2.24	2.35	2.27
		1225	4.38	5.52	4.70
	5000	300	5.76	6.15	6.07
		600	11.97	12.26	12.15
		1225	20.40	31.96	25.45
	10000	300	11.94	12.21	12.09
		600	23.95	26.14	24.76
		1225	43.68	46.23	44.42
100	1000	1225	4.53	8.13	7.03
		2500	9.19	9.35	9.23
		4950	18.07	18.53	18.37
	5000	1225	24.60	25.20	27.80
		2500	47.32	51.75	50.53
		4950	76.32	101.47	88.75
	10000	1225	46.35	49.70	47.63
		2500	96.20	103.71	99.04
		4950	198.56	201.56	199.30

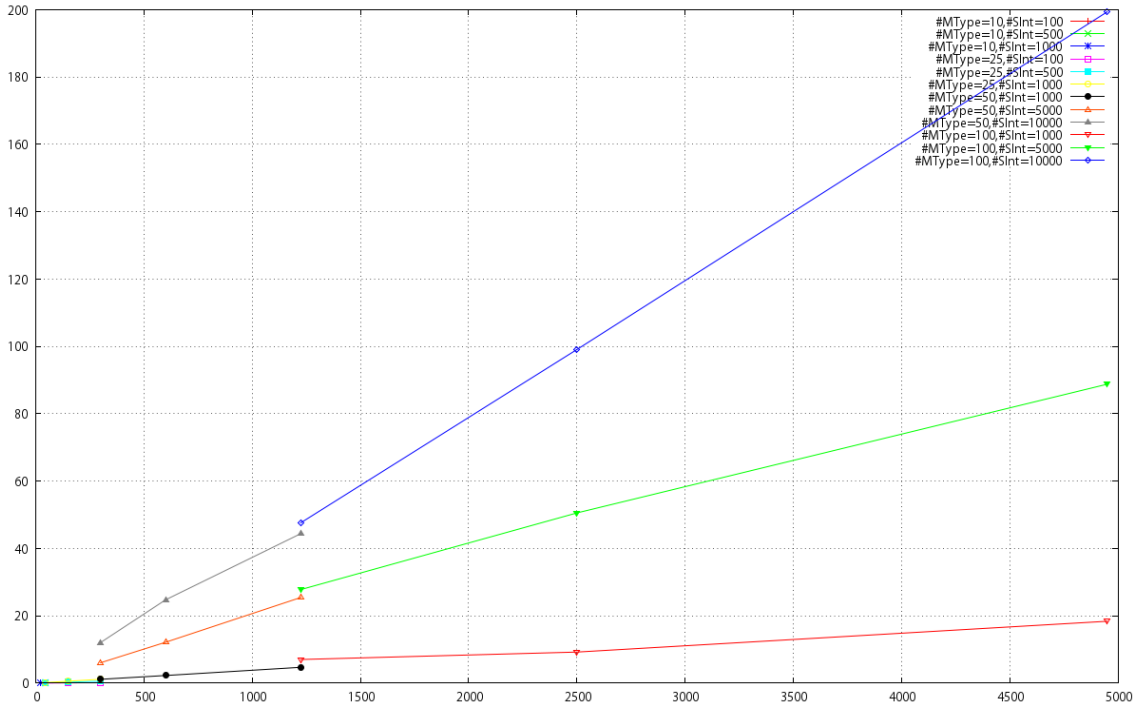


Figure 4.10: Complexity evolution as a function of the number of messages types, intervals and equations of affine transformations.

4.4 Related work

Several authors addressed the task of extracting temporal rules. Starting from the Apriori-like technique encountered in [3] to extract sequential patterns we find another extension in [96] which deals with the discovery of frequent episodes and episode rules. In addition, several enriched approaches on the extraction of temporal association rules and inter-transactional association rules are proposed [139, 150].

On the other side we have the task of mining temporal rules on interval-based data with which many peers try to deal with. [82] proposes another Apriori method that eventually recognizes such patterns. This is done by employing a definition of temporal patterns relying upon temporal relationships between interval-based events. In [145], a mining technique is presented which is capable of discovering containment relationships in series of interval events. The objective is achieved through deriving events from numerical time series by means of a quantization step. In [88], a general methodology encompasses the entire process of knowledge discovery in time series databases, addressing both the preprocessing and the rule mining step.

In [35] the theory of fluent learning is employed in order to extract common

patterns in time series. Fluent learning is a statistical technique whose results with multivariate time series with binary variables are considered relevant. Meanwhile, [76] and [75] deal with mining informative temporal rules based on a set of sequences of labeled intervals. This allows for more adaptability regarding the definition of temporal patterns which is provided in [82]. In [149], a new method is presented for mining temporal patterns of high frequencies. Afterwards, a deduction of temporal rules is made from such patterns. This work is based on the research conducted in [75] and on an algorithm proposed in [93]. We note that [93] deals with the discovery of temporal patterns from interval-based data. A new formalization regarding the issue of discovering frequent arrangements of temporal intervals can be found in [113]. The method acts on a database of sequences of events, where each event occurs during a time interval. Thus, it removes the assumption of handling only instantaneous events.

Compared to the works previously mentioned, the author in [75] suggests a formulation of the problem of extracting rules from temporal patterns. This issue is very closely related to the one described in [129]. In particular, the author proposes qualitative features to divide the time series into their corresponding segments. A method is presented for mining temporal patterns from which informative rules are derived. In [75] the issue of learning qualitative labels starting from temporal data in the form of occurrence-times events is described. From the different techniques listed we may cite smoothing, clustering and piecewise linear approximation. In comparison to [75], the authors in [129] introduced a more general environment. This paper [129] deals with several areas in the field of temporal data mining (TDM) [62, 94], that allows one to extract qualitative labels from temporal data. Such an extraction proceeds through a formalization of the framework upon which rely knowledge-based temporal abstractions. By doing so the authors are capable of incorporating simultaneously the task of episode mining and the general definition of a pattern into the same process.

Moreover, the process of deriving complex patterns is naturally contained into the definition of complex temporal abstractions. The pattern inference process does not have to be computed online with the rule mining step. The primary objective of the work in [129] is to find relevant temporal rules between complex patterns based on a set of time series. Following the ideas of [15], in [129] the process begins with raw time series introducing a step for the extraction of an interval-based representation. The representation is based on the formalism of Temporal Abstractions (TA). These are an abstraction of patterns displaying a particular behavior in temporal data. In addition, the set of abstractions of interest is constituted by following such a representation. The framework presented in [15] was extended and enriched in [129]. This evolution becomes visible through complex TA and a rule detection which permits rich patterns to be located at the same time in the left and right side of the rule. Clearly, this is an effort in dealing with problems encountered in previous proposals. Indeed, the modeling that is taken into account is simplistic even if these proposals suggest a

qualitative representation of the time series [76] or achieve it through TA [15]. The same may be stated on the patterns from which the temporal rules themselves are discovered.

Table 4.2: Axis of comparison between temporal-oriented approaches

Temporal patterns identification	**Temporal rules extraction**	**Segmentation of time series**
[3] Extraction of sequential patterns using Apriori-styled methods.	[3, 96] Based on frequent episodes and rules of episodes.	[75, 129] Segmentation is achieved applying techniques for learning qualitative labels.
[35] Utilize fluent learning (statistical approach) for common patterns.	[139, 150] Mining of temporal association rules and inter-transactional rules.	[93] Interval-represented data are used for characterizing temporal patterns.

Table 4.2 provides a compact view of these approaches. Each reference falls into a given dimension/column if and only if it allows for the dimension objective to be reached. One can clearly see that these dimensions are those encountered in the references of this section.

For the papers related to the present section, the objectives that are pursued are the extraction of temporal rules using interval-based data for time-based pattern recognition and segmentation of time series employing qualitative features. Among these features we can mention for example the occurrence time of events. An important task which remains to be tackled is the improvement of temporal patterns inside other types of pattern discovery. This is because temporal patterns are often addressed in a separate manner. In addition, they are almost universal and their valuable information can be exploited for mining other sequential data. One last objective would be to further improve methods that extract temporal patterns, since these patterns are not explicitly revealed by time-stamped records.

4.5 Summary

In this chapter we have shown that correctly mining the temporal order relations between message types is a very useful challenge, since solving it allows the reconstruction of the oriented graph modeling the raw version of the business protocol. Obtaining the oriented graph is the last step towards the extraction of the finite-state machine, that yields the untimed service protocol. We have shown that using realistic log data timestamped in the universal clock format is not an obstacle for correctly mining the associated timeseries. The approach that we have presented in this chapter solves the issue of temporal order in service logs by employing affine transformation testing between flow density functions. This method achieves its objective because of the underlying theoretical properties of the flow density functions. We have shown that employing the *delta* algorithm of the previous chapter, enhanced the performance of the *gamma* extraction algorithm. This leap is due to the conclusion that second-order affine transformations can be tested as first-order affine transfomations, thus simplifying the extraction of the temporal graph. The latter allows to obtain the oriented graph of the service model. Finally, we underline the experimental conclusion that the most important factor impacting the *gamma* algorithm results is the granularity level of flow density data.

Chapter 5

On timed transitions detection and extraction.

So far, we investigated the discovery of the protocol whose traces are visible in logs. In this chapter we intend to extract the time-triggered transitions that do not emit or receive messages in the process, thus not directly visible in activity logs.

5.1 Introduction

One of the most difficult problems to tackle is the extraction of temporal constraints called *timed transitions*. This is because they are not explicitly recorded in logs. In this chapter, we present our approach for discovering such transitions. We formally define a class of patterns called *proper timeouts* and show that they reveal the presence of timed transitions in the business protocol. We also present a polynomial algorithm for extracting such patterns, as well as some experiments.

5.2 Context, problem and approach

5.2.1 Timed business protocols

Some state changes are not related to the emission of explicit messages but to temporal constraints (validity period, expiration date, *etc*). The basic model was thus enhanced with timed transitions, and renamed *timed business protocol* [16, 17]. A *timed transition* (also called implicit transition) occurs automatically, after a time

interval is elapsed since the transition has been enabled (i.e., the source state of the transition has become the current state), or after some date is reached; it is labelled by the corresponding time constraint. Note that, since the model is deterministic, a state cannot have several outgoing timed transitions.

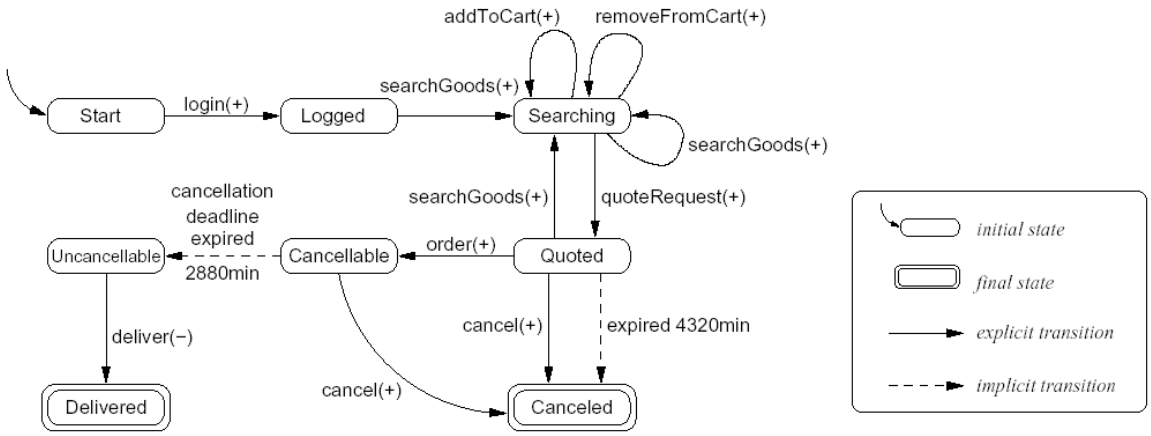


Figure 5.1: Example of a timed business protocol [16, 17].

Example 3 Fig. 5.1 illustrates more in detail the timed business protocol, already introduced in Chapter 1, describing the external behaviour of an order management service. Explicit transitions are represented by solid lines, and timed transitions by dotted lines. This protocol specifies that a customer must initially connect himself (*login* operation), before looking for products (*searchGoods*). Then, the customer can add or remove products from its cart (*addToCart*, *removeFromCart*), look for other goods (*searchGoods*), or ask for a quote (*quoteRequest*), which will be valid only during 3 days (i.e. 4320 min); during this period the goods can be ordered (*order*). If the user does not do it, the conversation finishes after 3 days (through the timed transition going out of *Quoted* state), and the order is canceled.

Using a business protocol (instead of a simple interface or code analysis) makes many problems easier to solve, because it is a *visual* model of a service behaviour. For example, it facilitates (i) the development of clients that can interact correctly with a service, (ii) the evolution of a service, when adding new functionalities, constraints or rules. As it is an *automaton*, it can be easily translated into a programming language. Thus, it enables to perform many tasks in an automatic way, such as (i) checking whether messages are sent or received according to specifications (i.e. service execution correctness), (ii) testing whether a service is replaceable or compatible with another one, or whether it conforms to a given standard. More generally, it provides a

high-level and precise language which allows expressing and reasoning about concepts at their natural level of abstraction. As such, using business protocols can improve service management, analysis and optimization.

5.2.2 Conversation logs

Extracting timed transitions would be very easy if we had access to "internal" service logs (defined in the implementation code): it should be enough to make sure that some information is recorded whenever a timed transition is triggered. However it seems not realistic, because in a management platform services are generally considered as "black boxes" whose external behaviour only (published in the interface) can be observed, and not internal operations. In fact the logs we analyze are recordings of conversations taking place between services (i.e. messages intercepted by servers or by specific interception tools[1]); that is why they are called *conversation logs*. In the sequel we will consider these logs from the point of view of a given service, i.e. we will keep only the messages that it sends or receives. In the logs of this service, there will be explicit traces of all triggered transitions except timed transitions, which are not related to explicit messages.

Various ways for collecting interaction logs of a service have been described, for example in [57]. According to service implementation and to the tools used for execution management, different information can be logged. In realistic scenarios, at least the content, the sender, the receiver and the timestamp of each message are collected. However this information can be insufficient to separate conversations, which can overlap[2]. It is highlighted in [108] that automatically providing a unique identifier for each conversation (if it is not logged by the management tool) is a research topic by itself. As such, it is assumed in [108] that this information is available. We will make the same assumption. More precisely, we assume that, for each message, the name, the timestamp and the conversation identifier are available.

We now give a more formal definition of logs. Let Msg be a set of *message names*. A *message occurrence* is a couple $M = (m, t)$, where $m \in Msg$ is the message name (denoted by $M.name$ in the sequel), and $t \in \mathbb{R}_+$ is the message timestamp (denoted by $M.time$ in the sequel). Formally, a *conversation* is a sequence of message occurrences $C = \langle M_1, M_2, \ldots, M_n \rangle$, where $n \in \mathbb{N}^*$, and $M_1.time < M_2.time < \cdots < M_n.time$. A *conversation log file* L is a multi-set of conversations, called simply *logs*

[1]We do not give details about how logs are gathered. This issue is beyond the scope of our work.

[2]Overlapping conversations result from the interactions of several instances of the service with different clients, in parallel. Isolating these conversations amounts to separating parallel executions of the service.

L in the sequel.

Example 4 For a service defined with respect to the business protocol illustrated by Fig. 5.1, we can, for instance, obtain the following logs:
⟨ (login, 9:18), (searchGoods, 9:20), (addToCart, 9:21), (quoteRequest, 9:22), (cancel, 9:51) ⟩ ⟨ (login, 11:03), (searchGoods, 11:04), (addToCart, 11:08), (quoteRequest, 11:12) ⟩

5.2.3 Problem statement and assumptions

Recall that we focus on timed transitions, which are local properties of the business protocol. The problem we address can be defined as follows: Let L be a conversation log file of a service S, whose underlying timed business protocol P is not known; extract from L the timed transitions of P.

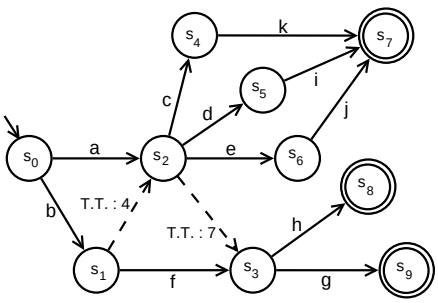
In order to avoid technical problems, and to focus on the central issue, the problem is simplified, by means of three restrictive working hypothesis, regarding the underlying timed business protocol. First, we assume that *transitions are uniquely labelled*, which means that there do not exist two transitions associated with the same message (c.f. Fig. 5.2). This implies that each message in the file corresponds to only one transition in the protocol. Second, we assume that *no final state has an incoming timed transition*, because our method cannot discover a timed transition that reaches a final state. Such transitions need an additional extracting technique that we plan to consider in future work. Actually, we can expect that, in real life scenarios, a notification message is sent when an expiration occurs. Finally, we assume that *there is no cycle* in the business protocol.

To make sure that all timed transitions can be found, we also assume that *logs are complete*, which means that all valid conversations are recorded in the logs. Finally, we assume that *logs are not noisy* (i.e. they are correctly recorded, in the right sequence).

Example 5 Logs L_1 (c.f. Fig. 5.2) will be our running example. Note that timestamps are relative to the beginning of each conversation. Protocol P_1, which has been used for generating logs L_1, is supposed to be unknown. Note also that protocol P_1 is intentionnally simplified compared to general business protocols, which can contain much more information.

5.2.4 Overall presentation of the approach

We recall that a timed transition is part of a business protocol, which is an abstract representation of a service behaviour. Since we only have logs (which are in fact



$C_1 = \langle (a,0),(c,1),(k,4) \rangle$
$C_2 = \langle (a,0),(c,3),(k,5) \rangle$
$C_3 = \langle (a,0),(d,2),(i,6) \rangle$
$C_4 = \langle (a,0),(d,5),(i,8) \rangle$
$C_5 = \langle (a,0),(e,4),(j,5) \rangle$
$C_6 = \langle (a,0),(e,6),(j,8) \rangle$
$C_7 = \langle (a,0),(h,8) \rangle$
$C_8 = \langle (a,0),(h,10) \rangle$
$C_9 = \langle (a,0),(g,15) \rangle$
$C_{10} = \langle (a,0),(g,16) \rangle$
$C_{11} = \langle (b,0),(f,1),(h,3) \rangle$
$C_{12} = \langle (b,0),(f,2),(g,4) \rangle$
$C_{13} = \langle (b,0),(f,3),(h,4) \rangle$
$C_{14} = \langle (b,0),(f,1),(g,5) \rangle$
$C_{15} = \langle (b,0),(c,6),(k,8) \rangle$
$C_{16} = \langle (b,0),(c,8),(k,9) \rangle$
$C_{17} = \langle (b,0),(d,7),(i,8) \rangle$
$C_{18} = \langle (b,0),(d,10),(i,11) \rangle$
$C_{19} = \langle (b,0),(e,8),(j,10) \rangle$
$C_{20} = \langle (b,0),(e,9),(j,10) \rangle$
$C_{21} = \langle (b,0),(h,13) \rangle$
$C_{22} = \langle (b,0),(h,15) \rangle$
$C_{23} = \langle (b,0),(g,14) \rangle$
$C_{24} = \langle (b,0),(g,15) \rangle$

Figure 5.2: Protocol P_1 (left) and associated logs L_1 (right).

recordings of service executions, in the concrete domain), we cannot work at this abstract level, and directly discover timed transitions. Furthermore, their occurrences cannot explicitly be disclosed by logs, in contrary to explicit transitions, whose labels are recorded. We need to identify some traces (i.e. some consequences) of a timed transition, showing that it has been triggered. Thus, our problem is twofold: (i) specify suitable patterns that could correspond to timed transitions, and (ii) extract these patterns from the logs.

We define a class of patterns called *proper timeouts*, and give a condition for a proper timeout to be *satisfied* by the logs. However, no equivalence can be found between the presence of a timed transition, and the satisfaction of a proper timeout. We present a necessary condition, which states that: if a timed transition belongs to the business protocol, then a corresponding proper timeout is satisfied by the logs. Conversely, we say that: a satisfied proper timeout reveals only the presence of a *potential* timed transition. Indeed, another scenario leads also to the satisfaction of a proper timeout, because it creates the same kind of traces in the logs. It occurs if, in some state of the service, some messages always take longer to be sent or received than others. In both scenarios, the presence of a timed transition will be presumed, because logs by themselves are not sufficient to distinguish them.

Although we cannot establish an equivalence between these patterns, proper timeouts are the best representations of timed transitions, in the logs. Due to the assumption on logs completeness, all timed transitions can be found, because each one of them involves the satisfaction of a proper timeout. That justifies the relevance of a timed transition discovery method based on proper timeout mining.

A basic "generate and test" approach for proper timeouts extraction would be intractable, because it would suffer from combinatorial explosion. Instead, we propose a simple characterization of the set of proper timeouts satisfied by the logs. It is based on a specific (and totally ordered) partition of the occurring episodes, which

represent pairs of messages that occur consecutively in a conversation. Once this partition is constructed, the set of proper timeouts is exactly given by its pairs of consecutive elements. Finally, we propose an incremental algorithm for constructing such a partition (presented in Section 4.2). This construction can be achieved in polynomial time with respect to the input size (i.e. the number of episodes) using only two simple operations (add and merge).

5.3 Associating patterns with timed transitions

5.3.1 Episodes

Let *Msg* be the set of message names, and L the conversation log file. To reason about consecutive messages, we introduce the notion of episode occurrence.

Definition 21 An **episode** is a sequence of two message names $\alpha = \langle m, m' \rangle$, with $m, m' \in Msg$.

Definition 22 Given an episode $\alpha = \langle m, m' \rangle$, an **occurrence** of α is a sequence of message occurrences $\langle M, M' \rangle$ such that there exists a conversation $C \in L$ satisfying:

$$\begin{cases} M, M' \in C \\ M.name = m \text{ and } M'.name = m' \\ M.time < M'.time \\ \nexists\, M'' \in C \text{ such that } M.time < M''.time < M'.time \end{cases}$$

If such a sequence exists, we say that α occurs in conversation C.

Given an episode α, we denote by $Occ(\alpha)$ its set of occurrences. We say that α occurs in logs L if α occurs in at least one conversation C of L, i.e. if $Occ(\alpha) \neq \phi$. We denote by Ep the set of episodes that occur in logs L.

Proposition 4 *Consider the set $P_m = \{\alpha \in Ep \mid \exists\, m' \in Msg,\ \alpha = \langle m, m' \rangle\}$, for each $m \in Msg$. Then, $\{P_m \mid m \in Msg\}$ is a partition of Ep.*

This means that one can partition the episodes, such that each part contains all the episodes whose first element is a given message m. It will enable to decompose our discovery task. Instead of analyzing all the episodes as a whole, we will treat each

element of this partition separately.

Example 6 Consider logs L_1 (c.f. Fig. 5.2). Here, $Ep = P_a \cup P_b \cup P_c \cup P_d \cup P_e \cup P_f$, where $P_b = \{\langle b,c\rangle, \langle b,d\rangle, \langle b,e\rangle, \langle b,f\rangle, \langle b,g\rangle, \langle b,h\rangle\}$ (note that for example $\langle b,i\rangle \notin P_b$), *etc.* Moreover, $Occ(\langle b,h\rangle) = \{\, \langle (b,0),(h,13)\rangle \,,\, \langle (b,0),(h,15)\rangle \,\}$ (but $\langle (b,0),(h,3)\rangle \notin Occ(\langle b,h\rangle)$), *etc.*

We define now the concept of occurrence duration. Intuitively, the occurrence duration of an episode occurrence is the difference between the message timestamps in this occurrence. From this, we define the minimal (respect. maximal) occurrence duration of an episode as the smallest (respect. greatest) occurrence duration of all its occurrences. The occurrence duration interval of an episode is the interval which includes all its occurrence durations.

Definition 23 Consider an episode $\alpha \in Ep$. The duration of an occurrence $\langle M, M'\rangle$ of α is $M'.time - M.time$. The minimal occurrence duration of α is $d_{min}(\alpha) = min\{M'.time - M.time \mid \langle M, M'\rangle \in Occ(\alpha)\}$. Its maximal occurrence duration is $d_{max}(\alpha) = max\{M'.time - M.time \mid \langle M, M'\rangle \in Occ(\alpha)\}$. The **occurrence duration interval** of α is $[d_{min}(\alpha); d_{max}(\alpha)]$.

In the same way, we define the minimal (respect. maximal) occurrence duration of a set of episodes as the minimum (respect. maximum) of all the minimal (respect. maximal) occurrence durations of these episodes. The occurrence duration interval of a set of episodes is the interval which includes all the occurrence durations of these episodes.

Definition 24 Consider a set of episodes $A \subseteq Ep$ $(A \neq \phi)$. The minimal occurrence duration of A is $D_{min}(A) = min\{d_{min}(\alpha) \mid \alpha \in A\}$. The maximal occurrence duration of A is $D_{max}(A) = max\{d_{max}(\alpha) \mid \alpha \in A\}$. The **occurrence duration interval** of A is $[D_{min}(A); D_{max}(A)]$.

Example 7 Consider logs L_1 at Figure 5.2. The occurrence duration interval of episode $\langle c,k\rangle$ is $[2;3]$. The occurrence duration intervals of sets $\{\langle a,c\rangle, \langle a,d\rangle\}$ and $\{\langle a,h\rangle\}$ are $[1;5]$ and $[8;10]$ respectively. They are disjoint and satisfy a precedence relation on the time scale. Obviously, this is due to the timed transition between states s_2 and s_3, triggered automatically at time 7 (after arrival in state s_2) if none of the messages c, d or e is emitted before.

Since it is the consequence of the presence of a timed transition, the precedence relation presented in this example will be useful in the sequel. If the reasoning is

reversed, finding that such a relation is satisfied by the data could lead to the discovery of a timed transition. Thus, we formalize this relation.

5.3.2 Order relation on sets of episodes

Definition 25 Consider $A, B \subseteq Ep$ $(A, B \neq \phi)$. We say that A **precedes** B, which is denoted by $A \prec B$, if $D_{max}(A) < D_{min}(B)$. We say that A and B are not comparable, which is denoted by $A \parallel B$, if $A \not\prec B$ and $B \not\prec A$.

Intuitively, the expression $A \prec B$ means that the occurrence duration interval of A precedes that of B on the time scale, and that they are disjoint. We say also that B follows A.

Property 3 *Relation $\prec$ is a strict order relation on $\mathcal{P}(Ep) \setminus \{\phi\}$.*

Proof. – *Irreflexivity*:

Consider $A \subseteq Ep$ $(A \neq \phi)$. $D_{min}(A) \leq D_{max}(A) \Rightarrow D_{max}(A) \not< D_{min}(A) \Rightarrow A \not\prec A$.

Therefore, $\forall\, A \subseteq Ep$ $(A \neq \phi)$, $A \not\prec A$.

– *Asymmetry*:

Consider $A, B \subset Ep$ such that $A \prec B$. As $A \prec B$, we have: $D_{max}(A) < D_{min}(B)$.

Since $D_{min}(A) \leq D_{max}(A)$ and $D_{min}(B) \leq D_{max}(B)$, we have: $D_{min}(A) < D_{max}(B)$.

Thus, $D_{max}(B) \not< D_{min}(A)$, i.e. $B \not\prec A$. Therefore, $\forall\, A, B \subset Ep$, $A \prec B \Rightarrow B \not\prec A$.

– *Transitivity*:

Consider $A, B, C \subset Ep$ such that $A \prec B$ and $B \prec C$. We have: $A \prec B \Rightarrow D_{max}(A) < D_{min}(B)$ and $B \prec C \Rightarrow D_{max}(B) < D_{min}(C)$. Since $D_{min}(B) \leq D_{max}(B)$, we have: $D_{max}(A) < D_{min}(C)$, i.e. $A \prec C$. Therefore, $\forall\, A, B, C \subset Ep$, we have: $(A \prec B$ and $B \prec C) \Rightarrow A \prec C$.

□

Example 8 Consider logs L_1 of Figure 5.2. We have:
· $\{\langle a, c\rangle\} \prec \{\langle a, g\rangle\}$, for $D_{max}(\{\langle a, c\rangle\}) = 3 < 15 = D_{min}(\{\langle a, g\rangle\})$

· $\{\langle a,c\rangle,\langle a,d\rangle\} \prec \{\langle a,g\rangle\}$, for $D_{max}(\{\langle a,c\rangle,\langle a,d\rangle\}) = 5 < 15 = D_{min}(\{\langle a,g\rangle\})$
· $\{\langle a,c\rangle,\langle a,d\rangle\} \prec \{\langle a,g\rangle,\langle a,h\rangle\}$, for $D_{max}(\{\langle a,c\rangle,\langle a,d\rangle\})$
$= 5 < 8 = D_{min}(\{\langle a,g\rangle,\langle a,h\rangle\})$
· $\{\langle a,c\rangle,\langle a,d\rangle,\langle a,e\rangle\} \prec \{\langle a,g\rangle,\langle a,h\rangle\}$ for $D_{max}(\{\langle a,c\rangle,\langle a,d\rangle,\langle a,e\rangle\})$
$< D_{min}(\{\langle a,g\rangle,\langle a,h\rangle\})$
· *etc*

Proposition 5 *Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. If there exists a timed transition, in the business protocol, between the state from which the transitions corresponding to the elements of A are going out, and the one from which the transitions corresponding to the elements of B are going out, then $A \prec B$.*

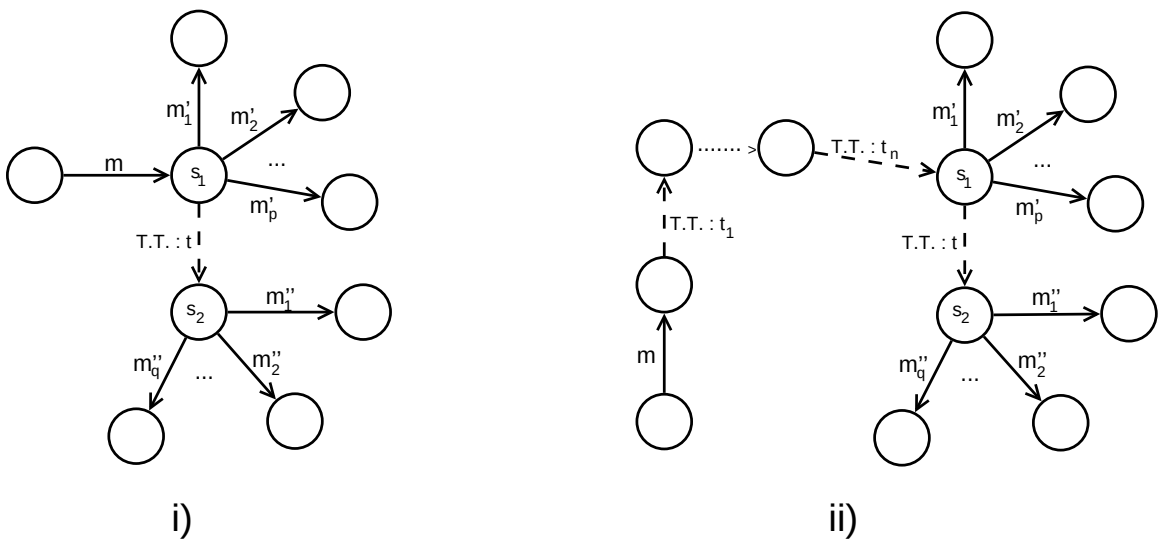


Figure 5.3: Various configurations associated with a timed transition having as time constraint t.

Proof. Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$ such that $A = \{\langle m, m'_1\rangle, \ldots, \langle m, m'_p\rangle\}$ and $B = \{\langle m, m''_1\rangle, \langle m, m''_2\rangle, \ldots, \langle m, m''_q\rangle\}$. Since transitions of the business protocol are uniquely labelled, $A \cap B = \phi$.

Assume there exists a timed transition, in the business protocol, between the state (denoted by s_1) from which the transitions labelled by m'_1, m'_2, $\ldots$, m'_p are going

out, and the one (denoted by s_2) from which the transitions labelled by m''_1, m''_2, ..., m''_q are going out. To this transition is associated some time constraint t. During a conversation, once state s_1 is reached, if none of the messages m'_1, m'_2, ..., m'_p is emitted before time t, the service reaches automatically state s_2. In other words, in a point of view which is global to all executions, we observe that: once state s_1 is reached, messages m'_1, m'_2, ..., m'_p can be emitted only before time t, although messages m''_1, m''_2, ..., m''_q can be emitted only after.

- If the transition labelled by m comes into state s_1 (c.f. Fig. 5.3 *i*)):

  We have: $\forall\ 1 \leq i \leq p$, $d_{max}(\langle m, m'_i \rangle) < t$, and $\forall\ 1 \leq j \leq q$, $d_{min}(\langle m, m''_j \rangle) > t$. As $D_{max}(A) = max_{1 \leq i \leq p}\{d_{max}(\langle m, m'_i \rangle)\}$ and $D_{min}(B) = min_{1 \leq j \leq q}\{d_{min}(\langle m, m''_j \rangle)\}$, we have: $D_{max}(A) < t < D_{min}(B)$. Therefore, $A \prec B$.

- Else (c.f. Fig. 5.3 *ii*)):

  The state in which the transition labelled by m is coming is linked to s_1 by a sequence of n ($n \geq 1$) timed transitions associated with time constraints t_1, t_2, ..., t_n. Then we have: $D_{max}(A) < t+T < D_{min}(B)$, where $T = t_1+t_2+\cdots+t_n$. Therefore, $A \prec B$.

Remark. We can prove that, if there exists a sequence of m timed transitions ($m \geq 1$) between the states from which the transitions corresponding to the elements of A and B respectively are going out, then $A \prec B$. In this case, if t_1, t_2, ..., t_m are the time constraints associated with the different timed transitions, the proof is similar, with $t = t_1 + t_2 + \cdots + t_m$.

□

This proposition is a necessary condition for the existence of a timed transition. Our problem would be solved if this condition was also sufficient (we would have an

object equivalent to a timed transition); but that is not the case.

Remark. The converse of Proposition 5 does not hold.

Counter-example. We have $\{\langle a, c\rangle\} \prec \{\langle a, e\rangle\}$ (for $D_{max}(\{\langle a, c\rangle\}) = 3 < 4 = D_{min}(\{\langle a, e\rangle\})$) in logs L_1, whereas the transitions labelled by c and e are going out of the same state (c.f. Fig. 5.2).

Proposition 5 and the assumption on logs completeness guarantee that the set of expressions $A \prec B$ verified by the logs encompasses all timed transitions. However, these expressions can also give false information, about non-existent transitions. This arises from the fact that the relation $\prec$ does not take into account all the information induced by the presence of a timed transition. That is why we define in the sequel a richer relation on sets of episodes.

5.3.3 Timeouts

Definition 26 A **timeout** is a triplet $T(m, A, B)$, where $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. We say that logs L satisfy the timeout $T(m, A, B)$, which is denoted by $L \vDash T(m, A, B)$, if:

$$\begin{cases} A \prec B \\ \forall\, \alpha \in P_m,\ \{\alpha\} \nparallel A \text{ or } \{\alpha\} \nparallel B \end{cases}$$

If $A = \{\langle m, m'_1\rangle, \langle m, m'_2\rangle, \ldots, \langle m, m'_p\rangle\}$, $B = \{\langle m, m''_1\rangle, \ldots, \langle m, m''_q\rangle\}$, and $L \vDash T(m, A, B)$, we write: $L \vDash T(m, \{m'_1, m'_2, \ldots, m'_p\}, \{m''_1, m''_2, \ldots, m''_q\})$.

Intuitively, the assertion $L \vDash T(m, A, B)$ means that, according to logs L, (i) occurrence durations of all the episodes in A are strictly less than occurrence durations of all the episodes in B, and that (ii) there is no episode having one occurrence whose duration belongs to the occurrence duration interval of A, and another occurrence whose duration belongs to the occurrence duration interval of B. With this timeout can be associated a deadline, greater than the maximum occurrence duration of all the episodes in A (i.e. $D_{max}(A)$), and less than the minimal occurrence duration of all the episodes in B (i.e. $D_{min}(B)$). The possible values for such a deadline are all real numbers in the interval $]D_{max}(A); D_{min}(B)[$, called the expiry interval of $T(m, A, B)$.

Example 9 $T(b, \{c, d, e\}, \{g, h\})$ is a timeout satisfied by logs L_1 (c.f. Fig. 5.2).

Indeed, the sets $\{\langle b,c\rangle, \langle b,d\rangle, \langle b,e\rangle\}$ and $\{\langle b,g\rangle, \langle b,h\rangle\}$ satisfy the conditions of Definition 26 for P_b, i.e.
· $\{\langle b,c\rangle, \langle b,d\rangle, \langle b,e\rangle\} \prec \{\langle b,g\rangle, \langle b,h\rangle\}$ (for $D_{max}(\{\langle b,c\rangle, \langle b,d\rangle, \langle b,e\rangle\})$
$< D_{min}(\ \langle b,g\rangle, \langle b,h\rangle\})\ )$
· $\forall\, \alpha \in P_b$, $\{\alpha\} \nparallel \{\langle b,c\rangle, \langle b,d\rangle, \langle b,e\rangle\}$ or $\{\alpha\} \nparallel \{\langle b,g\rangle, \langle b,h\rangle\}$ (for example $\{\langle b,f\rangle\} \prec \{\langle b,g\rangle, \langle b,h\rangle\}$)
We also verify that $L_1 \vDash T(a,\{c,d,e\},\{g\})$, $L_1 \vDash T(a,\{c,d\},\{g\})$, $L_1 \vDash T(b,\{f\},\{c,d,e\})$, $L_1 \vDash T(b,\{f\},\{g,h\})$, $L_1 \vDash T(b,\{f\},\{c,d,e,g,h\})$, *etc.*
On the other hand, $L_1 \nvDash T(a,\{c\},\{e\})$, for $\{\langle a,d\rangle\} \parallel \{\langle a,c\rangle\}$ and $\{\langle a,d\rangle\} \parallel \{\langle a,e\rangle\}$.

Proposition 6 *Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. If there exists a timed transition in the business protocol between the state from which the transitions corresponding to the elements of A are going out, and the one from which the transitions corresponding to the elements of B are going out, then $L \vDash T(m, A, B)$.*

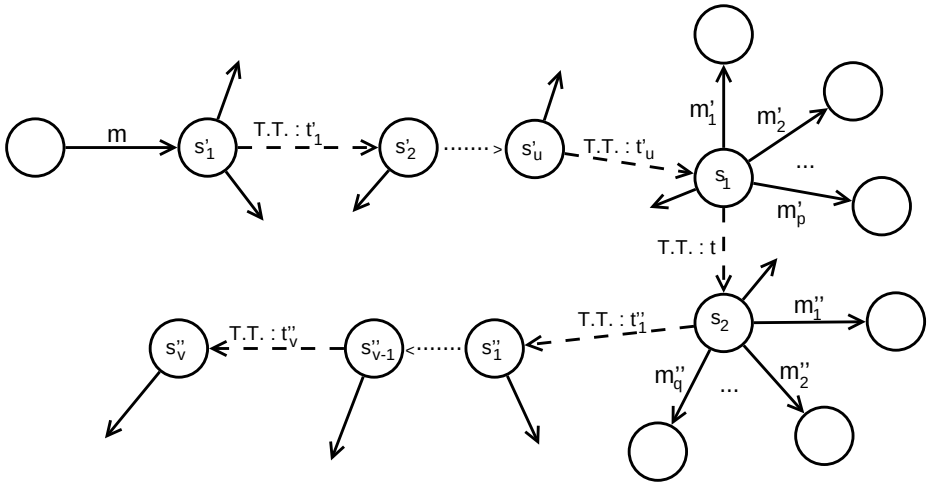


Figure 5.4: General configuration associated with A and B.

Proof. Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$ such that $A = \{\langle m, m'_1\rangle, \ldots, \langle m, m'_p\rangle\}$ and $B = \{\langle m, m''_1\rangle, \ldots, \langle m, m''_q\rangle\}$. Assume there exists a timed transition, in the business protocol, between the state from which the transitions labelled by m'_1, m'_2, ..., m'_p are going out, and the one from which the transitions labelled by m''_1,

$m''_2, \ldots, m''_q$ are going out. This configuration is illustrated by Fig. 5.4 (where u and v can be equal to zero). According to Proposition 5, we already know that $A \prec B$. Consider $\alpha = \langle m, m' \rangle \in P_m$. m' is associated with a transition going out of, either state s_1, or state s_2, or one of the states $s'_1, s'_2, \ldots, s'_u$, or one of the states $s''_1, s''_2, \ldots, s''_v$.

- If it is s_1, or one of the states $s'_1, s'_2, \ldots, s'_u$, we have: $\forall\ 1 \leq j \leq q,\ d_{max}(\alpha) < t + \sum_{k=1}^{u} t'_k < d_{min}(\langle m, m''_j \rangle)$. Thus: $D_{max}(\{\alpha\}) = d_{max}(\alpha) < min\{d_{min}(\langle m, m''_j \rangle) \mid 1 \leq j \leq q\} = D_{min}(B)$. Then, $\{\alpha\} \prec B$, and therefore $\{\alpha\} \nparallel B$.
- If it is s_2, or one of the states $s''_1, s''_2, \ldots, s''_v$, we have: $\forall\ 1 \leq i \leq p,\ d_{min}(\alpha) > t + \sum_{k=1}^{u} t'_k > d_{max}(\langle m, m'_i \rangle)$. Thus: $D_{min}(\{\alpha\}) = d_{min}(\alpha) > max\{d_{max}(\langle m, m'_i \rangle) \mid 1 \leq i \leq p\} = D_{max}(A)$. Then, $A \prec \{\alpha\}$, and therefore $\{\alpha\} \nparallel A$.
- Conclusion: $\forall\, \alpha \in P_m$, $\{\alpha\} \nparallel A$ or $\{\alpha\} \nparallel B$.

Since $A \prec B$, we have: $L \vDash T(m, A, B)$.

□

Remark. The converse of Proposition 6 does not hold.

Counter-example. The timeout $T(b, \{f\}, \{g, h\})$ is satisfied by logs L_1, although there is no timed transition between states s_1 and s_3 of protocol P_1 (c.f. Fig. 5.2). However, a chain composed of two timed transitions connects these states.

Proposition 6 and the assumption on logs completeness guarantee that each timed transition of the business protocol can be found via some timeout satisfied by the logs. However, a timeout is satisfied between two sets of episodes, if a timed transition is present between the states corresponding to those sets of episodes, as well as if it is a chain of timed transitions. Thus, we define a restricted class of timeouts, in order to avoid such ambiguity. Another problem related to timeouts is that they are much

more numerous than timed transitions are (c.f. Ex. 10).

Example 10 Consider protocol P_1 and logs L_1 (c.f. Fig. 5.2). The timed transition between states s_1 and s_2 involves the satisfaction of $T(b, \{f\}, \{c, d, e\})$, but also of $T(b, \{f\}, \{c, d, e, g, h\})$. The one between s_2 and s_3 leads to the satisfaction of $T(a, \{c, d, e\}, \{g\})$, but also of $T(a, \{c, d\}, \{g\})$.

This example illustrates that several forms of "redundancy" can appear. Nevertheless, we would like to give to the user the minimal information needed to find the timed transitions. This is why we define *proper timeouts*.

5.3.4 Proper timeouts

Definition 27 A **proper timeout** is a triplet $PT(m, A, B)$, where $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. We say that logs L satisfy the proper timeout $PT(m, A, B)$, which is denoted by $L \vDash PT(m, A, B)$, if:

$$\begin{cases} A \prec B \\ \forall\, \alpha \in P_m \setminus (A \cup B),\ \{\alpha\} \nparallel A \cup B \\ \forall\, X, Y \subset A\ (X, Y \neq \phi) \text{ such that } X \cup Y = A, \text{ we have } X \not\prec Y \\ \forall\, X, Y \subset B\ (X, Y \neq \phi) \text{ such that } X \cup Y = B, \text{ we have } X \not\prec Y \end{cases}$$

If $A = \{\langle m, m'_1\rangle, \langle m, m'_2\rangle, \ldots, \langle m, m'_p\rangle\}$, $B = \{\langle m, m''_1\rangle, \ldots, \langle m, m''_q\rangle\}$, and $L \vDash PT(m, A, B)$, for simplicity we write: $L \vDash PT(m, \{m'_1, m'_2, \ldots, m'_p\}, \{m''_1, m''_2, \ldots, m''_q\})$.

Intuitively, the assertion $L \vDash PT(m, A, B)$ means that, according to logs L, (i) occurrence durations of all the episodes in A are strictly less than occurrence durations of all the episodes in B, and (ii) there is no episode, except those of A and B, having occurrences whose durations belong to the occurrence duration interval of $A \cup B$ (interval including the occurrence duration intervals of A and B), and that (iii) there is no partition of A or B composed of two subsets ordered by the relation $\prec$. Obviously, a proper timeout is a timeout. This result is formalized by the following property, whose converse does not hold (c.f. Ex. 11).

Property 4 *Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. If $L \vDash PT(m, A, B)$, then $L \vDash T(m, A, B)$.*

Proof. Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. Assume that: $L \vDash PT(m, A, B)$. Then we have: $A \prec B$. Consider $\alpha \in P_m$.

- If $\alpha \in P_m \setminus (A \cup B)$, by assumption, we have: $\{\alpha\} \nparallel A \cup B$, i.e. $\{\alpha\} \prec A \cup B$ or $A \cup B \prec \{\alpha\}$. Thus: $d_{max}(\alpha) < D_{min}(A \cup B)$ or $d_{min}(\alpha) > D_{max}(A \cup B)$. Then, $d_{max}(\alpha) < D_{min}(A)$ or $d_{min}(\alpha) > D_{max}(A)$, for $A \subset A \cup B \Rightarrow [D_{min}(A \cup B) \leq D_{min}(A)$ and $D_{max}(A \cup B) \geq D_{max}(A)]$. Therefore, $\{\alpha\} \prec A$ or $A \prec \{\alpha\}$, i.e. $\{\alpha\} \nparallel A$.
- If $\alpha \in A$, we have: $d_{max}(\alpha) \leq D_{max}(A)$. As $D_{max}(A) < D_{min}(B)$ (for $A \prec B$), we have: $d_{max}(\alpha) < D_{min}(B)$. Then, $\{\alpha\} \prec B$, and therefore $\{\alpha\} \nparallel B$.
- If $\alpha \in B$, we have: $d_{min}(\alpha) \geq D_{min}(B)$. As $D_{min}(B) > D_{max}(A)$ (for $A \prec B$), we have: $d_{min}(\alpha) > D_{max}(A)$. Then, $A \prec \{\alpha\}$, and therefore $\{\alpha\} \nparallel A$.
- Conclusion: $\forall\, \alpha \in P_m$, $\{\alpha\} \nparallel A$ or $\{\alpha\} \nparallel B$

Since $A \prec B$, we have: $L \vDash T(m, A, B)$.

□

Example 11 $PT(b, \{c, d, e\}, \{g, h\})$ is a proper timeout satisfied by logs L_1 (c.f. Fig. 5.2), because $\{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$ and $\{\langle b, g\rangle, \langle b, h\rangle\}$ satisfy the conditions of Definition 27, for P_b, i.e.
· $\{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\} \prec \{\langle b, g\rangle, \langle b, h\rangle\}$
· $\{\langle b, f\rangle\} \nparallel \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle, \langle b, g\rangle, \langle b, h\rangle\}$
· $\forall\, X, Y \subset \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$ $(X, Y \neq \phi)$ such that $X \cup Y = \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$, we have $X \not\prec Y$
· $\{\langle b, g\rangle\} \parallel \{\langle b, h\rangle\}$
We also verify that, for example, $L_1 \vDash PT(a, \{c, d, e\}, \{h\})$.
On the other hand, we have:
· $L_1 \nvDash PT(b, \{f\}, \{g, h\})$, for $\{\langle b, c\rangle\} \parallel \{\langle b, f\rangle, \langle b, g\rangle, \langle b, h\rangle\}$
· $L_1 \nvDash PT(b, \{f\}, \{c, d, e, g, h\})$, for $\{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\} \prec \{\langle b, g\rangle, \langle b, h\rangle\}$
· $L_1 \nvDash PT(a, \{c, d\}, \{g\})$, for $\{\langle a, e\rangle\} \parallel \{\langle a, c\rangle, \langle a, d\rangle, \langle a, g\rangle\}$

Proposition 7 *Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. If there exists a timed transition, in the business protocol, between two states s_1 and s_2 such that the*

sets of transitions going out of s_1 and s_2 respectively are in bijection with A and B, then there exist $A' \subseteq A$ and $B' \subseteq B$ $(A', B' \neq \phi)$ such that $L \vDash PT(m, A', B')$.

Proof. Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. Assume there exists a timed transition, in the business protocol, between two states s_1 and s_2 such that the sets of transitions going out of s_1 and s_2 are in bijection with A and B respectively.

Consider $\{A_1, A_2, \ldots, A_x\}$ the partition (which can be reduced to one element) of A such that:

– $A_1 \prec A_2 \prec \cdots \prec A_x$ (knowing that $\prec$ is an order relation), and

– $\forall\ 1 \leq k \leq x$, $\forall\ X, Y \subset A_k$ $(X, Y \neq \phi)$ such that $X \cup Y = A_k$, we have $X \not\prec Y$.

Consider $\{B_1, B_2, \ldots, B_y\}$ the partition (which can be reduced to one element) of B such that:

– $B_1 \prec B_2 \prec \cdots \prec B_y$, and

– $\forall\ 1 \leq k \leq y$, $\forall\ X, Y \subset B_k$ $(X, Y \neq \phi)$ such that $X \cup Y = B_k$, we have $X \not\prec Y$.

Denote by m'_1, m'_2, …, m'_p the elements of A_x, and by m''_1, m''_2, …, m''_q the elements of B_1. This configuration is illustrated by Fig. 5.4 (where u and v can equal zero). According to Proposition 6, we already know that $L \vDash T(m, A_x, B_1)$, and thus $A_x \prec B_1$. Consider $\alpha = \langle m, m' \rangle \in P_m \setminus (A_x \cup B_1)$. m' is associated with a transition going out of, either state s_1, or state s_2, or one of the states s'_1, s'_2, …, s'_u, or one of the states s''_1, s''_2, …, s''_v.

- If it is one of the states s'_1, s'_2, …, s'_u, we have:

  $\forall\ 1 \leq i \leq p$, $d_{max}(\alpha) < \sum_{k=1}^{u} t'_k < d_{min}(\langle m, m'_i \rangle)$; and $\forall\ 1 \leq j \leq q$, $d_{max}(\alpha) < \sum_{k=1}^{u} t'_k < d_{min}(\langle m, m''_j \rangle)$. Thus: $d_{max}(\alpha) < min(\{d_{min}(\langle m, m'_i \rangle) \mid 1 \leq i \leq p\} \cup \{d_{min}(\langle m, m''_j \rangle) \mid 1 \leq j \leq q\})$, i.e. $D_{max}(\{\alpha\}) < D_{min}(A_x \cup B_1)$. Then, $\{\alpha\} \prec A_x \cup B_1$, and therefore $\{\alpha\} \nparallel A_x \cup B_1$.

- If it is one of the states s''_1, s''_2, ..., s''_v, we have:

  $\forall\ 1 \leq i \leq p$, $d_{min}(\alpha) > t + t''_1 + \sum_{k=1}^{u} t'_k > d_{max}(\langle m, m'_i \rangle)$; and $\forall\ 1 \leq j \leq q$, $d_{min}(\alpha) > t + t''_1 + \sum_{k=1}^{u} t'_k > d_{max}(\langle m, m''_j \rangle)$. Thus: $d_{min}(\alpha) > max(\{d_{max}(\langle m, m'_i \rangle) \mid 1 \leq i \leq p\} \cup \{d_{max}(\langle m, m''_j \rangle) \mid 1 \leq j \leq q\})$, i.e. $D_{min}(\{\alpha\}) > D_{max}(A_x \cup B_1)$. Then, $A_x \cup B_1 \prec \{\alpha\}$, and therefore $\{\alpha\} \nparallel A_x \cup B_1$.

- If it is state s_1 (which implies that $x \geq 2$):

  Since $\alpha \in P_m \setminus A_x$, we have: $\alpha \in A_l$, with $1 \leq l \leq x - 1$. Thus: $d_{max}(\alpha) \leq D_{max}(A_l) < D_{min}(A_x)$ (for $A_l \prec A_x$, by transitivity). Furthermore, $\forall\ 1 \leq j \leq q$, $d_{max}(\alpha) < t + \sum_{k=1}^{u} t'_k < d_{min}(\langle m, m''_j \rangle)$. Then, $d_{max}(\alpha) < D_{min}(B_1)$, and therefore $D_{max}(\{\alpha\}) = d_{max}(\alpha) < D_{min}(A_x \cup B_1)$. Accordingly, $\{\alpha\} \prec A_x \cup B_1$, and thus $\{\alpha\} \nparallel A_x \cup B_1$.

- If it is state s_2 (which implies that $y \geq 2$):

  Since $\alpha \in P_m \setminus B_1$, we have: $\alpha \in B_h$, with $2 \leq h \leq y$. Thus: $d_{min}(\alpha) \geq D_{min}(B_h) > D_{max}(B_1)$ (for $B_1 \prec B_h$, by transitivity). Furthermore, $\forall\ 1 \leq i \leq p$, $d_{min}(\alpha) > t + \sum_{k=1}^{u} t'_k > d_{max}(\langle m, m'_i \rangle)$. Then, $d_{min}(\alpha) > D_{max}(A_x)$, and therefore $D_{min}(\{\alpha\}) = d_{min}(\alpha) > D_{max}(A_x \cup B_1)$. Accordingly, $A_x \cup B_1 \prec \{\alpha\}$, and thus $\{\alpha\} \nparallel A_x \cup B_1$.

- Conclusion: $\forall\ \alpha \in P_m \setminus (A_x \cup B_1)$, $\{\alpha\} \nparallel A_x \cup B_1$.

By assumption, we have also:

- $\forall\ X, Y \subset A_x$ $(X, Y \neq \phi)$ such that $X \cup Y = A_x$, we have $X \not\prec Y$;
- $\forall\ X, Y \subset B_1$ $(X, Y \neq \phi)$ such that $X \cup Y = B_1$, we have $X \not\prec Y$.

Therefore, $L \vDash PT(m, A_x, B_1)$, with $A_x \subseteq A$ and $B_1 \subseteq B$ $(A_x, B_1 \neq \phi)$.

$\square$

Remark. The converse of Proposition 7 does not hold.

Counter-example. We have $L_1 \vDash PT(a, \{h\}, \{g\})$, although the transitions labelled by h and g are going out of the same state (c.f. Fig. 5.2). The satisfaction of this proper timeout is explained by the fact that, in logs L_1, after message a has been emitted, message g always takes longer to be emitted than message h.

According to Proposition 7 and the assumption on logs completeness, since each timed transition involves the satisfaction by the logs of a proper timeout, we can find all of them. However, we can discover more proper timeouts than there are timed transitions, if some messages always take longer to be sent or received than the messages associated with all other transitions of the same state. The following theorem states that this is the only possible alternative.

Theorem 9 *Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. If $L \vDash PT(m, A, B)$, then there exist in the business protocol:*

- *either two states s_1 and s_2, such that s_2 is linked to s_1 by a timed transition, that A corresponds to a subset of the transitions going out of s_1, and that B corresponds to a subset of the transitions going out of s_2,*
- *or one state s, such that $A \cup B$ corresponds to a subset of the transitions going out of s, and that the messages in B always take longer to be emitted than those in A.*

Proof. Consider $m \in Msg$, and $A, B \subset P_m$ $(A, B \neq \phi)$. Assume that $L \vDash PT(m, A, B)$, i.e.:

$$\begin{cases} A \prec B & (1) \\ \forall\, \alpha \in P_m \setminus (A \cup B),\ \{\alpha\} \nparallel A \cup B & (2) \\ \forall\, X, Y \subset A\ (X, Y \neq \phi) \text{ such that } X \cup Y = A, \text{ we have } X \not\prec Y & (3) \\ \forall\, X, Y \subset B\ (X, Y \neq \phi) \text{ such that } X \cup Y = B, \text{ we have } X \not\prec Y & (4) \end{cases}$$

Denote by s_m the state of the business protocol in which the transition labelled by m is coming. Since $A, B \subset P_m$, the elements in $A \cup B$ correspond to transitions which are going out of, either s_m, or a state linked to s_m by a timed transition, or by a sequence of timed transitions.

- If all these transitions are going out of the same state (s_m or another):

  Since $A \prec B$, we know that the messages associated with the elements of B always take longer to be emitted than those associated with elements of A.

- If these transitions are going out of two different states s_1 and s_2:

  s_1 and s_2 are linked by a sequence of p timed transitions, where $p \geq 1$ (if $p = 1$, it is a single timed transition). Without loss of generality, we can assume that s_1 is at the "beginning" of the sequence, and s_2 at the "end". Let E_1 and E_2 denote the sets of episodes from P_m corresponding to the transitions going out of s_1 and s_2 respectively.

  We have: $A \cup B \subseteq E_1 \cup E_2$, $(A \cup B) \cap E_1 \neq \phi$, and $(A \cup B) \cap E_2 \neq \phi$.

  Assume that $E_1 \cap A = \phi$. We have $A \subseteq E_2$, and therefore $E_1 \cap B \neq \phi$. Assume that $E_2 \cap B \neq \phi$. We have $(E_1 \cap B) \cup (E_2 \cap B) = B$. Since s_2 is linked to s_1 by a sequence of timed transitions, we have: $E_1 \cap B \prec E_2 \cap B$, which contradicts (4). Then, $E_2 \cap B = \phi$, and thus $B \subseteq E_1$. As $A \subseteq E_2$, and s_2 is at the end of the timed transitions sequence, we have: $B \prec A$, which contradicts (1). Therefore, $E_1 \cap A \neq \phi$.

  Assume that $E_2 \cap A \neq \phi$. We have $(E_1 \cap A) \cup (E_2 \cap A) = A$. Since s_2 is linked to s_1 by a sequence of timed transitions, we have: $E_1 \cap A \prec E_2 \cap A$, which contradicts (3). Therefore, $E_2 \cap A = \phi$. Then, $A \subseteq E_1$, and thus $E_2 \cap B \neq \phi$. As we cannot have $E_1 \cap B \neq \phi$, we have: $B \subseteq E_2$. Thus, A and B correspond

to two sets of transitions going out of s_1 and s_2 respectively.

Assume that $p \geq 2$ (i.e. the sequence is composed of several transitions). There exists a state s_3 "between" s_1 and s_2. Let E_3 denote the set of episodes from P_m associated with the transitions going out of s_3. Consider $\alpha \in E_3$. We have: $d_{min}(\alpha) < D_{max}(B) = D_{max}(A \cup B)$ (for $A \prec B$), and thus $A \cup B \not\prec \{\alpha\}$. Furthermore, $d_{max}(\alpha) > D_{min}(A) = D_{min}(A \cup B)$ (for $A \prec B$), and thus $\{\alpha\} \not\prec A \cup B$. Therefore, $\{\alpha\} \parallel A \cup B$ with $\alpha \in P_m \setminus (A \cup B)$, which contradicts (2). Consequently, $p = 1$, i.e. s_2 is linked to s_1 by a single timed transition.

- If these transitions are going out of n states $s_1, s_2, \ldots, s_n$ (with $n \geq 3$):

  Let $E_1, E_2, \ldots, E_n$ denote the sets of episodes from P_m which correspond to the transitions going out of $s_1, s_2, \ldots, s_n$ respectively. We have: $A \cup B \subseteq \bigcup_{i=1}^{n} E_i$, and $\forall\ 1 \leq i \leq n$, $(A \cup B) \cap E_i \neq \phi$. Assume that: $\exists\ i \in [\![1; n]\!]$ such that $A \subseteq E_i$. If $I = [\![1; n]\!] \setminus \{i\}$, we have: $card(I) \geq 2$, and $\forall\ k \in I$, $B \cap E_k \neq \phi$. Therefore, $\exists\ j \in I$ such that $B \cap E_j \prec \bigcup_{k \in I \setminus \{j\}} (B \cap E_k)$, which contradicts (4). Thus, $\nexists\ i \in [\![1; n]\!]$ such that $A \subseteq E_i$. If $J = \{k \in [\![1; n]\!] \mid A \cap E_k \neq \phi\}$, we have: $card(J) \geq 2$. Therefore, $\exists\ j \in J$ such that $A \cap E_j \prec \bigcup_{k \in J \setminus \{j\}} (A \cap E_k)$, which contradicts (3). Thus, we cannot have $n \geq 3$.

□

Although we cannot ensure a total mapping between these objects, proper timeouts are the best possible representations of timed transitions, in the logs, for practical purposes. Theorem 9 guarantees that, if we discover a proper timeout in the logs, then there is a timed transition in the business protocol, or some messages take longer to be emitted than others, knowing that the logs by themselves are not sufficient to detect such an alternative. Thus, we will say that: a satisfied proper timeout reveals the presence of a *potential* timed transition. That justifies the relevance of the development of a timed transition discovery method based on the research of the proper timeouts satisfied by the logs.

5.4 Extracting the proper timeouts

The "naive" discovery method consists in generating all possible proper timeouts, and testing for each one of them if the conditions of Definition 27 are satisfied. However, this method is double exponential because (i) the number of candidates is exponential, and (ii) for each candidate, if $A \prec B$, we must check that all subsets of A and B are not comparable with respect to $\prec$. Instead, we propose a new characterization of the set of proper timeouts satisfied by the logs, which will enable us to construct this set in a simple way.

5.4.1 Characterization of the satisfied proper timeouts

This characterization is formalized by the following theorem. It states that: for a set P_m of episodes whose first message is m, if we build a partition Π of P_m such that (i) Π is totally ordered by the relation $\prec$, and (ii) for each element of Π, there does not exist a sub-partition of this element composed of two subsets ordered by the relation $\prec$; then (a) there is a proper timeout "between" each pair of consecutive elements in Π, and (b) these are the only proper timeouts satisfied by the logs and related to message m .

Theorem 10 *Consider $m \in Msg$, $i_m \in \mathbb{N}^*$, and $\{P_m^{(1)}, P_m^{(2)}, \ldots, P_m^{(i_m)}\}$ a partition of P_m. The following assertions are equivalent:*

$$i) \begin{cases} P_m^{(1)} \prec P_m^{(2)} \prec \cdots \prec P_m^{(i_m)}, \text{ and} \\ \forall\, 1 \leq i \leq i_m,\ \forall\, X, Y \subset P_m^{(i)}\ (X, Y \neq \phi) \text{ such that } X \cup Y = P_m^{(i)},\ \text{we have } X \not\prec Y \end{cases}$$

$$ii) \begin{cases} \forall\, 1 \leq i < i_m,\ L \vDash PT(m, P_m^{(i)}, P_m^{(i+1)}), \text{ and} \\ \forall\, A, B \subset P_m\ (A, B \neq \phi) \text{ such that } L \vDash PT(m, A, B), \text{ we have:} \\ \qquad \exists\, i \in [\![1; i_m - 1]\!] \text{ such that } A = P_m^{(i)} \text{ and } B = P_m^{(i+1)} \end{cases}$$

Proof. Consider $m \in Msg$, $i_m \in \mathbb{N}^*$, and $\{P_m^{(1)}, P_m^{(2)}, \ldots, P_m^{(i_m)}\}$ a partition of P_m.

$i) \Rightarrow ii)$: Assume that

$$\begin{cases} P_m^{(1)} \prec P_m^{(2)} \prec \cdots \prec P_m^{(i_m)}, \text{ and} \\ \forall\, 1 \leq i \leq i_m, \forall\, X, Y \subset P_m^{(i)}\ (X, Y \neq \phi) \text{ such that } X \cup Y = P_m^{(i)},\ \text{we have } X \not\prec Y \end{cases}$$

Consider $i \in [\![1; i_m - 1]\!]$.

- $P_m^{(i)} \prec P_m^{(i+1)}$.
- $\forall\, X, Y \subset P_m^{(i)}$ $(X, Y \neq \phi)$ such that $X \cup Y = P_m^{(i)}$, we have $X \not\prec Y$.
- $\forall\, X, Y \subset P_m^{(i+1)}$ $(X, Y \neq \phi)$ such that $X \cup Y = P_m^{(i+1)}$, we have $X \not\prec Y$.
- Consider $\alpha \in P_m \setminus (P_m^{(i)} \cup P_m^{(i+1)})$. Since $\{P_m^{(j)} \,|\, 1 \leq j \leq i_m\}$ is a partition of P_m, $\exists\, j \in [\![1; i_m]\!] \setminus \{i, i+1\}$ such that $\alpha \in P_m^{(j)}$.
  - If $j < i$, then $P_m^{(j)} \prec P_m^{(i)}$ and $P_m^{(j)} \prec P_m^{(i+1)}$. As $\{\alpha\} \subseteq P_m^{(j)}$, we have: $\{\alpha\} \prec P_m^{(i)}$ and $\{\alpha\} \prec P_m^{(i+1)}$. Thus, $\{\alpha\} \prec P_m^{(i)} \cup P_m^{(i+1)}$, and thereby $\{\alpha\} \nparallel P_m^{(i)} \cup P_m^{(i+1)}$.
  - If $j > i+1$, then $P_m^{(i)} \prec P_m^{(j)}$ and $P_m^{(i+1)} \prec P_m^{(j)}$. As $\{\alpha\} \subseteq P_m^{(j)}$, we have: $P_m^{(i)} \prec \{\alpha\}$ and $P_m^{(i+1)} \prec \{\alpha\}$. Thus, $P_m^{(i)} \cup P_m^{(i+1)} \prec \{\alpha\}$, and thereby $\{\alpha\} \nparallel P_m^{(i)} \cup P_m^{(i+1)}$.
  - Conclusion: $\forall\, \alpha \in P_m \setminus (P_m^{(i)} \cup P_m^{(i+1)})$, $\{\alpha\} \nparallel P_m^{(i)} \cup P_m^{(i+1)}$.
- Therefore, $L \vDash PT(m, P_m^{(i)}, P_m^{(i+1)})$.

Consequently, $\forall\, 1 \leq i < i_m,\ L \vDash PT(m, P_m^{(i)}, P_m^{(i+1)})$.

Consider $A, B \subset P_m (A, B \neq \phi)$ such that $L \vDash PT(m, A, B)$. We have:

$$\begin{cases} A \prec B & (1) \\ \forall\, \alpha \in P_m \setminus (A \cup B),\ \{\alpha\} \nparallel A \cup B & (2) \\ \forall\, X, Y \subset A\ (X, Y \neq \phi) \text{ such that } X \cup Y = A, \text{ we have } X \not\prec Y & (3) \\ \forall\, X, Y \subset B\ (X, Y \neq \phi) \text{ such that } X \cup Y = B, \text{ we have } X \not\prec Y & (4) \end{cases}$$

- Since $A \subset P_m$ and $\{P_m^{(j)} \mid 1 \leq j \leq i_m\}$ is a partition of P_m, we know that: $\exists\, i \in [\![1; i_m]\!]$ such that $A \cap P_m^{(i)} \neq \phi$. We can define $I_A = \{i \in [\![1; i_m]\!] \,|\, A \cap P_m^{(i)} \neq$

$\phi\}$, $i_A = max(I_A)$ and $I'_A = I_A \setminus \{i_A\}$. Assume that $I'_A \neq \phi$. We have: $A = \bigcup_{i=1}^{i_m}(A \cap P_m^{(i)}) = \bigcup_{i \in I_A}(A \cap P_m^{(i)}) = (A \cap P_m^{(i_A)}) \cup (\bigcup_{i \in I'_A}(A \cap P_m^{(i)}))$, with $A \cap P_m^{(i_A)} \subset A$ and $\bigcup_{i \in I'_A}(A \cap P_m^{(i)}) \subset A$. However, $\forall\, i \in I'_A$, $i < i_A$, and thus: $\forall\, i \in I'_A$, $P_m^{(i)} \prec P_m^{(i_A)}$. Since $\forall\, i \in I_A$, $A \cap P_m^{(i)} \subseteq P_m^{(i)}$, we have: $\forall\, i \in I'_A$, $A \cap P_m^{(i)} \prec A \cap P_m^{(i_A)}$. Therefore $\bigcup_{i \in I'_A}(A \cap P_m^{(i)}) \prec A \cap P_m^{(i_A)}$, which contradicts (3). Thereby, $I'_A = \phi$. Consequently, $\exists!\, i \in [\![1; i_m]\!]$ such that $A \cap P_m^{(i)} \neq \phi$, i.e. $\exists\, i \in [\![1; i_m]\!]$ such that $A \subseteq P_m^{(i)}$.

- In the same way (and according to (4)), we prove that: $\exists\, j \in [\![1; i_m]\!]$ such that $B \subseteq P_m^{(j)}$.
- Assume that: $A \subsetneq P_m^{(i)}$. Consider $\alpha \in P_m^{(i)} \setminus A \subset P_m \setminus A$.
  - If $\alpha \in B$, since $A \prec B$, we have: $A \prec \{\alpha\}$.
  - If $\alpha \notin B$, $\alpha \in P_m \setminus (A \cup B)$ and, according to (2), $\{\alpha\} \nparallel A \cup B$. Thus, $\{\alpha\} \nparallel A$, i.e. $\{\alpha\} \prec A$ or $A \prec \{\alpha\}$.
  - Conclusion: $\forall\, \alpha \in P_m^{(i)} \setminus A$, we have: $\{\alpha\} \prec A$ or $A \prec \{\alpha\}$.

  Define $X = \{\alpha \in P_m^{(i)} \setminus A \mid \{\alpha\} \prec A\} \subset P_m^{(i)}$ and $Y = \{\alpha \in P_m^{(i)} \setminus A \mid A \prec \{\alpha\}\} \subset P_m^{(i)}$. We have: $P_m^{(i)} \setminus A = X \cup Y$, and thus $P_m^{(i)} = X \cup Y \cup A$. As $A \neq P_m^{(i)}$, we have: $X \neq \phi$ or $Y \neq \phi$.
  - If $X \neq \phi$ and $Y = \phi$:

    Since $\forall\, \alpha \in X$, $\{\alpha\} \prec A$, we have: $X \prec A$. Therefore, $P_m^{(i)} = X \cup A$ with $X \prec A$, which contradicts i).
  - If $X = \phi$ and $Y \neq \phi$:

    Since $\forall\, \alpha \in Y$, $A \prec \{\alpha\}$, we have: $A \prec Y$. Therefore, $P_m^{(i)} = Y \cup A$ with $A \prec Y$, which contradicts i).

- If $X \neq \phi$ and $Y \neq \phi$:

  We have: $X \prec A$ and $A \prec Y$. Therefore, $X \prec (A \cup Y)$ with $P_m^{(i)} = X \cup (Y \cup A)$, which contradicts i).

Consequently, $A = P_m^{(i)}$.

- In the same way, we prove that: $B = P_m^{(j)}$.
- Assume that: $j < i$. Then we have: $P_m^{(j)} \prec P_m^{(i)}$, i.e. $B \prec A$, which contradicts (1). Therefore $j \geq i$.
- Assume that: $j > i + 1$. Then we have: $P_m^{(i)} \prec P_m^{(i+1)} \prec P_m^{(j)}$, i.e. $A \prec P_m^{(i+1)} \prec B$. Consider $\alpha \in P_m^{(i+1)} \subset P_m \setminus (A \cup B)$. We have $A \prec \{\alpha\} \prec B$, and thus: $\{\alpha\} \not\prec A$ and $B \not\prec \{\alpha\}$. Thereby, $\{\alpha\} \not\prec A \cup B$ and $A \cup B \not\prec \{\alpha\}$. Thus, $\{\alpha\} \parallel A \cup B$ with $\alpha \in P_m \setminus (A \cup B)$, which contradicts (2). Therefore, $j \in \{i, i+1\}$.
- If $j = i$, then $A = B$, which is impossible, for $A \prec B$. Therefore, $j = i + 1$.
- Conclusion: $\exists\, i \in [\![1; i_m - 1]\!]$ such that $A = P_m^{(i)}$ and $B = P_m^{(i+1)}$

Consequently, $\forall\, A, B \subset P_m$ $(A, B \neq \phi)$ such that $L \vDash PT(m, A, B)$, we have: $\exists\, i \in [\![1; i_m - 1]\!]$ such that $A = P_m^{(i)}$ and $B = P_m^{(i+1)}$.

ii) $\Rightarrow$ *i)*:

$\forall\, i \in [\![1; i_m - 1]\!]$, since $L \vDash PT(m, P_m^{(i)}, P_m^{(i+1)})$, we have:

- $P_m^{(i)} \prec P_m^{(i+1)}$
- $\forall\, X, Y \subset P_m^{(i)}$ $(X, Y \neq \phi)$ such that $X \cup Y = P_m^{(i)}$, we have $X \not\prec Y$
- $\forall\, X, Y \subset P_m^{(i+1)}$ $(X, Y \neq \phi)$ such that $X \cup Y = P_m^{(i+1)}$, we have $X \not\prec Y$

Therefore, $P_m^{(1)} \prec P_m^{(2)} \prec \cdots \prec P_m^{(i_m)}$, and $\forall\, 1 \leq i \leq i_m$, $\forall\, X, Y \subset P_m^{(i)}$ $(X, Y \neq \phi)$ such that $X \cup Y = P_m^{(i)}$, we have $X \not\prec Y$.

□

Example 12 The set of proper timeouts satisfied by logs L_1 (c.f. Fig. 5.2) and related to message b is exactly $\{\, PT(b, \{f\}, \{c, d, e\})\,,\ PT(b, \{c, d, e\}, \{g, h\})\,\}$. Indeed, the partition $\{\, \{\langle b, f\rangle\}\,,\ \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}\,,\ \{\langle b, g\rangle, \langle b, h\rangle\}\,\}$ of P_b satisfies the conditions of Theorem 10 i)
· $\{\langle b, f\rangle\} \prec \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\} \prec \{\langle b, g\rangle, \langle b, h\rangle\}$
· $\forall\, X, Y \subset \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$ $(X, Y \neq \phi)$ such that $X \cup Y = \{\langle b, c\rangle, \langle b, d\rangle, \langle b, e\rangle\}$, we have $X \not\prec Y$
· $\{\langle b, g\rangle\} \parallel \{\langle b, h\rangle\}$

This example illustrates that, given $m \in Msg$ and the partition[3] Π of P_m satisfying Theorem 10 i), the set of proper timeouts satisfied by the logs and related to message m is exactly given by the set of pairs of consecutive elements in Π. In the sequel we present an algorithm for constructing such a partition in an incremental way.

5.4.2 Quality measure based on statistical properties

In this dissertation we investigate the possibility of assessing a quality measure in the case of an experimentally confirmed assumption of logs presenting the statistical properties of an Exponential distribution. For an elaborate introduction to the statistical tools used in this article the reader is referred to [126] for the Poisson distribution, and more details on the Exponential distribution can be found in [10, 138].

Definition 28 Consider α an episode. We define the *occurrences' random variable* as the number of occurrences of α taking place in the same time interval. Formally speaking :

$$\chi(\alpha) = Card\{M_2.time \mid \exists \langle M_1, M_2\rangle,\ \langle M_3, M_4\rangle \in Occ(\alpha),\\ M_2.time \in [M_4.time \pm \epsilon]\}$$

Intuitively, this variable represents the number of the occurrences of a given episode whose time occurrences are located in the same time interval the length value of which is two times ϵ, since ϵ represents the radius from M_4. Also note that ϵ might eventually be different for each episode, since different episodes might very easily present different densities of occurrences that might require adaptable values of ϵ.

[3]The considered set of proper timeouts being unique, this partition is unique too.

Definition 29 Let $\langle M_1, M_2 \rangle$ be an occurrence of the episode $\alpha = \langle m_1, m_2 \rangle$. We define ξ as the random variable representing the occurrence duration for all occurrences of α. In other words, given the episode occurrence $\langle M_1, M_2 \rangle$ of $\alpha = \langle m_1, m_2 \rangle$, ξ is defined as being the waiting time for the occurrence of M_2 after M_1.

Hypothesis : *Consider an episode $\alpha = \langle m_1, m_2 \rangle \in E_p$. Let ξ_{α_i}, $\alpha_i \in Occ(\alpha)$ be the independent random variables of occurrence durations of α. Then, ξ_{α_i} follows an Exponential distribution.*

This means that the values of inter-arrival waiting times between messages in a conversation path follow the Exponential distribution. As reported in [138], in real-world scenarios, the hypothesis of a constant rate (or probability per unit time) is often not satisfied but still it remains approximatively constant during fixed-length time intervals. We point out that a web service instance satisfies the *memorylessness* property requirement as our experiments have shown. Memorylessness is a property of certain probability distributions wherein any derived probability from a set of random samples is distinct and has no information (i.e. "memory") of earlier samples.

Theoretical reasoning: Consider the homogeneous Poisson distribution $N(t)$ with rate parameter λ, and let T_k be the time of the k-th dialog occurrence, for k $\in [1, 2, 3..., n]$. It is obvious that the number of occurrences before some fixed time t, which is determined by the timed transition, is less than k *if and only if* the waiting time until the k-th occurrence is more than t. In other words, the event $[N(t) < k]$ takes place *if and only if* the event $[T_k] > t$ also does. This implies that the probabilities of these events are the same:

$$P[T_k > t] = P[N(t) < k]$$

Thus we obtain:

$$P[N(t) < k] = 1 - (P[N(t) = k] + P[N(t) = k+1] + \ldots + P[N(t) = k+i] + \ldots) = 1 - \sum_{i=0}^{\infty} P[N(t) = k+i] = 1 - \sum_{i=0}^{\infty} \frac{\exp(-\lambda t)(\lambda t)^{k+i}}{(k+i)!}$$

This being an exponential form, the distribution of waiting time values should therefore be exponential.

Consequence: Let ξ_{α_i}, $\alpha_i \in Occ(\alpha)$ be the independent random variables of occurrence durations of α. Then, $\sum_i \xi_i$ follows an Erlang distribution.

The reason why this is verified can be understood with ease. Each client that is going to use the service corresponding to this protocol will follow a path from the initial state to a terminal one. This means that each service client instance will generate one and only one conversation. Meanwhile, the waiting time for the next message to occur is independent of the preceding messages. In other words, the ξ_{α_i} random variables are independent from one another. At the same time, each ξ_{α_i} variable follows the

same distribution as the theoretical reasoning and experiments show, so they are distributed in an identical manner. When these two conditions are simultaneously satisfied then the sum of the random variables follows the Erlang distribution.

Property 5 *Consider an episode α and the random variable $\chi(\alpha)$ associated to it. Then, $\chi(\alpha)$ follows a Poisson distribution.*

This property is directly verified once that the exponential distribution of the inter-arrival time is proved since the two distributions are strictly correlated between them. Nevertheless we list the requirements met by this random variable that motivate furthermore this idea. The random variable $\gamma(\sigma)$ satisfies the following criteria :

1. Dialog occurrences arrivals are independent
2. $\forall \alpha \in Occ(\alpha)$ the random variables defined by $\chi(\alpha)$ share the same probability space.
3. These random variables have the same probability distribution.
4. The time considered here is continuous and obeys to an exponential distribution.

For a more detailed view of these requirements the reader is referred to [4, 126].

Heuristics for quality measure

The reason for using a stochastic approach is the fact that statistical data contained within Web services logs are non-stationary, *i.e.* its statistical properties vary with time.

Let us assume that a timed transition exists immediately after a given message(or set of messages). This implies that there are not any occurrences of this(these) message(s) after the timed transition activation. Moreover, a message whose occurrences are located in two of more different paths of the finite-state machine, *i.e.* different conversations leading to terminal states, obeys to the same statistical behavior. This however does not necessarily mean that the rate parameter of the distribution is always the same, since this assumption is not realistic. We recall that the occurrences' time of the messages are not related to absolute time but to a relative time whose initialization point is fixed as the service instance startup. Even if this is simple to achieve, it still remans important to not overlook this fact. The main idea behind the quality measure proposed in this paper for having a supplementary proof on whether or not a discovered timed transition has strong chances to be authentic is to compute the probability of message occurrence events.

One of the main advantages offered by stochastic modelling is its resistance toward noise in logs. Indeed, even if there may be incorrect or erroneous entries, these are not going to influence the overall result since they are not taken into consideration for the statistical fitting function is defined by messages having the largest support.

Definition 30 Let $\langle M_1, M_2 \rangle$ be an occurrence of episode $\alpha = \langle m_1, m_2 \rangle$ such that $\alpha \in E_p$ and be $Occ(\alpha)$ its occurrence set. Consider the event E of having an occurrence

of α such that the corresponding occurrence duration is bigger than $d_{max}(\alpha)$. We define the **quality measure** $\Delta(\alpha)$ of α in relationship with a timed transition by the probability $P[E]$.

Proposition 8 *Consider an episode $\alpha \in E_p$, $Occ(\alpha)$ its occurrences' set, k the cardinality of $Occ(\alpha)$ and the event E described in the preceding definition. Then,*

$$P[E] = \sum_{i=0}^{k-1} \left(\frac{\exp(-\lambda d_{max}(\alpha))(\lambda d_{max}(\alpha))^i}{i!}\right) \tag{5.4.1}$$

Proof. Let us calculate the probability of the event E. E is formally defined as $[\exists \langle M_1, M_2 \rangle \in Occ(\alpha), M_2.time - M_1.time > d_{max}(\alpha)]$. We can therefore write: $P[\exists \langle M_1, M_2 \rangle \in Occ(\alpha), M_2.time - M_1.time > d_{max}(\alpha)] \Leftrightarrow P[T_k > d_{max}(\alpha)] = P[N_{d_{max}(\alpha)} < k | k = |Occ(\alpha)|] = \sum_{i=0}^{k-1} (P[N_{d_{max}(\alpha)} = i])$. Thus, we obtain the precedent equation. □

The idea laying behind the definition of a quality measure is based on the fact that the more this measure is elevated for a particular episode, the higher is the probability of encountering occurrences of the episode in the given time interval. Therefore, if no occurrence is found, we conclude that the source of this discontinuity that caused the occurrence interruption should be a timed transition. However, we should keep in mind that the λ parameter used to calculate the quality measure, is being calculated from logs and consequently it is not always as accurate as we would wish it to be.

Limitations of stochastic processes

Despite the advantages that we have seen so far, the probability-based quality measure encounters nevertheless difficulties in facing some particular situations. When these scenarios take place, the result of this measure might not always be as expected. Here we discuss these cases.

Unstable rate parameter If the rate parameter λ of the exponential distribution takes very different values for the same prefix during separated time intervals, this might potentially lower the quality measure. This difficulty can be overcome by preprocessing log data in order to separate them into clusters belonging to the same time intervals, so that the messages composing the episodes into each cluster have the same rate parameter, thus avoiding conflicts during probability calculations. The

reason that allows this solution to work is that λ remains quite stable during fixed-length time intervals.

Incomplete logs Another problem arises when there are not enough samples inside the logs for a message preceding a timed transition. In this case, the computed rate parameter λ might be very different from the real parameter. Thus, the calculation of the probability would be severely influenced and consequently it would be impossible to predict whether the result is correct or not.This approach provides the correct results as long as the statistical behavior is known in advance and the logs are rich enough to make possible the retrieval of the statistical parameters.

5.4.3 Algorithm and experiments

The construction process is expressed by Algorithm 1. The input is composed of a given message name m, the set P_m of episodes whose first message is m, and the occurrence duration intervals of all episodes in P_m (derived from the logs by the global algorithm presented in the sequel). The output is the partition Π of P_m satisfying Theorem 10 i). Π is constructed incrementally, starting with an empty partition, and inserting one by one the elements of P_m in such a way that Theorem 10 i) is satisfied at each step. This means that the sets composing Π are strictly ordered by $\prec$. In order to describe the general step of the algorithm, let us now consider that Π is already partly constructed. Let α be an episode of P_m not yet considered. A single pass is made over the partition in order to determine (i) whether the occurrence duration intervals of some elements of Π overlap the one of α, and (ii) between which sets of Π α is located according to $\prec$ (which is done by comparing their occurrence duration intervals). If there is no overlap, a new set containing α is created and inserted into the partition in compliance with $\prec$. If the overlap takes place with only one element of Π, α is simply inserted in this set. If the overlap occurs between α and several elements of Π, they are necessarily consecutive according to $\prec$; as such they are merged and α is inserted into the resulting set. Let us highlight that once a set is created in the partition, it is never split. In fact, the algorithm is based on three simple set operations: creation, insertion and merge. As for each episode $\alpha \in P_m$ only one pass is made over the partition, the complexity of this algorithm is $\mathcal{O}(|P_m|^2)$.

Proof. Correctness of Algorithm 1.

The termination of the $partitionP_m$ algorithm is guaranteed by the fact that P_m is finite. Its soundness and completeness are proven by showing that, at each step of the algorithm (i.e. for each new element inserted in the partition), the property stated in

Algorithm 7 *partition*P_m

Require: m, P_m, and $\forall\, \alpha \in P_m$, the Occurrence Duration Interval of α (denoted by $ODI(\alpha)$)

Ensure: Π the partition of P_m satisfying Theorem 10 i)

1: $\Pi = \emptyset$
2: **while** $P_m \neq \emptyset$ **do**
3: Choose episode $\alpha \in P_m$; Remove α from P_m
4: $Overlap = \emptyset$; $B_{inf} = \emptyset$; $B_{sup} = \emptyset$
5: *// pass made over the partition*
6: **for all** set $S \in \Pi$ **do**
7: **if** $ODI(S) \cap ODI(\alpha) \neq \emptyset$ **then**
8: Insert S into $Overlap$
9: **else if** $S \prec \{\alpha\}$ **then**
10: $B_{inf} = S$
11: **else**
12: $B_{sup} = S$; goto line 15
13: **end if**
14: **end for**
15: *// update of the partition*
16: **if** $Overlap = \emptyset$ **then**
17: Insert $\{\alpha\}$ into Π between B_{inf} and B_{sup}
18: **else if** $|Overlap| = 1$ **then**
19: Insert α into the set S of Π that belongs to $Overlap$
20: **else**
21: Merge all the sets of Π that belong to $Overlap$ and insert α into the resulting set
22: **end if**
23: **end while**

Theorem 10 i) is satisfied. This is obvious in the first step. In the general step, let us assume that the property was satisfied in the previous step. Then, different scenarios have to be analyzed:

- *Overlap* is empty. If $\{\alpha\}$ has to be inserted at the beginning of the partition: we find that B_{inf} is empty and that B_{sup} is the first element of Π (and $\{\alpha\}$ is inserted before B_{sup}). If it has to be at the end of the partition: we find that B_{inf} is the last element of Π and that B_{sup} is empty (and $\{\alpha\}$ is inserted after B_{inf}). Otherwise: we find that B_{inf} and B_{sup} are two consecutive elements of Π such that $B_{inf} \prec \{\alpha\} \prec B_{sup}$ (and $\{\alpha\}$ is inserted between B_{inf} and B_{sup}). Thus $\{\alpha\}$ is inserted in accordance with the order relationship between partition elements, which ensures that the first part of the property is still satisfied. Furthermore, this insertion does not change the fact that the sets of the partition cannot be decomposed. Thus, the second part of the property is also still satisfied.

- *Overlap* contains only one element. Since α is inserted into this set, the order of the partition elements is preserved. Since the occurrence duration intervals of this set and of α are not disjoined, this insertion does not enable to decompose this set. The other sets of the partition remain non decomposable.

- *Overlap* contains several elements. Let us denote them by $E_1, E_2, \ldots, E_k$, and assume without loss of generality that $E_1 \prec E_2 \prec \cdots \prec E_k$. Let us denote $E_1 \cup E_2 \cup \cdots \cup E_k \cup \{\alpha\}$ by E. We have $B_{inf} \prec \{\alpha\} \prec B_{sup}$, and $\forall\, 1 \leq i \leq k\,, ODI(E_i) \cap ODI(\alpha) \neq \emptyset$. Thus, in the partition, we have $\cdots \prec B_{inf} \prec E_1 \prec E_2 \prec \cdots \prec E_k \prec B_{sup} \prec \ldots$. Therefore, $\cdots \prec B_{inf} \prec E \prec B_{sup} \prec \ldots$. Furthermore, since $\forall\, 1 \leq i \leq k\,, ODI(E_i) \cap ODI(\alpha) \neq \emptyset$, E cannot be

decomposed. The other sets of the partition remain non decomposable.

□

Example 13 Given logs L_1 (c.f. Fig. 5.2), algorithm $partitionP_m$ constructs the partition Π of $P_a = \{\langle a, c\rangle, \langle a, h\rangle, \langle a, e\rangle, \langle a, g\rangle, \langle a, d\rangle\}$, by including the episodes one by one (the final result being independent of the insertions order):

- $\langle a, c\rangle$: in the first stage, the first set is created; $\Pi = \{\{\langle a, c\rangle\}\}$
- $\langle a, h\rangle$: since $\{\langle a, c\rangle\} \prec \{\langle a, h\rangle\}$ (i.e. there is no overlap), $\langle a, h\rangle$ is inserted in a new element of the partition after $\{\langle a, c\rangle\}$; $\Pi = \{\{\langle a, c\rangle\}, \{\langle a, h\rangle\}\}$
- $\langle a, e\rangle$: as $\{\langle a, c\rangle\} \prec \{\langle a, e\rangle\}$ and $\{\langle a, e\rangle\} \prec \{\langle a, h\rangle\}$, $\langle a, e\rangle$ is inserted in a new part between $\{\langle a, c\rangle\}$ and $\{\langle a, h\rangle\}$; $\Pi = \{\{\langle a, c\rangle\}, \{\langle a, e\rangle\}, \{\langle a, h\rangle\}\}$
- $\langle a, g\rangle$: since $\{\langle a, h\rangle\} \prec \{\langle a, g\rangle\}$, $\langle a, g\rangle$ is inserted in a new set of the partition after $\{\langle a, h\rangle\}$; $\Pi = \{\{\langle a, c\rangle\}, \{\langle a, e\rangle\}, \{\langle a, h\rangle\}, \{\langle a, g\rangle\}\}$
- $\langle a, d\rangle$: as $\{\langle a, d\rangle\} \parallel \{\langle a, c\rangle\}$ and $\{\langle a, d\rangle\} \parallel \{\langle a, e\rangle\}$ (i.e. they overlap), $\{\langle a, c\rangle\}$ and $\{\langle a, e\rangle\}$ are merged; $\Pi = \{\{\langle a, c\rangle, \langle a, e\rangle, \langle a, d\rangle\}, \{\langle a, h\rangle\}, \{\langle a, g\rangle\}\}$.

The global method for extracting all the proper timeouts satisfied by the logs is divided in two steps. The first one is a preprocessing of the data, performed in order to compute the set of message names Msg, the set of episodes Ep, and the occurrence duration intervals of all episodes in Ep. A single pass is made over the logs, during which the occurrence duration of each sequence of two consecutive messages is calculated[4]. The second step consists in constructing the partition $\{P_m \mid m \in Msg\}$ of Ep, and running algorithm $partitionP_m$ for each $m \in Msg$. The size of the logs being far greater than the number of episodes, the first step will be the most costly. Thus the complexity of the global algorithm is $\mathcal{O}(|L|)$.

Example 14 The proper timeouts satisfied by logs L_1 (c.f. Fig. 5.2), and extracted by the global method, are listed in Table 5.1.

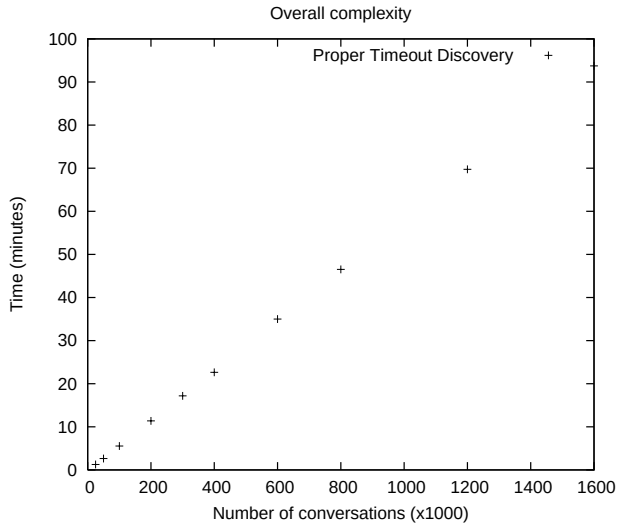
Experiments. We implemented our discovery process to test its scalability. Tests were performed on a computer with an Intel Pentium 4, 2.8 GHz processor, 2 GB of RAM, running Microsoft Windows XP Professional Edition SP2. In order to easily have a large amount of data, we employed the DOBS tool, that will be introduced in the following chapter. The implementation of the discovery algorithm is written

[4]In other words, we use an analysis window of length 2 over the data.

Table 5.1: Proper timeouts satisfied by logs L_1.

	proper timeout	expiry interval
P_a	$PT(a, \{c, d, e\}, \{h\})$	$]6; 8[$
	$PT(a, \{h\}, \{g\})$	$]10; 15[$
P_b	$PT(b, \{f\}, \{c, d, e\})$	$]3; 6[$
	$PT(b, \{c, d, e\}, \{g, h\})$	$]10; 13[$

in Java (version $1.5.0 - 06$) using the Eclipse development environment and the $java$ $-Xms512m$ $-Xmx1024m$ instruction for defining the heap size of the JVM. The maximum time for each test was set to 19 hours. Results of our experiments are presented in Fig. 5.5. They confirm the complexity results we established formally: (i) partition complexity is quadratic wrt the number of episodes, (ii) partition time is very small compared to preprocessing time, and (iii) global discovery (i.e. log preprocessing and partition) complexity is linear wrt the size of logs. Considering the size of logs, the running time is reasonable enough to state that our method is scalable. The final test will be to run our algorithm on real-life data in further experiments.



Number of episodes	Running time of partition process (in seconds)
2	0.71
4	12.5
6	27.3
8	49.4

Figure 5.5: Running times of overall discovery method (*left*) and partition process (*right*).

5.5 Discussion and state of the art

As illustrated previously, elements of the partition $\{P_m \mid m \in Msg\}$ of Ep are treated separately. Thus, with each part P_m will be associated a set of discovered proper timeouts. This process can seem redundant, in a sense that, if two transitions labelled, for example, by messages a and b enter in the same state from which a timed transition is going out, then two different proper timeouts will be satisfied by the logs (one for P_a and another for P_b), and interpreted as representing two different potential timed transitions. In fact, it is possible to realize that there is only one, by using cross-checking between all sets of proper timeouts (c.f. Ex. 15), which can be done automatically.

This cross-checking can also enable to reject some proper timeouts which cannot represent timed transitions (c.f. Ex. 15). Indeed, as already explained, a proper timeout is satisfied if, in some state of the service, some messages take longer to be sent or received than others. However, cross-checking can be inefficient, if there is no contradicting proper timeout, to reject a "fake" timed transition. In such a case, only a domain expert having some knowledge about the service can make a decision. More generally, the verification of the discovered proper timeouts by an expert could always increase the confidence about the result.

Example 15 Consider the proper timeouts satisfied by logs L_1 (c.f. Table 5.1). $PT(a, \{h\}, \{g\})$ is rejected because it cannot represent a timed transition. Indeed, $L_1 \vDash PT(b, \{c, d, e\}, \{g, h\})$ implies that $\{\langle b, h\rangle\} \parallel \{\langle b, g\rangle\}$; thus, h and g label two transitions going out of the same state (and g takes longer to be emitted than h, only after a has been emitted). Regarding the "redundancy" problem, we can see that $PT(a, \{c, d, e\}, \{h\})$ and $PT(b, \{c, d, e\}, \{g, h\})$ represent the same timed transition, because there is a correspondence between the involved sets of messages. Finally, the result consists only in two timed transitions: one corresponding to $PT(b, \{f\}, \{c, d, e\})$, and the other corresponding to $PT(a, \{c, d, e\}, \{h\})$ and $PT(b, \{c, d, e\}, \{g, h\})$.

Recall that we assumed that we have complete logs. It can be seen as a very assumption, but it is not. In fact, we ask only for *valid* conversations (i.e. the ones which finish in a final state), and not for all paths of the business protocol, to be separately represented in the logs. Furthermore, services generally do not allow a lot of operations, and complex services are mainly constructed by composing more simple ones. As such, for a given service, the set of valid conversations is quite small. Second, logs completeness is mainly a theoretical assumption, made in order to prove that our algorithm can discover all timed transitions in this case. In fact, we do not ask for logs to be complete in real life scenarios. Thus, some timed transitions can

be missed. In a re-engineering point of view, this can mean that some operations are useless, and justify an evolution of the service.

5.6 Summary

In order to mine the timed business protocol from activity logs, we presented in this chapter an approach for extracting the timed transitions of the service protocol. In order to achieve this objective, we have defined the concept of proper timeouts. Proper timeouts are defined in such a way, that they can be the optimal formal representation of timed transitions. In addition, we have provided an algorithm for identifying these proper timeouts. Nevertheless, we conclude that the need for an expert analysis is still requires, since we have shown that discovered proper timeouts might not always represent time transitions. However, given the fact that these transitions are not stored in logs, extracting proper timeouts represents an important complementary solution to the global issue of business protocol discovery.

Chapter 6

Issues on assessment, simulation and log data.

Model assessment and model mining, along with what-if analysis, are three key processes which are important in software design and engineering, service-oriented architectures and business process management. Model assessment evaluates the compliance of a model towards its specification before or after implementation. Model mining allows the extraction of the implemented model from its activity logs and without prior knowledge on the model itself. This chapter presents DOBS, a model evaluation and data generation tool for the improvement and testing of model compliance and correctness and for assisting the process of mining state diagrams or flowcharts, such as business processes, web services etc. DOBS is a continuous time/discrete event simulator that allows the design, simulation and testing of a behavioral model before its implementation, or to check and evaluate an existing real-world model such as a business process or web service for compliance requirements towards specifications. The data generation feature allows to analyze the output as well as to test mining methods on large amounts of realistic high-quality data. Experimental results show the efficiency and effectiveness of DOBS in modeling and analyzing the model behavior, testing multiple execution scenarios, as well as the huge production capacity of realistic configurable data.

6.1 Introduction

The first motivation of the this chapter is the need for data for preliminary validation of model assessment and mining algorithms. This is a problem that is common to all the research community working on data mining. For example, during QDB 2009 [122] participants pointed out that authentic data sets are extremely hard to obtain for a

researcher. Indeed, only researchers employed in R&D departments, or those participating in particular collaborations involving partners from data-generating fields (or which are in possession of these data sets) such as medical centers, large businesses, government agencies, service providers etc. can make usage of such datasets. All the other researchers are faced with a continuous and disturbing lack of data, despite exceptions like the logs provided by [115]. This concerns not only real-world data but also synthetic sources such as generators, simulators, etc. Typical scenarios where data generation and model simulation are strongly required are model assessment, what-if analysis and model mining. The other motivation is the need for a generic simulator that, by emulating business processes, workflows, and web services, can be used for what-if analysis.

Model assessment (also referred to as model compliance) addresses the issue of conformance towards a policy, standard, law or technical specification that has been clearly defined [147]. This also includes but is not limited to conforming towards business regulations and stated user service requirements [119]. As the authors in [151] point out in their survey, compliance toward regulations is finding more and more attention in the eyes of the research community. Applications of model assessment include workflow checking, protocol verification, and constraint validation [98]. Factors that motivate model compliance utility are: cost of the implementation prototype just for assessing the model, cost of re-implementing once that assessment results are negative, risk of testing on real-world already deployed systems, complexity of designed systems which prohibits exhaustive static verification and validation.

Concrete and oriented applications are mining of workflows, business processes and software specifications, business protocol discovery and web application behavior extraction. Applications include: post-mortem monitoring, checking the equivalence between the specification and implementation, obtaining the specification if it does not exist, checking for security flaws, verifying that constraints in the execution flows are satisfied, checking if the designed model is correct, complete and finite (i.e. no deadlocks, infinite loops, bottlenecks), verify performance parameters on given parts of a model. Yet data for mining is extremely hard to get mainly for confidentiality reasons, or because the data is used for commercial purposes.

Model assessment and model mining merge together in synergy, especially regarding the tasks of (i) checking the equivalence between the specification and implementation and (ii) obtaining the specification if it does not exist. In this double-sided context we present **DOBS** (**D**ynamic m**O**del **B**ehavior **S**imulator), a modeling-and-emulating generator tool which allows the expert of business processes,web services, or any other dynamic behavior-based systems, to design, test and simulate a behavioral model such as a business process or a web service / application. It also allows to generate activity data, of both low and high level of abstraction, from the simulation of the service protocol execution and testing. DOBS can thus be used as a testing

or data generation tool, or both. It can improve model compliance while helping decision support during all stages of model assessment evaluation life-cycle. DOBS utility for compliance analysis may vary depending on the context of its usage since critical systems require both pre/post assessment, whereas nominal systems should in principle require only pre-implementation assessment.

The chapter is organized as follows: Section 6.1.1 describes the principles and objectives of DOBS. Section 6.2 describes its architecture and implementation. Section 6.3 provides experimental results on the capabilities of DOBS. Section 6.4 explores existing attempts in achieving similar objectives as those of DOBS. Finally, we draw some conclusions in Section 6.5, along with future improvements.

6.1.1 Simulating Dynamic Behaviors: Principles and Objectives

DOBS is an analysis and decision support compliance verifier and simulation tool for data generation, modeling, optimization and validation of compliance in processes, services and software. The core of DOBS is a discrete event simulator that allows for the reproduction of the behavior of dynamic models. DOBS is thus mainly designed for simulating business processes, web services, workflows and software control charts. This can be used to play what-if scenarios in case of a future update, allowing experts to assess problematic points, and the impact of every modification on the evaluation criteria. By doing so, DOBS avoids the expensive and time-consuming task of implementing changes on the software/real application level.

The design principles of DOBS are based on what is generally expected from a tool of this kind. DOBS should allow through its graphical user-interface to model very complex structures in far less time compared to the amount of time required for implementing mock-ups or prototypes on the source code level. The modules composing DOBS were designed to interoperate seamlessly, so that the user can concentrate on the semantics of the result and of the model. The target user of DOBS is an individual which is familiar with workflows. This encompasses a wide range of potential users, such as management personnel, software designers, process and web service experts, etc.

The main objectives of DOBS are to assess and comply with (i) the quality of the model behavior in terms of completeness, scalability, and consistency, and (ii) the properties and quality of generated data in terms of temporal consistency, pattern coherence, and statistical properties. DOBS also targets the generation of text logs from live executions of processes and service. In the case of an existing running model, DOBS allows to check the conformance between the specification model using

its GUI, and the real system.

The *completeness* criterion requires the model execution to cover all existing transitions, in other words every component of the behavior model is to be explored. *Scalability* expresses the capacity of DOBS in performing in both a rapid and reliable way when simulating large, complex, and multiple-instance models. *Consistency* on the other hand, assures that the designed model behaves according to the specification, and does not show unexpected, undefined, or non-deterministic behaviors. *Temporal consistency* ensures for temporal data to correctly follow the time logic specified in the model. For example, timestamps are supposedly processed and logged in the required order without undesired value alteration and overlapping. *Pattern coherence* ensures that structures such as loops, parallel execution branches and time-triggered events are correctly translated into data patterns. *Statistical properties* are also to be verified in order to check that generated data values do conform to expectations, therefore increasing the probability of a positive data and behavior assessment.

We note in the end that in this Chapter we explicit units that represent the data which is logged into activity logs (traces) into the following set: **T**asks, **O**perations, **M**essages, **A**ctivities, and **E**vents. In the following, all these interaction units will be referred to as a **TOMAE**.

6.2 The DOBS system

6.2.1 Architecture

The architecture of the DOBS tool is shown on Figure 6.1 which depicts the conceptual schema of its main components: the Graphical User Interface (GUI), the Block Library (BL), the Simulator Controller(s) (SC), the Data Generator(s) (DG), the Log Pre-processing Library (LPL), and the Model Explorer (ME).

- The **GUI** component is ubiquitous and it is the starting and ending point for every simulation step. It mainly allows users to load a simulation model from a file and store it, to design and run a model from scratch by means of the Block Library(BL) and Model Explorer(ME) modules.

- The **Model Explorer (ME)** is responsible for defining and configuring all the data variables that the model is going to employ as of input, output or internal type, as well as events which are going to be triggered during runtime of the model simulation.

- The **Simulator Controller (SC)** component implements the behavior specific to the targeted model. SC has six main sub-components which are described as follows:

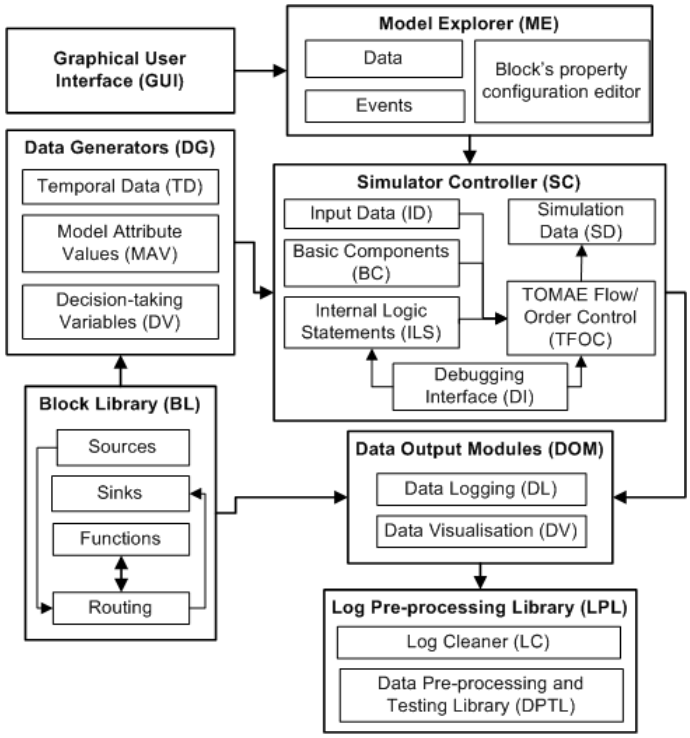


Figure 6.1: Conceptual model of DOBS

1. The Input Data (ID) receives all incoming data from outside the SC and eventually prepares them for further usage.
2. The Basic Components (BC) block provides the elementary modules that will compose the automata whose execution represents the behavior of the model. These modules include states, transitions, super-states, junction points, user-written functions, etc.
3. The Internal Logic Statements (ILS) is composed of single and optional statements such as variable instantiation, arithmetics and so on. These statements are edited and inserted directly into the transition labels of the automata, and executed if and only if the event identifying the transition is triggered and the associated condition returns the boolean **true** value. For an illustration of ILS, see Figure 6.2.
4. The TOMAE Flow/Order Control represents the core of SC. It implements the user-specified behavior by means of transition connections between states, enforcing transition constraints over determined values, probabilistic transition selection employing the input random user-defined distribution generated by DG.
5. The Debugging Interface (DI) offers functionalities that allow a quick detection of errors made during the modeling phase in DOBS.

6. The Simulation Data (SD) is in charge of all data that is of interest to the model from the designer point of view.

- The **Data Generators (DG)** module has the task of grouping all blocks whose function is to generate all the necessary data that will be used and processed during a simulation. DG is composed of three sub-components:
1. The Temporal Data (TD) provides realistic time values that are associated to TOMAE occurrences or state and transition activations in any given point of the model flowchart. DOBS uses continuous time values and the time interval can be defined by the user as finite (fixed-duration simulation) or infinite (very long duration of the simulation).
2. The Model Attribute Values (MAV) constitute the set of data that will be associated to every attribute of an existing TOMAE in the flowchart.
3. The Decision-making Variables (DV) are part of a particular, yet important group of DG. These variables bear the decision of TOMAE selection in multiple-choice scenarios. For example, when several transitions exit the same state, it will be the task of a DV to generate the value that will be used to discriminate the selected transition or TOMAE to be followed on the next simulation step.

- **The Block Library Module (BL)** provides all the elementary blocks that will compose every model. Three sub-components constitute this module. They are categorized based on their functionalities:

The *Source* module includes all blocks that are responsible for data, value and noise generation. The *Sinks* module constitutes the set of blocks acting as the output interface. The *Functions* module is the most flexible part. It is composed of both pre-defined and new user-written functions that enhance the capabilities of existing library blocks in BL.The *Routing* module is composed of blocks that channel the data and other values between the model components.
These blocks allow users to design lighter models which are not visually overloaded with simple connections that quickly overload the GUI.

- The objective of the **Data Output Module (DOM)** is to ensure the appropriate handling of the output incoming from the SC module. More precisely, its two components Data Logging (DL) and Data Visualization (DV) deal respectively with (i) recording the SC simulation data by utilizing the correct data type storage format which comes in the form of arrays, matrices, cell arrays, and symbolic values, and (ii) provide the appropriate data visualization interfaces by using either numerical displays for direct value reading, or plotting functions for observation of patterns or statistical study. Examples are depicted in Figure 6.3 and 6.4 that are described in detail in Section 6.3.

- The **Log Pre-processing Library (LPL)** constitutes the final stage of the DOBS usage flow, and deals with the crucial task of (i) cleaning logs from redundant

and other irrelevant data (c.f. Log Cleaner (LC) sub-component) and (ii) reorganizing data structures and manipulating data content in order to make it fit for further usage, as well as testing its properties in order to ensure that the output corresponds to the users' expectations before feeding the output to furthermore processing and analysis steps. Testing also includes visualization techniques, an example of which is illustrated in Fig.6.4.

An illustration of the SC block design interface is given in Figure 6.2. The diagram on this figure is a simulation model of the fictious TradingWS web service. One can see on this figure the states through which this service goes as well as the transitions that are taken upon satisfaction of the corresponding conditions. The lines of text that appear on the interface are the ILS statements.

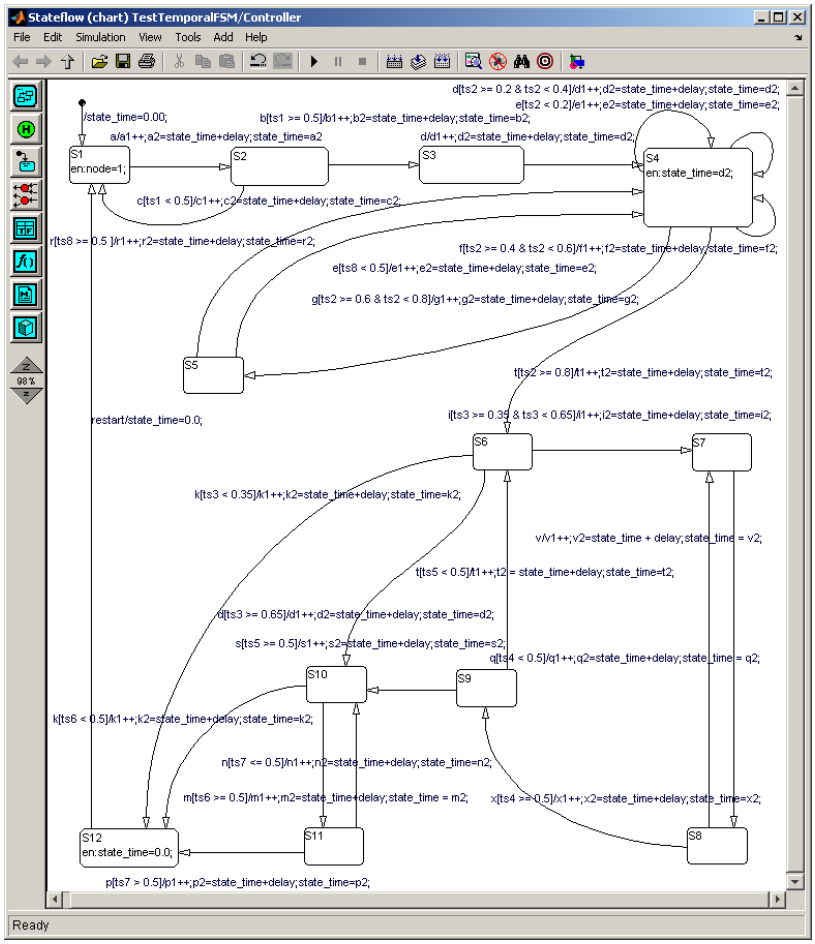


Figure 6.2: Behavior designer interface - SC component

6.2.2 Implementation

DOBS is implemented in the Matlab Simulink environment [100], and the LPL module is written in the Matlab programming language[1]. The Matlab Simulink environment was selected mainly because it has the required features for an efficient implementation of DOBS. Among these features we can mention the set of existing blocksets and toolboxes. The choice was also influenced by the opportunity to graphically design dynamic models using the Simulink environment. This software package is well known for modeling, simulating, and analyzing dynamic systems. It also allows for simultaneously simulating several models at a time which is a useful feature.

Moreover, MATLAB offers interesting features through its programming language and graphic capabilities. Its high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features, allowed for a rapid creation of the necessary functions that constitute the LPL module. Also, this software offers extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs.

The final reason for this choice is the existence of very numerous libraries that are made freely available from many academic researchers. Nevertheless, since Matlab is not a free software we also explored other candidates. Indeed, Scilab [135] offers a quite interesting option, with its Scicos code generator whose functionalities are unfortunately quite limited compared to those of Simulink, notably regarding the dynamic simulation capabilities. Octave [67] on the other hand is even more limited since it only offers a command-line interface thus allowing only for the LPL library to be tested upon it.

6.3 Experiments and testing

This section presents an experimental evaluation of DOBS based on several key criteria. The main objectives of the experiments were to check that DOBS meets its objectives provided in Section 6.1.1. We briefly recall them: (i) the quality of the model behavior in terms of completeness, scalability, and consistency, and (ii) the properties and quality of generated data in terms of temporal consistency, pattern coherence, and statistical properties.

DOBS was used to generate data for (i) the WatchMe scenario [119] in order to assess compliance restrictions (ii) the Drug Dispensation process [120] for process mining based on uncertain data, and (iii) the mockup commercial web service TradingWS

[1]The model files, along with a demo video, as well as additional information on the scenarios that are implemented using DOBS can be downloaded at http://liris.cnrs.fr/kreshnik.musaraj/technology/simulation/

Table 6.1: Performance metrics of simulated messages from TradingWS web service

# Instances	Time (sec.)	# Generated events
500	69	7485
1000	135	14778
2500	331	36080
5000	652	71207
10000	1473	132562
25000	3409	353549

(whose behavior model is given in Figure 6.2) for client service behavior simulation. For all three scenarios transition selections are randomly chosen following a uniform distribution at generation time. TOMAE inter-arrival times were independently configured in order to assure that no correlation occurred during the simulation. This is in fact a required condition for a realistic simulation.

Table 6.1 provides samples of simulations based on varying values for occurrences of entire instances and messages. One can notice the linear progression of the time required for simulation and generation versus both the number of messages and instances. This result becomes even more visible in Figure 6.5.

We show in Table 6.2 the experimental results on the selection rate of multiple transitions. This corresponds to the criteria of completeness and consistency. Indeed, the results show that all of the considered messages were executed according to the specified behavior in Figure 6.2. Moreover, the divergence between the expected selection rate and the experimental rate is quite low. This non-zero divergence corresponds to what is expected from a real execution of the model. In addition, messages **d**, **r**, **g**, and **f** have the highest level of divergence. This is due to their loop-based behavior and this provides further proof that the pattern coherence criterion is respected during simulation.

Figure 6.3 illustrates the correct temporal evolution of occurrences of TOMAEs when using a relative timeline, i.e. a clock reset to zero at the beginning of each model instance simulation. Figure 6.4 clearly shows that TOMAEs are correlated during the simulation as expected. On the right hand of the chart one sees that the dynamics of the four considered messages are indeed correlated. This derives from the structure of the TradingWS service that connects these messages. The visual pattern on the right provides further proof that the temporal constraints are not violated during log generation and transformation. Both plots were obtained using the LPL module for data cleansing and visualization. Figure 6.5 depicts the temporal performance of DOBS during generation increasing quantities of data, with the impact of both instance and event number. On the other hand, Figure 6.6 demonstrates that DOBS does preserve the statistical properties of data by comparing the parameters of

Table 6.2: Statistical metrics of simulated messages from TradingWS web service

Message type (abbreviation)	Selection rate in mutiple-choice transitions (%)	Estimated difference with expected rate
loginOK (b)	0.50556	$+5.561 \times 10^{-3}$
loginFail (c)	0.49443	-5.561×10^{-3}
browseProducts (d)	0.22849	$+28.49 \times 10^{-3}$
addToList (e)	0.20552	$+5.527 \times 10^{-3}$
order (r)	0.55121	$+51.21 \times 10^{-3}$
viewDetails (g)	0.17361	-26.39×10^{-3}
deleteFromList (f)	0.16776	-31.27×10^{-3}
confirmProductList (t)	0.20236	-2.361×10^{-3}

output data with the theoretical estimation of the statistical distribution (Exponential distribution in this case).

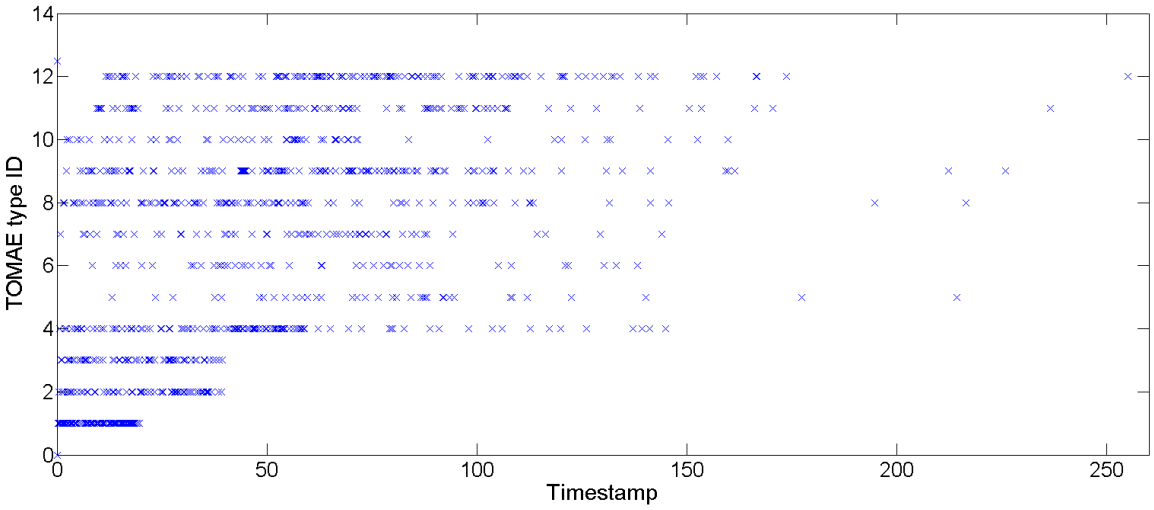


Figure 6.3: Timeline sequencing of simulated TOMAEs from the WatchMe scenario [119]

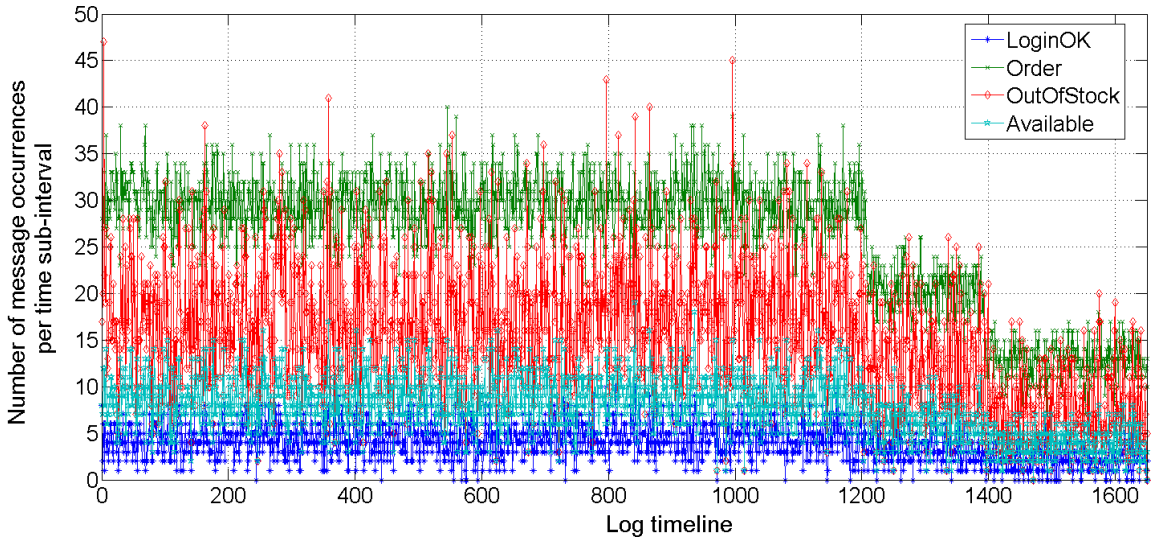


Figure 6.4: Temporal distribution of simulated messages from TradingWS web service in Figure 6.2

6.4 Related work

Existing attempts to design and build tools that might achieve similar goals to those of DOBS present limitations since they are designed to deal with particular situations, hence suffering from non-generic functionalities which severely restricted their extensive usage. Several tools address the issue of simulation and data generation of state-diagrams.

In [13] the authors present SYMIAN, a decision support tool for the improvement of incident management performance. This tool tests corrective measures for the IT support organization by improving performance measures. Since this simulation tool is targeting the performance optimization of IT management processes in a very precise manner, it is thus not possible to employ it for more generic goals such as the ones of DOBS. The main difference between DOBS and SYMIAN is the target objective: the former addresses assessment analysis and data generation, while the latter considers only performance issues. In that sense, SYMIAN can be seen as a potential application case of the more universal DOBS.

The Sage-Combinat toolbox [110] also offers interesting capabilities in exploring weighted automatons, which corresponds indeed to one of the many features of DOBS. This toolbox runs on the former MuPad application, which no longer exists since it was actually acquired by The Mathworks and incorporated into the Symbolic Math

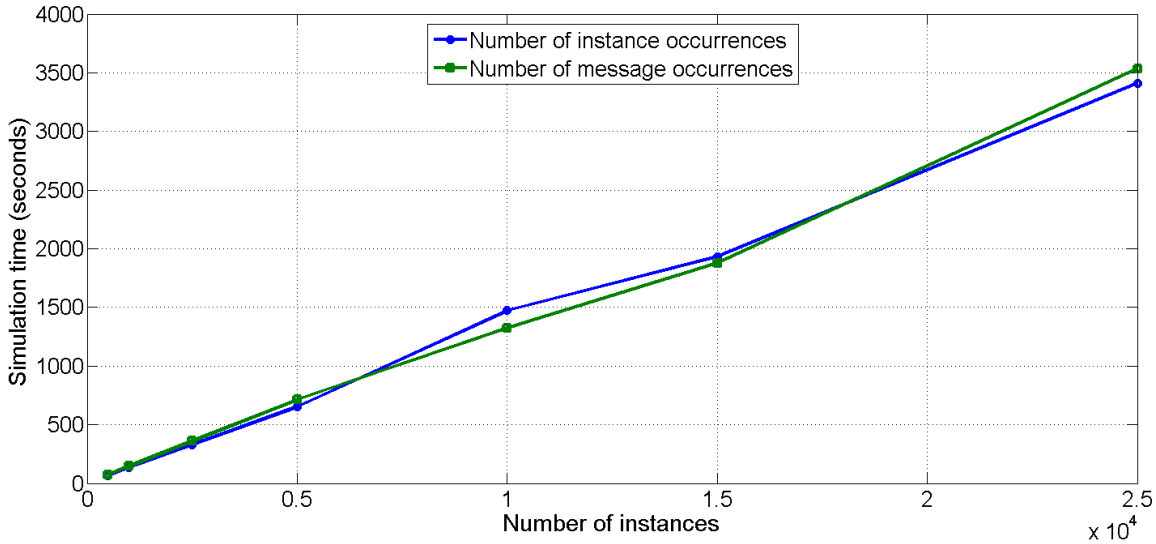


Figure 6.5: Scalability measures versus number of instances and events generated for TradingWS.

Toolbox for Matlab [100]. Yet, exploring finite-state machines is a very narrow application of Sage-Combinat, which, as an algebraic combinatorics tool, has objectives that extremely diverge from the scope of the domains considered in this part of our work. Nevertheless, this toolset has the important property of guaranteeing that all the transitions of a given automata are explored. We have shown in the previous section that this property is indeed fundamental, this is why a particular attention was given to the fact that DOBS could offer the same guarantee. Also, since the application which served as a running platform for Sage-Combinat is no longer officially available, this severely limits any future usage.

DOBS incorporates a new and innovative approach that provides for the first time a proposal and implementation framework for modeling the inner mechanisms of state diagrams in the context of processes, web services and software. Thus DOBS supports behavior analysis and data generation for these systems.

6.5 Summary

Simulation of business processes, web service business protocols, and other structures based on state diagrams for assessment analysis and data generation for mining applications is a complex task. Nevertheless achieving these objectives is very helpful for

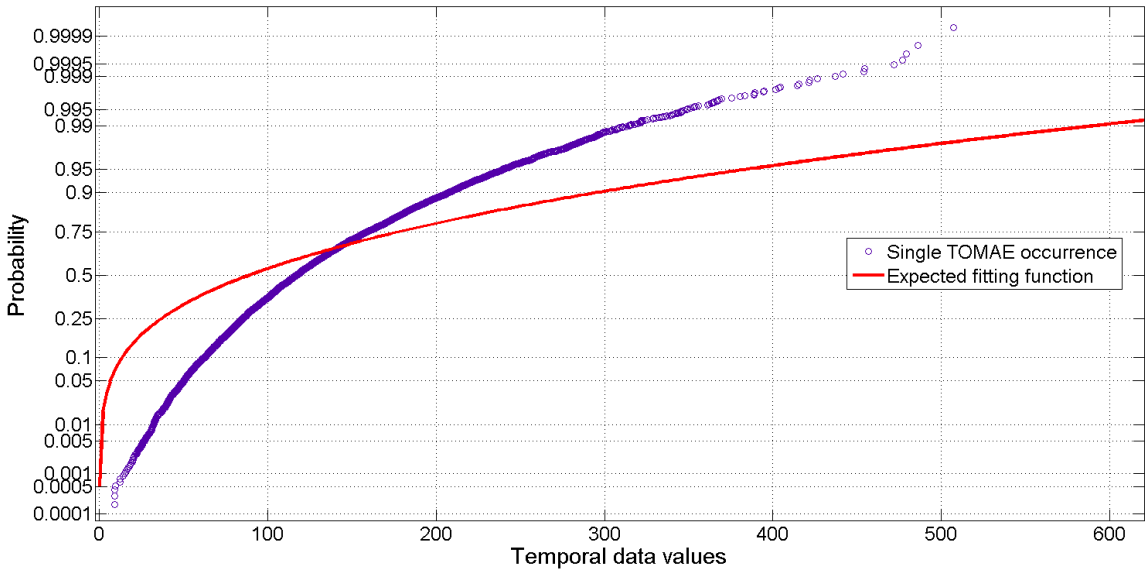


Figure 6.6: Distribution of simulated messages (blue) of TradingWS and theoretical fitting function (red)

assisting and allowing these analysis and mining applications to be tested. This chapter described and detailed the DOBS tool for the improvement and testing of model compliance and correctness and for assisting the process of mining state diagrams or flowcharts.

Chapter 7

Conclusions and future work

7.1 Concluding summary

Protocol mining by means of message correlation for web service business protocol mining is an important step in SOA architectures. This domain requires further attention since it is far from being exhaustively explored. This motivates the efforts on establishing automatic methods for message correlation relying on virtually no assumptions on the logs used as starting point. This concerns not only the properties of log data (absence of noise, statistical properties, etc.), but also the data content itself, which is subject to change in large measures depending on the context and SOA implementations. We presented a method for: (i) Correlation of messages using almost no information aside message timestamps, (ii) Modeling of the dynamic flow of messages into a business protocol, (iii) Business protocol generation from the algebraic description of the flow of messages, and (iv) A linear-complexity algorithm for the correlation and discovery process. We provided an algebraic form that is equivalent to the finite-state machine notation of a business protocol to continue on obtaining the linear system that described a business protocol by means of linear regression techniques. This was achieved via the Ordinary Least Squares estimator that was shown to be noise-resistent while providing the coefficients of the linear equations.

We continued by presenting a variable grain-size algorithm that extends the usage of temporal operators, and based on the study of cardinality properties allows the correlation between timeseries. The approach does not operate on any assumption on the existence of extracted facts and is capable of inferring temporal data facts and handling the pre-processing step. In addition, we introduced a multiple-parameter configuration that based on the theoretical contributions, allows for the extraction of the temporal graph that models the temporal relationships existing between messages. This result was built on the capability of assessing the affine transformations between

flow data density functions. Moreover, we deduced from experimental results that despite the false-appearance of potentially high complexity, the *gamma* algorithm remains robust and is influenced only by one main parameter, i.e. the granularity level.

In the context of the discovery of the timed business protocol of a Web service from its conversation logs, we focused on extracting timed transitions. Our contribution was based on a formal framework leading to the definition of proper timeouts. We have shown that proper timeouts are the best representations of timed transitions in conversation logs. We have given a simple characterization of the set of proper timeouts satisfied by the logs. We also proposed a polynomial algorithm for extracting these patterns.

Finally, we presented our contribution in simulating business processes, web service business protocols, and other structures based on state diagrams for assessment analysis and data generation for mining applications. We have shown that achieving these objectives is very helpful for assisting and allowing these analysis and mining applications to be tested. In addition, we described and detailed the DOBS tool for for the improvement and testing of model compliance and correctness and for assisting the process of mining state diagrams or flowcharts.

7.2 Future work

One of our main future work is to combine the *delta* algorithm with other methods that already provide protocol mining from conversation logs. An on-going effort is addressing the extension of the algorithm into the Business Process mining domain. In order to achieve this, the *delta* algorithm needs to be adapted to account for process-specific patterns such as parallel transitions, AND-splits, AND-joins, etc. Another perspective that we plan to investigate is the adaptation of the *delta* algorithm in order to extract cross-organizational service protocol, and to mine logs generated from composite web services.

As for the analysis of temporal series, we plan to employ similar approaches that appear to be very promising. One of these approaches is already at an advanced stage and is based on the usage of continuous functions, instead of piecewise linear functions. We intend to explore the advantages of the derivability of such functions, for determining the time shifts, and thus the temporal order between messages. Another future objective of high added value is to automatically compute the optimal values of Z_v for the different objectives (assessing a FOAT, computing the horizontal and vertical shift of PLFs, etc.).

As future work for the extraction of temporal constraints, our first objective is to broaden our method: we plan to deal with more general business protocols, with

cycles, and where transitions are not necessarily uniquely labelled, or having timed transitions entering in a final state. It would be also relevant to analyze noisy logs and to propose a probabilistic method. Another interesting prospect would be to make use of our technique to discover the entire business protocol. In fact, the result we propose to a user contains not only proper timeouts but also some local knowledge about transitions. We would like to investigate whether gathering this local information could lead to a coherent global knowledge about the protocol.

Regarding our DOBS tool, we plan to furthermore enhance it with the incorporation of more explicit compliance concerns, in order to test, for high-level semantic constraints, for example security, privacy, protocol specified actions etc.

A particular interest will be devoted to incorporating other statistical models as a basis for model design. For example, statistical distributions for the inter-arrival time between TOMAEs in a model will include the exponential and Erlang distributions which can be very helpful in designing telecommunication-based systems, extending therefore the potential applications for DOBS. Another important feature that will be added is to provide DOBS as a service which will be accessible over the web. Finally, DOBS will be completed with adapter modules in order to link its input entry point with BPEL or other object models, such as for example the BPMN Modeler for Eclipse [79]. Metrics for assessing the similarity and divergence between models are also part of future capabilities that will be integrated into DOBS.

Bibliography

[1] ABPMP, *Association of business process management professionals*, `Available at http://www.abpmp.org/ (2010)`.

[2] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann, *Mining process models from workflow logs*, EDBT '98 (Valencia, Spain), Springer, Mar 1998, pp. 469–483.

[3] Rakesh Agrawal and Ramakrishnan Srikant, *Mining sequential patterns*, ICDE '95 (Taipei, Taiwan), IEEE Computer Society, Mar 1995, pp. 3–14.

[4] Arnold O. Allen, *Probability, statistics and queueing theory, with computer science applications*, Academic Press, 1978.

[5] James F. Allen, *Maintaining knowledge about temporal intervals*, Commun. ACM **26** (1983), no. 11, 832–843.

[6] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju, *Web Services: Concepts, Architecture and Applications*, Springer Verlag, 2004.

[7] Habrard Amaury, *Modeles et techniques en inference grammaticale probabiliste : de la gestion du bruit a lÕextraction de connaissances*, Ph.D. thesis, Universite Jean Monnet de Saint-Etienne, Saint-Etienne, France, 2004.

[8] Assaf Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, and Stefano Pogliani, *Web service choreography interface (wsci) 1.0*, `W3C Note. Available at http://www.w3.org/TR/wsci/ (2002).`

[9] D. Austin, A. Barbir, C. Ferris, and S. Garg, *Web services architecture requirements*, `W3C Working Group Note. Available at http://www.w3.org/TR/wsa-reqs/ (2004).`

[10] Gerrald Baillardgeon, *Probabilites et statistiques avec applications en technologie et ingenierie*, Editions SMG, 2002, pp. 238–240.

[11] Alistair Barros, Marlon Dumas, and Phillipa Oaks, *A critical overview of the web service choreography description language.*

[12] Alistair Barros, Marlon Dumas, and Arthur H. M. ter Hofstede, *Service interaction patterns*, Business Process Management 2005 (Nancy, France), Springer, Sep 2005, pp. 302–318.

[13] Claudio Bartolini, Cesare Stefanelli, and Mauro Tortonesi, *Symian: A simulation tool for the optimization of the it incident management process*, DSOM '08: Proceedings of the 19th IFIP/IEEE international workshop on Distributed Systems: Operations and Management (Berlin, Heidelberg), Springer-Verlag, 2008, pp. 83–94.

[14] Catriel Beeri, Anat Eyal, Tova Milo, and Alon Pilberg, *Monitoring business processes with queries*, VLDB '07: Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment, 2007, pp. 603–614.

[15] Riccardo Bellazzi, Cristiana Larizza, and Paolo Magni, *Temporal data mining for the quality assessment of hemodialysis services*, Artificial Intelligence in Medicine **34** (2005), no. 1, 25–39.

[16] Boualem Benatallah, Fabio Casati, Julien Ponge, and Farouk Toumani, *Compatibility and replaceability analysis for timed web service protocols*, BDA '05 (Saint-Malo, France), Oct 2005.

[17] _____, *On temporal abstractions of web services protocols*, CAiSE '05 Short Paper Proceedings (Porto, Portugal), Springer, June 2005, pp. 39–44.

[18] Boualem Benatallah, Fabio Casati, and Farouk Toumani, *Analysis and management of web service protocols*, Conceptual Modeling - ER '04 (Shanghai, China), Springer, Nov 2004, pp. 524–541.

[19] _____, *Representing, analysing and managing web service protocols*, Data & Knowledge Engineering **58** (2006), no. 3, 327–357.

[20] Boualem Benatallah, Fabio Casati, Farouk Toumani, Julien Ponge, and Hamid R. Motahari Nezhad, *Service mosaic: A model-driven framework for web services life-cycle management*, IEEE Internet Computing **10** (2006), no. 4, 55–63.

[21] Daniela Berardi, *Automatic composition services: Models, techniques and tools*, Ph.D. thesis, Universita degli Studi di Roma, La Sapienza, Roma, Italy, 2005.

[22] Dirk Beyer, Arindam Chakrabarti, and Thomas A. Henzinger, *Web service interfaces*, WWW '05: Proceedings of the 14th international conference on World Wide Web (New York, NY, USA), ACM, 2005, pp. 148–159.

[23] Alan W. Biermann and Jerome A. Feldman, *On the synthesis of finite state machines from samples of their behavior*, IEEE Transactions on Computers **21** (1972), no. 6, 592–597.

[24] Don Box, David Ehnebuske, Gopal Kakivaya, Layman Andrew, and al., *Simple object access protocol (soap) 1.1. w3c note*, `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`.

[25] Antonio Brogi, Carlos Canal, Ernesto Pimentel, and Antonio Vallecillo, *Formalizing web service choreographies*, Electron. Notes Theor. Comput. Sci. **105** (2004), 73–94.

[26] Christoph Bussler, *B2b integration: Concepts and architecture*, Springer, 2003.

[27] Javier Cámara, Carlos Canal, Javier Cubo, and Antonio Vallecillo, *Formalizing wsbpel business processes using process algebra*, Electron. Notes Theor. Comput. Sci. **154** (2006), no. 1, 159–173.

[28] Josep Carmona, Jordi Cortadella, and Michael Kishinevsky, *Divide-and-conquer strategies for process mining*, BPM '09: Proceedings of the 7th International Conference on Business Process Management (Berlin, Heidelberg), Springer-Verlag, 2009, pp. 327–343.

[29] Fabio Casati, Malu Castellanos, Umeshwar Dayal, and Norman Salazar, *A generic solution for warehousing business process data*, VLDB '07: Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment, 2007, pp. 1128–1137.

[30] Malu Castellanos, Fabio Casati, Ming-Chien Shan, and Umesh Dayal, *ibom: A platform for intelligent business operation management*, ICDE '05: Proceedings of the 21st International Conference on Data Engineering (Washington, DC, USA), IEEE Computer Society, 2005, pp. 1084–1095.

[31] ICT Centre, *Service management framework*, `Available at http://research.ict.csiro.au/`.

[32] Samprit Chatterjee and Ali S. Hadi, *Regression analysis by example, 3rd ed.*, ch. 2, pp. 21–50, Wiley-Interscience, New York, 2000.

[33] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana, *Web services description language (wsdl) 1.1*, W3C Working Group Note. Available at `http://www.w3.org/TR/wsdl (2001)`.

[34] Jen-Yao Chung, Kwei-Jay Lin, and Richard G. Mathieu, *Guest editors' introduction: Web services computing–advancing software interoperability*, Computer **36** (2003), no. 10, 35–37.

[35] Paul R. Cohen, *Fluent learning: Elucidating the structure of episodes*, IDA' 01: 4th International Conference on Advances in Intelligent Data Analysis, 2001, pp. 268–277.

[36] William W. Cohen, *Fast effective rule induction*, In Proceedings of the 12th International Conference on Machine Learning, Morgan Kaufmann, 1995, pp. 115–123.

[37] Alberto Colombo, Ernesto Damiani, and Gabriele Gianini, *Discovering the software process by means of stochastic workflow analysis*, J. Syst. Archit. **52** (2006), no. 11, 684–692.

[38] Jonathan E. Cook and Alexander L. Wolf, *Automating process discovery through event-data analysis*, ICSE '95: Proceedings of the 17th international conference on Software engineering, ACM, 1995, pp. 73–82.

[39] Jonathan E. Cook and Alexander L. Wolf, *Discovering models of software processes from event-based data*, ACM Transactions on Software Engineering and Methodology **7** (1998), no. 3, 215–249.

[40] Francisco Curbera, Rania Khalaf, Nirmal Mukhi, Stefan Tai, and Sanjiva Weerawarana, *The next step in web services*, Commun. ACM **46** (2003), no. 10, 29–34.

[41] Sreerupa Das and Michael C. Mozer, *A unified gradient-descent/clustering architecture for finite state machine induction*, Advances in Neural Information Processing Systems 6, Morgan Kaufmann, 1994, pp. 19–26.

[42] Thomas H. Davenport, *Process innovation: reengineering work through information technology*, Harvard Business School Press, Boston, MA, USA, 1993.

[43] Booth David, Champion Michael, Ferris Chris, and McCabe Francis, *Web services architecture*, `W3C Working Draft. Available at http://www.w3.org/TR/2003/WD-ws-arch-20030514/ (2003).`

[44] Colin de la Higuera, *A bibliographical study of grammatical inference*, Pattern Recogn. **38** (2005), 1332–1348.

[45] A.K.A. de Medeiros, Wil M.P. van der Aalst, and A.J.M.M. Weijters, *Workflow mining: Current status and future directions*, On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, volume 2888 of Lecture Notes in Computer Science, Springer, 2003, pp. 389–406.

[46] Wim De Pauw, Man Lei, E. Pring, L. Villard, M. Arnold, and John F. Morar, *Web services navigator: visualizing the execution of web services*, IBM Syst. J. **44** (2005), no. 4, 821–845.

[47] Tom Debevoise, *Business process management with a business rules approach: Implementing the service oriented architecture*, BookSurge Publishing, Charleston, SC, USA, 2007.

[48] Gero Decker, Oliver Kopp, and Alistair Barros, *An introduction to service choreographies*, Information Technology **50** (2008), no. 2, 122–127.

[49] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske, *Bpel4chor: Extending bpel for modeling choreographies*, Web Services, IEEE International Conference on **0** (2007), 296–303.

[50] Giovanni Denaro, Mauro Pezzé, Davide Tosi, and Daniela Schilling, *Towards self-adaptive service-oriented architectures*, TAV-WEB '06 (Portland, Maine, USA), ACM, Jul 2006, pp. 10–16.

[51] Nirmit Desai, Ashok U. Mallya, Amit K. Chopra, and Munindar P. Singh, *Interaction protocols as design abstractions for business processes*, IEEE Trans. Softw. Eng. **31** (2005), no. 12, 1015–1027.

[52] Dider Devaurs, Kreshnik Musaraj, Fabien De Marchi, and Mohand Said Hacid, *Timed transition discovery from web service conversation logs*, 20th International Conference on Advanced Information Systems Engineering (CAISE'08), ACM, 2008, pp. 53–56.

[53] Gartner's Application Development and Maintenance Research., *Business activity monitoring: Calm before the storm. id number: Le-15-9727*, `Available at http://www.gartner.com/resources/105500/105562/105562.pdf (2002).`

[54] _____, *Note m-16-8153, the bpa market cathes another major updraft*, `Available at http://www.gartner.com (2002).`

[55] Jordan Diane and Evdemon John, *Web services business process execution language version 2.0*, `OASIS Standard. Available at http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf (2007).`

[56] Schahram Dustdar and Robert Gombotz, *Discovering web service workflows using web services interaction mining*, Int. J. Business Process Integration and Management **1** (2006), no. 4, 256–266.

[57] Schahram Dustdar, Robert Gombotz, and Karim Baïna, *Web services interaction mining*, Tech. Report TUV-1841-2004-16, Technical University of Vienna, Vienna, Austria, Sep 2004.

[58] Schahram Dustdar and Wolfgang Schreiner, *A survey on web services composition*, Int. J. Web Grid Serv. **1** (2005), no. 1, 1–30.

[59] eBay Developers Program, *The ebay trading api, available at http://developer.ebay.com/products/trading/*, 2010.

[60] Allen L. Edwards, *Introduction to linear regression and correlation*, ch. 3, pp. 20–32, W.H.Freeman & Co Ltd, San Francisco, 1976.

[61] Clarence A. Ellis, *Formal and informal models of office activity*, Information Processing, 1983, pp. 11–22.

[62] Roddick John F. and Spiliopoulou Myra, *A survey of temporal knowledge discovery paradigms and methods*, IEEE Trans. on Knowl. and Data Eng. **14** (2002), no. 4, 750–767.

[63] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth, *From data mining to knowledge discovery: an overview*, American Association for Artificial Intelligence, 1996, pp. 1–34.

[64] Diogo R. Ferreira and Daniel Gillblad, *Discovering process models from unlabelled event logs*, BPM '09: Proceedings of the 7th International Conference on Business Process Management (Berlin, Heidelberg), Springer-Verlag, 2009, pp. 143–158.

[65] Roy Thomas Fielding, *Architectural styles and the design of network-based software architectures*, Ph.D. thesis, Irvine, California, USA, 2000.

[66] Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth, *An overview of workflow management: From process modeling to workflow automation infrastructure*, DISTRIBUTED AND PARALLEL DATABASES, 1995, pp. 119–153.

[67] GNU, *Octave*, `Available at http://www.gnu.org/software/octave/`.

[68] Michael Goebel and Le Gruenwald, *A survey of data mining and knowledge discovery software tools*, SIGKDD Explorer Newsletter **1** (1999), no. 1, 20–33.

[69] Bart Goethals, *Survey on frequent pattern mining*, Manuscript, 2003.

[70] Gianluigi Greco, Antonella Guzzo, Giuseppe Manco, and Domenico Saccà, *Mining and reasoning on workflows*, IEEE Transactions on Knowledge and Data Engineering **17** (2005), no. 4, 519–534.

[71] Chunqin Gu, Hui you Chang, Yang Yi, and Sun Yat-sen, *Overview of workflow mining technology*, GRC '07: IEEE International Conference on Granular Computing, 2007, IEEE, 2007, pp. 347 – 347.

[72] Rachid Hamadi and Boualem Benatallah, *A petri net-based model for web service composition*, ADC '03: Proceedings of the 14th Australasian database conference (Adelaide, Australia), Australian Computer Society, Inc., Feb 2003, pp. 191–200.

[73] Hewlett-Packard, *Hp openview solutions*, `Available at http://www.managementsoftware.hp.com (2010)`.

[74] David Hollingsworth, *The workflow reference model*, `Available at http://www.wfmc.org/standards/docs/tc003v11.pdf (1995)`.

[75] Frank Höppner, *Knowledge discovery from sequential data*, Ph.D. thesis, TU Braunschweig, FB 1: Mathematik, Informatik, Braunschweig, Germany, January 2003.

[76] Frank Höppner and Frank Klawonn, *Finding informative rules in interval sequences*, Intelligent Data Analysis Int. Journal **6** (2002), 237–256.

[77] Hesuan Hu, Zhiwu Li, and Anrong Wang, *Mining of flexible manufacturing system using work event logs and petri nets*, ADMA '06: 2nd International Conference Advanced Data Mining and Applications, 2006, pp. 380–387.

[78] San-Yih Hwang and Wan-Shiou Yang, *On the discovery of process models from their instances*, Decision Support Systems **34** (2002), no. 1, 41–57.

[79] Intalio Inc., *Soa tools bpmn modeler*, `Available at http://www.eclipse.org/bpmn/` .

[80] Ilse C. F. Ipsen, *Numerical matrix analysis: Linear systems and least squares*, Society for Industrial and Applied Mathematics, January 2009.

[81] R.P. Jagadeesh Chandra Bose and Wil M.P. van der Aalst, *Abstractions in process mining: A taxonomy of patterns*, BPM '09: Proceedings of the 7th International Conference on Business Process Management (Berlin, Heidelberg), Springer-Verlag, 2009, pp. 159–175.

[82] Po-Shan Kam and Ada Wai-Chee Fu, *Discovering temporal patterns for interval-based events*, Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery (DaWaKÕ00, Springer, 2000, pp. 317–326.

[83] Baina Karim, Benatallah Boualem, Casati Fabio, and Toumani Farouk, *Model-driven web service development*, CAiSEÕ04: Proceedings of the 16th IntÕl Conf. Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 3084, ACM, 2004, pp. 290–306.

[84] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon, *Web services choreography description language version 1.0, w3c candidate recommendation*, Nov. 2006, `Available at http://www.w3.org/TR/ws-cdl-10/`.

[85] Justus Klingemann, Jurgen Wasch, and Karl Aberer, *Deriving service models in cross-organizational workflows*, Proceedings of RIDE - Information Technology for Virtual Enterprises (RIDE-VE '99), 1999, pp. 100–107.

[86] Ryan K. L. Ko, *A computer scientist's introductory guide to business process management (bpm)*, Crossroads **15** (2009), no. 4.

[87] Heather Kreger, *Fulfilling the web services promise*, Commun. ACM **46** (2003), no. 6, 29–ff.

[88] Mark Last, Yaron Klein, and Abraham Kandel, *Knowledge discovery in time series databases*, IEEE Transactions on Systems, Man, and Cybernetics, Part B **31** (2001), no. 1, 160–169.

[89] Neal Leavitt, *Are web services finally ready to deliver?*, Computer **37** (2004), no. 11, 14–18.

[90] Frank Leymann and Dieter Roller, *Workflow-based applications*, IBM Syst. J. **36** (1997), no. 1, 102–123.

[91] ———, *Production workflow: concepts and techniques*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[92] Chen Li, Manfred Reichert, and Andreas Wombacher, *Discovering reference models by mining process variants using a heuristic approach*, BPM '09: Proceedings of the 7th International Conference on Business Process Management (Berlin, Heidelberg), Springer-Verlag, 2009, pp. 344–362.

[93] Ming-Yen Lin and Suh-Yin Lee, *Fast discovery of sequential patterns through memory indexing and database partitioning*, J. Inf. Sci. Eng. **21** (2005), no. 1, 109–128.

[94] Weiqiang Lin, Mehmet A. Orgun, and Graham J. Williams, *An overview of temporal data mining*, 1st Australian data mining workshop (ADM02), 2002, pp. 83–90.

[95] Workflow management Coalition, *Terminology and glossary. document number wfmctc-1011*, `Available at` $http://www.wfmc.org/standards/docs/TC-1011\_term\_glossary\_v3.pdf$ `(1999)`.

[96] Heikki Mannila and Hannu Toivonen, *Discovering generalized episodes using minimal occurrences*, International Conference on Knowledge Discovery and Data Mining (KDD-96), AAAI Press, 1996, pp. 146–151.

[97] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo, *Discovery of frequent episodes in event sequences*, Data Mining and Knowledge Discovery **1** (1997), no. 3, 259–289.

[98] Elisabetta De Maria, Angelo Montanari, and Marco Zantoni, *Checking workflow schemas with time constraints using timed automata*, OTM Workshops, 2005, pp. 1–2.

[99] Florent Masseglia, Maguelonne Teisseire, and Pascal Poncelet, *Sequential pattern mining: A survey on issues and approaches*, Encyclopedia of Data Warehousing and Mining (London, UK), Information Science Publishing, 2005.

[100] The Mathworks, *Matlab and simulink*, `Available at http://www.mathworks.com`.

[101] Cantara Michele, *User survey analysis: Soa, web services and web 2.0 user adoption trends and recommendations for software vendors, north america and europe, 2005-2006*, `Available at http://www.gartner.com (2007)`.

[102] N. Mitra and Y. Lafon, *Simple object access protocol (soap) 1.2. w3c recommendation*, `http://www.w3.org/TR/soap/`.

[103] Laura Măruşter, A. J. Weijters, Wil M.P. van der Aalst, and Antal Bosch, *A rule-based approach for process discovery: Dealing with noise and imbalance in process logs*, Data Mining in Knowledge Discovery **13** (2006), no. 1, 67–87.

[104] Kreshnik Musaraj, Tetsuya Yoshida, Florian Daniel, Mohand-Said Hacid, Fabio Casati, and Boualem Benatallah, *Message correlation and web service protocol mining from inaccurate logs*, International Conference on Web Services (ICWS'10), 2010, pp. 259–266.

[105] Hamid R. Motahari Nezhad, *Discovery and adaptation of process views*, Ph.D. thesis, Computer Science and Engineering, Faculty of Engineering, UNSW, Sydney, Australia, 2008.

[106] Hamid R. Motahari Nezhad, Boualem Benatallah, Fabio Casati, and Farouk Toumani, *Web services interoperability specifications*, Computer **39** (2006), 24–32.

[107] Hamid R. Motahari Nezhad, Boualem Benatallah, Régis Saint-Paul, Fabio Casati, and Periklis Andritsos, *Process spaceship: discovering and exploring process views from event logs in data spaces*, Very Large Data Bases **1** (2008), no. 2, 1412–1415.

[108] Hamid R. Motahari Nezhad, Régis Saint-Paul, Boualem Benatallah, and Fabio Casati, *Protocol discovery from imperfect service interaction logs*, ICDE '07 (Istanbul, Turkey), IEEE, Apr 2007, pp. 1405–1409.

[109] James R. Norris, *Markov chains*, Cambrige Press, Cambridge, MA, 1998.

[110] CNRS France & others NSF USA, *Sage-combinat: Extensible toolbox for computer exploration in algebraic combinatorics*, `Available at http://wiki.sagemath.org/combinat`.

[111] Tech. Rep. OMG, *Business process modeling notation (bpmn) specification, final adopted specification*, Feb. 2006, `Available at www.bpmn.org/`.

[112] David O'Riordan, *Business process standards for web services*, 2002.

[113] Panagiotis Papapetrou, George Kollios, Stan Sclaroff, and Dimitrios Gunopulos, *Discovering frequent arrangements of temporal intervals*, ICDM '05: Proceedings of the 5th IEEE International Conference on Data Mining, 2005, pp. 354–361.

[114] Michael P. Papazoglou and Dimitrios Georgakopoulos, *Introduction: Service oriented computing*, Commun. ACM **46** (2003), no. 10, 24–28.

[115] Greg Pass, Abdur Chowdhury, and Cayley Torgeson, *A picture of search*, The First International Conference on Scalable Information Systems, 2006.

[116] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu, *Mining sequential patterns by pattern-growth: The prefixspan approach*, IEEE Transactions on Knowledge and Data Engineering **16** (2004), no. 11, 1424–1440.

[117] Chris Peltz, *Web services orchestration and choreography*, Computer **36** (2003), 46–52.

[118] Shankar R. Ponnekanti and Armando Fox, *Interoperability among independently evolving web services*, Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware (New York, NY, USA), Springer-Verlag New York, Inc., 2004, pp. 331–351.

[119] European Commision Seventh Framework Programme, *Compliance-driven models, languages, and architectures for services*, `Available at http://www.compas-ict.eu/`.

[120] ———, *Managing assurance, security and trust for services*, `Available at http://www.master-fp7.eu/`.

[121] Andy Pryke, *The data mine*, 2010, `http://www.the-data-mine.com/`.

[122] QDB09, 7th International Workshop on Quality in Databases at VLDB'09, 2009.

[123] Aubrey J. Rembert, *Comprehensive workflow mining*, ACM-SE 44: Proceedings of the 44th annual Southeast regional conference (New York, NY, USA), ACM, 2006, pp. 222–227.

[124] Chinnici Roberto, Moreau Jean-Jacques, Ryman Arthur, and Weerawarana Sanjiva, *Web services description language (wsdl) 2.0*, W3C Recommendation. Available at `http://www.w3.org/TR/wsdl20/ (2007)`.

[125] John F. Roddick, Kathleen Hornsby, and Myra Spiliopoulou, *An updated bibliography of temporal, spatial, and spatio-temporal data mining research*, TSDM '00: Proceedings of the 1st International Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining-Revised Papers (London, UK), Springer-Verlag, 2001, pp. 147–164.

[126] Sheldon Ross, *A first course in probability*, Prentice Hall, 2006, pp. 171–184.

[127] Anne Rozinat and Wil M. P. van der Aalst, *Conformance testing: Measuring the fit and appropriateness of event logs and process models*, Business Process Management Workshops, 2005, pp. 163–176.

[128] ———, *Decision mining in prom*, Business Process Management, 2006, pp. 420–425.

[129] Lucia Sacchi, Cristiana Larizza, Carlo Combi, and Riccardo Bellazzi, *Data mining with temporal abstractions: learning rules from time series*, Data Mining Knowledge Discovery **15** (2007), no. 2, 217–247.

[130] Belkacem Serrour, Daniel P. Gasparotto, Hamamache Kheddouci, and Boualem Benatallah, *Message correlation and business protocol discovery in service interaction logs*, 20th International Conference on Advanced Information Systems Engineering (CAISE'08), 2008, pp. 405–419.

[131] Glenn Shafer, *A mathematical theory of evidence*, Princeton University Press, 1976.

[132] Ricardo Silva, Jiji Zhang, and James G. Shanahan, *Probabilistic workflow mining*, KDD '05 (Chicago, Illinois, USA), ACM, Aug 2005, pp. 275–284.

[133] Michael Sipser, *Introduction to the theory of computation*, PWS, Boston, 1997.

[134] P.A Smart, Harry Maddern, and Roger S. Maull, *Understanding business process management: Implications for theory and practice*, British Journal of Management **20** (2008), no. 4, 491–507.

[135] Open Source, *Platform for numerical computation*, `Available at http://www.scilab.org`.

[136] Michael Stal, *Web services: beyond component-based computing*, Commun. ACM **45** (2002), no. 10, 71–76.

[137] Zbigniew Suraj, *Discovering concurrent process models in data: A rough set approach*, Proceedings of the 12th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC'09), Delhi, India, Springer Berlin / Heidelberg, 2009, pp. 12–19.

[138] Kishor T. Trivedi, *Probability and statistics with reliability, queueing and computer science applications*, John Wiley and sons, 2004.

[139] A.K.H. Tung, Hongjun Lu, Jiawei Han, and Ling Feng, *Efficient mining of intertransaction association rules*, Knowledge and Data Engineering, IEEE Transactions on **15** (2003), no. 1, 43–56.

[140] Mark Turner, David Budgen, and Pearl Brereton, *Turning software into a service*, Computer **36** (2003), no. 10, 38–44.

[141] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske, *Business process management: A survey*, Business Process Management, 2003, pp. 1–12.

[142] Wil M.P. van der Aalst, Kees Van Hee, Max Hee, Remmerts De Vries, Jaap Rigter, Eric Verbeek, and Marc Voorhoeve, *Workflow management: Models, methods, and systems*, MIT Press, Cambridge, MA, USA, 2002.

[143] Wil M.P. van der Aalst, Arthur H. M. Ter Hofstede, and Mathias Weske, *Business process management: A survey*, Proceedings of the 1st International Conference on Business Process Management, volume 2678 of LNCS, Springer-Verlag, 2003, pp. 1–12.

[144] Wil M.P. van der Aalst, A.J.M.M. Weijters, and Laura Maruster, *Workflow mining: Discovering process models from event logs*, IEEE Transactions on Knowledge and Data Engineering **16** (2004), no. 9, 1128–1142.

[145] Roy Villafane, Kien A. Hua, Duc Tran, and Basab Maulik, *Knowledge discovery from series of interval events*, Journal of Intelligent Information Systems **15** (2000), 71–89.

[146] Barbara von Halle, *Business rules applied: Building better systems using the business rules approach*, Wiley, Hoboken, NJ, USA, 2001.

[147] Wikipedia, *Regulatory compliance*, `Available at http://en.wikipedia.org/wiki/Compliance_(regulation)`.

[148] W. M. P. Wil 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 Knowl. Eng. **47** (2003), no. 2, 237–267.

[149] Edi Winarko and John F. Roddick, *Discovering richer temporal association rules from interval-based data*, Proceedings of the international conference on data warehousing and knowledge discovery DaWaK (Copenhagen, Denmark), 2005, pp. 315–325.

[150] Li Yingjiu, Ning Peng, Wang X. Sean, and Jajodia Sushil, *Discovering calendar-based temporal association rules*, Data Knowl. Eng. **44** (2003), no. 2, 193–218.

[151] Ernst & Young, *European fraud survey 2009*, `http://www2.eycom.ch/publications/items/fraud_eu_2009/200904_EY_European_Fraud_Survey.pdf`.

[152] Johannes Maria Zaha, Alistair Barros, Marlon Dumas, and Arthur ter Hofstede, *Let's dance: A language for service behavior modeling*, Cooperative Information Systems 2006 International Conference - CoopIS '06, vol. 4275/2006, Springer Berlin / Heidelberg, 2006, pp. 145–162.

[153] Daniel Zwillinger, *Crc standard mathematical tables and formulae, affine transformations."*, CRC Press, Boca Raton, FL, USA, 1995.